Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Erweiterung des HunFlair-Frameworks zur Erkennung genetischer Varianten in biomedizinischen Texten

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Kay Steinbauer
geboren am: 13.08.1999
geboren in: Berlin

Gutachter/innen: Prof. Dr. Ulf Leser
Prof. Dr. Alan Akbik

eingereicht am: ..... verteidigt am: .....

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In den letzten zwei Jahrzehnten gab es einen enormen Anstieg an genetischen Daten des Menschen. Das „1000 Genomes Project“ [Con15], ein Projekt, das die Genome von 2.504 Menschen aus 26 verschiedenen Populationen weltweit untersucht hat, um eine umfassende Beschreibung der genetischen Variation beim Menschen zu liefern, hat insgesamt über 88 Millionen genetische Varianten identifiziert.
Die Ergebnisse dieses Projekts wurden in den Jahren 2012 und 2015 veröffentlicht.
Daneben gibt es weitere ähnliche Projekte, wie das „The NIH Roadmap Epigenomics Mapping Consortium“ [Ber+10].
Dabei entstehen Gigabytes- bis Terrabytes an Datenmengen.
Die Medizin, insbesondere die Onkologie, hat erhebliche Fortschritte gemacht und ist heute in der Lage individuelle Behandlungen auf der Grundlage des genetischen Profils eines Menschen anzubieten [Mal+20]. In dem Artikel von Brittain et al. [BST17] wird die Behandlung von familiäre Hypercholesterinämie als wichtiges Beispiel für den Einsatz von personalisierten Behandlungsmethoden angegeben. Familiäre Hypercholesterinämie ist eine häufige erbliche Störung, die etwa 1 von 500 Personen betrifft, welche zu einem erhöhten Risiko für Herz-Kreislauf-Erkrankungen führt. Zur Behandlung werden heute individuelle Therapieansätze gewählt, die sich nach den konkreten genetischen Veränderungen eines Patienten richten. Dazu ist ein tiefes Verständnis der genetischen Varianten und ihrer Beziehung zu Krankheiten unerlässlich.
Um diese enormen Datenmengen der medizinischen Forschung zur Verfügung stellen zu können, ist es notwendig, diese Texte automatisiert zu verarbeiten. Das automatisierte Erkennen genetischer Varianten in wissenschaftlichen Texten ist nicht trivial. Häufig werden genetische Varianten in natürlicher Sprache und nicht nach standardisierten Nomenklaturen beschrieben.
Ein Beispiel aus dem BioRed [Luo+22] Korpus ist die Beschreibung „*heterozygous nucleotide (G –> C) substitution at position 1201 (exon 2) of the hGR gene*“. Nach den Richtlinien des Human Genome Variation Society (HGVS) [Dun+16] würde man dies wie folgt schreiben: „*hGR:c.1201G>C*“.
Solche Abweichungen in der Beschreibungsweise führen dazu, dass eine schlagwortbasierte Suche oder die Verwendung regulärer Ausdrücke oft zu keinem zufriedenstellenden Ergebnis führt [Wei+13]. Aus diesem Grund bietet der Einsatz von Modellen aus dem maschinellen Lernen eine nützliche Methodik, um mit den Schwierigkeiten inkonsistenter Beschreibungen umzugehen. Darüber hinaus bieten sie die Möglichkeit, die große Mengen genetischer Daten effizient zu verarbeiten. Diese Modelle lernen eigenständig, komplexe Muster und Strukturen aus Daten zu erkennen, mit denen sie trainiert werden. Man hofft, dass sich die gelernten Muster auf ungesehene Daten generalisieren lassen und somit auch bisher unbekannte genetische Variationen identifiziert werden können.

1.2 Ziel der Arbeit

Named Entity Recognition (NER) ist eine wichtige Anwendung im Natural Language Processing (NLP), bei der es darum geht, bestimmte Informationen, sogenannte *Entitäten*, aus natürlichsprachlichen Texten zu extrahieren. In dieser Arbeit liegt der Fokus auf der Erkennung genetischer Variationen. Konkret wird der NER-Tagger *HunFlair* [Web+21] so erweitert, dass neben den Entitäten *Chemikalien*, *Krankheiten*, *Zelllinien*, *Gene* und *Spezies* auch *genetische Variationen* aus biomedizinischen Texten automatisch identifiziert werden können.
Für diese Erweiterung werden drei verschiedene NER-Tagger entwickelt und getestet: Ein Modell nutzt die BiLSTM-CRF-Architektur [Lam+16], während die anderen beiden auf einem Transformer-basierten Modell [YLL22] aufbauen.
In der Arbeit wird außerdem neben dem „Wie“ (die verwendeten Modelle) auch das „Was“ (Erkennungsziel) untersucht. Das Erkennungsziel in diesem Kontext bezieht sich darauf, was spezifisch als genetische Variation identifiziert werden soll. In der Literatur wird zwischen „coarse-grained“ und „fine-grained“ NER unterschieden [Li+22]. Während coarse-grained NER die erkannten Entitäten in allgemeine Kategorien einteilt, ermöglicht fine-grained NER eine detailliertere Klassifizierung.
Betrachtet man beispielsweise den Satz „*A patient with an Arg987Ter mutation has been previously reported.*“ erwartet man von einem coarse-grained NER-Tagger, dass „Arg987Ter“ dem Label „GeneticVariant“ zugeordnet wird. Bei einem fine-grained Ansatz hingegen würde erwartet, dass „Arg“ und „Ter“ jeweils als z.B. „ProteinAminoAcid“ identifiziert werden, da sie Proteine repräsentieren, und „987“ als „MutationPosition“, da diese die Position angibt, an der die genetische Variation stattfindet. Dies stellt eine detailliertere Klassifizierung der genetischen Variation dar, da die genetischen Variation auf einer feineren Ebene betrachtet wird. Die präziseren Informationen über die Bestandteile der genetischen Variation kann es dem Modell ermöglichen, ein besseres Verständnis der Struktur zu entwickeln.
Um die entwickelten Modelle zu bewerten, werden zwei grundlegende Fragen untersucht: Wie präzise können die Modelle die Labels zuordnen? Und welchen Einfluss hat ein fine-grained NER-Ansatz auf die Erkennung der genetischen Variationen?

1.3 Genetische Variation

Die MedlinePlus definiert eine genetische Variation als „*a permanent change in the DNA sequence that makes up a gene*“[1]. Das Genom, das alle vererbbaren Informationen einer Zelle umfasst, besteht hauptsächlich aus DNA. Diese DNA kodiert genetische Informationen in Form von Nukleotidsequenzen, die aus vier Basen bestehen: Adenin (A), Thymin (T), Cytosin (C) und Guanin (G). Abschnitte dieser Sequenzen, die ein bestimmtes Protein oder eine bestimmte RNA kodieren, werden als Gene bezeichnet. Die menschliche DNA umfasst etwa 20.000 bis 25.000 solcher Gene[2]. Unterschiede in der DNA-Sequenz im Vergleich

[1] `https://medlineplus.gov/genetics/understanding/mutationsanddisorders/genemutation/`
[2] `https://www.mpg.de/6351866/genom_interaktom`

zum typischen Muster einer Art oder einer Bevölkerungsgruppe werden als genetische Variationen bezeichnet. Diese Variationen können von Veränderungen einzelner Basenpaare bis hin zu Abweichungen, die mehrere Millionen Basenpaare umfassen, reichen. Es gibt verschiedene Arten von genetischen Variationen, darunter Einzelnukleotid-Polymorphismen (SNPs), Insertionen und Deletionen (Indels), sowie Frame-Shift-Mutationen. Eine standardisierte Beschreibungsart für genetische Variationen ist die HGVS-Nomenklatur, die von der Human Genome Variation Society entwickelt wurde[Dun+16].

1.4 Themenabgrenzung

Der Fokus dieser Arbeit liegt in der automatischen Erkennung von genetischen Varianten in biomedizinischen Texten. Obwohl das Verlinken dieser Entitäten mit Datenbankidentifikatoren, bekannt als Named Entity Linking (NEL), ein wesentlicher Bestandteil des Prozesses ist – beispielsweise die Zuordnung der genetischen Variation 'c.908G>A' zur dbSNP-ID 'rs756250449', konzentriert sich diese Arbeit ausschließlich auf den Erkennungsteil.

2 Verwandte Arbeiten

Schon in den 1960er Jahren haben Wissenschaftler angefangen die Zusammenhänge von genetischen Veränderungen und ihren Auswirkungen zu dokumentieren, wie Dr. Victor A. McKusick und seine Kollegen an der Johns Hopkins University. Daraus entstand *Mendelian Inheritance in Man* (MIM), ein Nachschlagewerk für menschliche Gene und genetische Störungen, das zunächst als Buch und seit 1987 unter dem Namen *Online Mendelian Inheritance in Man*[3] (OMIM) online verfügbar ist.
Anfangs wurden diese Datenbanken manuell von Experten erstellt und verwaltet, was jedoch mit einem sehr hohen Zeitaufwand verbunden war. Zudem nimmt die Menge der biomedizinischen Texten immer stärker zu, was eine manuelle Bearbeitung zusätzlich erschwert. Der Artikel von Horn et al. [Hor+04] betont bereits im Jahre 2004 die Notwendigkeit von automatisierten Modellen zur Datenverarbeitung, da „[the] data that is published daily in the scientific literature is outstripping".

2.1 VTag

Mit *VTag* [McD+04] wurde 2004 das erste Modell zur Erkennung von genetischen Variationen in biomedizinischen Texten veröffentlicht. VTag setzt dabei auf ein Conditional Random Field (CRF) [LMP01] Modell, welches schon damals erfolgreich für andere biomedizinische Namensentität-Tagger eingesetzt wurde, z.B. [TW02].

2.2 MutationFinder

Mit *MutationFinder* [Cap+07] wurde 2007 eines der ersten Modelle dieser Art veröffentlicht. MutationFinder ist ein regelbasiertes Open-Source-System zur Identifizierung von Punktmutationen aus wissenschaftlichen Texten. Es baut auf dem Basissystem *MuteXt* [Hor+04] auf und verwendet zur Erkennung von Punktmutationen (wie z.B. A42G) mehr als 700 reguläre Ausdrücke.

2.3 tmVar

Im Gegensatz dazu ist *tmVar* [Wei+13], das erstmals 2013 veröffentlicht wurde, eines der ersten Tools, das Algorithmen des maschinellen Lernens einsetzte. Das Modell umfasst drei Hauptkomponenten: Vorverarbeitung (Tokenisierung), Mutationserkennung (CRF) und Nachverarbeitung (reguläre Ausdrücke). Eine Besonderheit ist hierbei, dass die Tokenisierung auf einer feineren Ebene durchgeführt wird, als es bei traditionellen Methoden üblich ist. Im traditionellen Ansatz erfolgt die Tokenisierung oft auf der Wortebene, wobei Leerzeichen oder Satzzeichen als Trennzeichen dienen.

[3] Verfügbar unter: `https://www.ncbi.nlm.nih.gov/omim`

2.4 tmVar3

Das *tmVar3* Modell [Wei+22] wurde 2022 veröffentlicht und erweitert den Ansatz von tmVar auf eine breitere Palette von genetische Variationen. Die Leistung von tmVar 3.0 wurde anhand von drei unabhängigen Datensätzen evaluiert und erreicht auf ihnen einen F1-Score von über 90%. tmVar3 wird außerdem von State-of-the-art genetische Variantenmodelle wie BERN2 [Sun+22] zur Erkennung und Normalisierung verwendet.

3 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für die in dieser Arbeit verwendeten Modelle und Methoden zur Implementierung eines NER-Taggers für genetische Variationen vorgestellt.
Im ersten Teil wird das NER-Problem formal vorgestellt, sowie das *Sequence Labeling*, das als methodisches Gerüst dient, um NER-Aufgaben zu lösen. Außerdem wird auf die Unterschiede und Schwierigkeiten der biomedizinischen NER eingegangen.
Im zweiten Teil geht es darum, wie für die natürlichsprachlichen Texte eine geeignete maschinelle Darstellung erstellt werden kann. Diese Repräsentation ermöglicht es den im dritten Abschnitt vorgestellten neuronalen Netzarchitekturen, darunter LSTMs und Transformer, die Texte effizient zu verarbeiten.

3.1 Named Entity Recognition

Bei der Named Entity Recognition (NER) geht es um die automatisierte Erkennung und Klassifizierung von Entitäten in natürlichsprachigen Texten. Eine „benannte Entität“ wird in der *Seventh Message Understanding Conference (MUC-7)* definiert als „*proper names and quantities of interest*“ [Chi98].
Formal lässt sich das NER Problem wie folgt beschreiben:
Sei $S = (t_1, t_2, \ldots, t_n)$ eine Sequenz von Elementen und L die Menge der vordefinierten Entitätsklassen.
Dann ist das Ziel von NER eine Liste von Tupeln (I_s, I_e, l) zurückzugeben, wobei $I_s \in [1, n]$ der Startindex und $I_e \in [1, n]$ der Endindex in der Sequenz S ist und $l \in L$ die zugeordnete Entitätsklasse ist.
Die Methoden zur automatisierten Erkennung von benannten Entitäten aus Texten lassen sich in drei Kategorien einteilen: manuell erstellte regelbasierte Methoden, Methoden basierend auf Deep Learning und Hybridmethoden [MAM08].
Die regelbasierten Methoden nutzen eine Reihe von selbstdefinierten Regeln, die auf grammatischen, syntaktischen und orthografischen Merkmalen basieren, in Kombination mit Wörterbüchern, die eine Sammlung von relevanten Wörtern enthalten. Der Nachteil dieser Methode ist, dass sie fehleranfällig ist, wenn unerwartete Variationen im Text auftreten. Sie können Schwierigkeiten haben, mit informellen Schreibweisen, Abkürzungen oder anderen unregelmäßigen Textmustern umzugehen. Außerdem ist die manuelle Erstellung dieser Regeln zeitaufwändig und kostenintensiv [KMN24].
Die ersten Ansätze, wie das NER-Problem mithilfe von Deep Learning Methoden gelöst werden kann, wurde in der Arbeit von Collobert et al. [Col+11] vorgestellt. In dieser Arbeit wird ein ähnlicher Ansatz verwendet, wobei das NER-Problem als Sequence Labeling umformuliert wird, das im Abschnitt 3.1.2 näher beschrieben wird. Der Aufbau eines auf Deep Learning basierenden NER besteht in der Regel aus 3 Komponenten: einem Embedding Modul, einem Context Encoder und einem Entity Decoder, wie in Abbildung 1 dargestellt. Das Embedding Modul verwendet Techniken, die im Abschnitt 3.2 beschrieben sind, um den Text in eine maschinenverarbeitbare Form zu bringen. Diese werden dann von den im Abschnitt 3.4 vorgestellten Deep Learning Modellen im Context Encoder

verarbeitet, um die Kontextinformationen der Wörter im Satz zu erfassen. Der Entity Decoder nimmt kontextabhängige Darstellungen als Eingabe und erzeugt eine Sequenz von Labels, die dann verwendet werden, um die Entitäten im Text zusammenzuführen. Die Hybridmethoden kombinieren Elemente der beiden Methoden. Obwohl Hybridmethoden bessere Ergebnisse als die beiden anderen Methoden liefern können, verwenden wir in dieser Arbeit dennoch eine Deep Learning basierende NER-Methode. Dies liegt daran, dass die Nachteile von regelbasierten Methoden bleiben und die manuelle Erstellung der Regeln sehr zeitaufwändig bleibt.

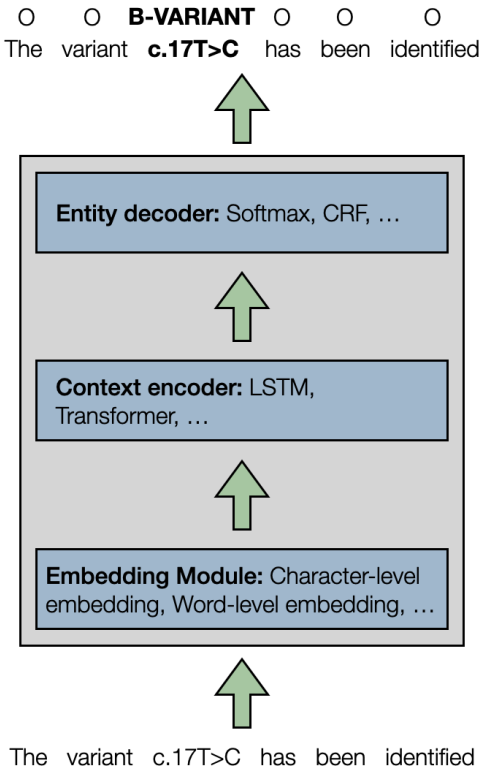

Abbildung 1: Aufbau eines Deep Learning basierten NER Models

3.1.1 Biomedizinische Named Entity Recognition

Im biomedizinischen Named Entity Recognition (BioNER) konzentrieren wir uns auf spezifische Entitäten wie Gene, Proteine und Krankheiten, wobei der Schwerpunkt dieser Arbeit darin liegt, genetische Variationen als benannte Entitäten zu erkennen.
Ein Beispiel für eine annotierte genetische Variante in einem Fließtext ist in der Abbildung 2 zu finden. Dabei wurde in dem Text die Zeichenkette „heterozygous nucleotide (G–>C) substitution at position 1201 (exon 2) of the hGr gene“ identifiziert und als genetische Variation klassifiziert.

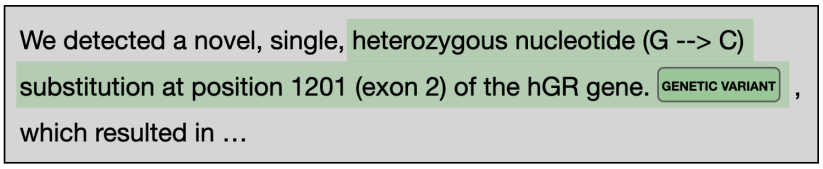

Abbildung 2: Zeigt eine annotierte genetische Variante in einem Fließtext

Die Erkennung von biomedizinischen Entitäten gilt als schwerer als das herkömmliche Named Entity Recognition [Son+21]. Die Herausforderungen liegen darin, dass zum einen die Entitäten häufig aus mehreren Wörter bestehen, unterschiedliche Schreibweisen für dieselbe Entität existieren und Entitäten je nach Kontext unterschiedliche Bedeutungen haben können. In Zhou et al. [Zho+04] wurde beispielsweise gezeigt, dass 18,6% der biomedizinischen Entitäten im GENIA V3.0 [OTK02] Korpus aus vier oder mehr Wörtern bestehen. Außerdem ist die Schreibweise der Entitäten nicht immer einheitlich, z. B. bezeichnen „5-Cholesten-3beta-ol“ und „(3beta)-cholest-5-en-3-ol“ dieselbe chemische Substanz. Es existieren dazu noch viele weitere Schreibweisen dieser Entität.
Die Kombination dieser Faktoren, macht die BioNER zu einem komplexem Problem.

3.1.2 Sequence Labeling

Das *Sequence Labeling* ist in der natürlichen Sprachverarbeitung ein Aufgabenbereich, in dem es darum geht, jedem Elemente einer Sequenz, mit einem der vordefinierten Labels zu klassifizieren. Diese Methode wird eingesetzt, um strukturelle und kontextuelle Muster in den Sequenzen zu erkennen. Formal kann das dabei wie folgt definiert werden:
Sei L die Menge der vordefinierten Labels und A ein endliches Alphabet. Dann ist $\mathcal{X} = (A^*)^*$ die Menge aller Sequenzen und $\mathcal{Y} = L^*$ die Menge aller Labelfolgen. Eine gelabelte Sequenz ist dann ein Paar S = (X, Y) mit $X \in \mathcal{X}$, $Y \in \mathcal{Y}$ und $|X| = |Y|$.
Ein Beispiel dafür ist S = (X, Y) mit X = („Montag“, „wird“, „es“, „regnen“) und Y = (DATE, O, O, O). Hier ist $L = \{DATE, O\}$ und „DATE“ steht für ein Element das eine Zeitangabe repräsentiert, während „O“ für ein sonstiges Element steht.
Ein maschinelles Lernmodell versucht dabei beim Sequence Labeling, die bedingte Wahrscheinlichkeit $P(Y|X)$ zu lernen, geben einer Menge S_{train} von Paaren aus Sequenz und zugehöriger Labelsequenz. Dabei ist $S_{all} = S_{train} \cup S_{test}$ mit $S_{train} \cap S_{test} = \emptyset$ die Gesamtmenge aller verfügbaren Paaren. Das Ziel dabei ist, dass das gelernte Modell auf der Testmenge S_{test} möglichst gute Leistung erzielt.
Der Unterschied zum NER Problem besteht darin, dass am Ende die gelabelten Elemente wieder zusammengeführt werden müssen, da Entitäten sich auch über mehrere Elemente erstrecken können. Um dies tun zu können, wird ein Annotationsschema wie das im Abschnitt 3.3 beschriebene BIO-Labeling verwendet.

3.2 Repräsentation von natürlichsprachlichen Texten

Eine geeignete maschinelle Repräsentation für natürlichsprachliche Texte zu finden, ist eine der zentralen Aufgaben der Natural Language Processing [Kor21]. Dies ist besonders

wichtig für Modelle des maschinellen Lernens. Es konnte gezeigt werden, dass die Leistungsfähigkeit eines Modells stark von der Qualität der Daten und ihrer Repräsentation abhängt, mit denen es trainiert wird, und durch diese limitiert wird [Bre+19].
Die einfachste Methode, einen Text in eine numerische Darstellung zu überführen, besteht darin, jedes Wort als Vektor mit der Dimensionalität der Vokabulargröße zu repräsentieren. Der Vektor enthält für jeden Eintrag den Wert 0, außer für die Position, die dem Wort im Vokabular entspricht; dieser Eintrag wird auf 1 gesetzt. Diese Art der numerischen Transformation wird als *One-Hot Encoding* bezeichnet. Allerdings hat One-Hot Encoding den Nachteil, dass es zu hochdimensionalen Darstellungen der Wörter führt, insbesondere bei einem großen Vokabular. Außerdem speichern die One-Hot-Vektoren keine semantischen Informationen in ihrer Darstellung, so dass es nicht möglich ist, semantische Ähnlichkeiten zwischen Wörtern durch den Vergleich der Vektoren zu erhalten.
Ein anderer Ansatz ist die *Distributed Representation*, bei der Wörter durch niedrigdimensionale, reelle dichte Vektoren repräsentiert werden, wobei jede Dimension eine semantische Eigenschaft des Wortes darstellt.
Die Aufgabe des Embedding-Moduls ist es, genau diese vektorisierte Darstellung aus dem Text zu konstruieren. Dabei unterscheidet man zwischen den Wort Embeddings und den Zeichen Embeddings.

3.2.1 Wort Embeddings

Nach dem Artikel von Almeida et al. [AX19] können Wort Embeddings definiert werden als „*dense, distributed, fixed-length word vectors, built using word co-occurrence statistics as per the distributional hypothesis*“.
Die Co-occurrence bezieht sich auf die Anzahl der Male, die zwei Wörter w_1 und w_2 in einem Korpus gemeinsam auftreten und „dicht“ heißt in dem Kontext, dass die meisten Werte im Vektor ungleich Null sind, im Gegensatz zu den One-Hot Encodings. Die Distributional Hypothesis [Har54] bildet die Grundlage für die Wort Embeddings und besagt, dass Wörter, die in einem ähnlichen Kontext vorkommen, auch eine ähnliche Bedeutung haben. Der Vorteil in dieser Annahme liegt darin, dass kein annotierter Korpus benötigt wird, um die Wort Embeddings zu erstellen, da sie basierend auf der Annahme durch die Berechnung von Co-Occurrences automatisiert generiert werden können. Hierfür werden Algorithmen wie Continuous Bag of Words und Continuous Skip-Gram Modelle eingesetzt [Mik+13].
Die Wort Embeddings sind dadurch in der Lage die semantische Bedeutung erfassen, indem sie die Beziehungen zu ihren umgebenden Wörtern in einem gegebenen Korpus berücksichtigen. Wort Embeddings lassen sich für zweidimensionale Vektoren gut visualisieren. Ein Beispiel ist in Abbildung 3 dargestellt.

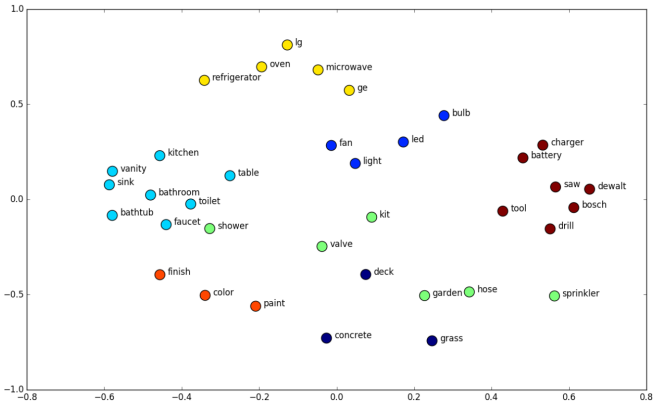

Abbildung 3: 2D-Visualisierung verschiedener Wort Embeddings. Quelle: `https://neptune.ai/blog/word-embeddings-guide`

In der Abbildung ist zu erkennen, dass für Wörter, die in ähnlichen Kontexten verwendet werden, wie 'battery' und 'charger', die Vektoren nahe beieinander liegen, während die Vektoren für Wörter, die in weniger ähnlichen Kontexten verwendet werden, wie 'kitchen', weiter auseinander liegen. Dadurch können die maschinellen Lernmodelle die semantische Ähnlichkeit zwischen zweier Worte ableiten. Dies machen sie, indem die Modelle den Abstand der Vektoren als Maß für die semantische Beziehung interpretieren.
Es gibt eine Vielzahl von Wort Embeddings, die bekanntesten sind Word2Vec, GloVe und FastText.

3.2.2 Zeichen Embeddings

Im Gegensatz zu den Wort Embeddings, erstellen die Zeichen Embeddings für jedes Wort einen eigenen Vektor, indem sie die Vektoren für jedes einzelne Zeichen des Wortes kombinieren. Die Abbildung 4 veranschaulicht, wie ein solcher Prozess ablaufen könnte.

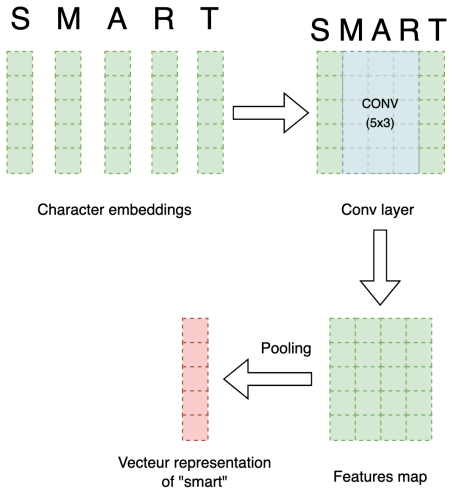

Abbildung 4: Beispiel für die Erstellung von Zeichen Embeddings. Quelle: [KMN24]

Zeichen-Embeddings bieten den Vorteil, dass sie besser mit unbekannten Wörtern umgehen können, die nicht im Trainingsvokabular enthalten sind. Zudem sind sie robuster gegenüber Tippfehlern und Variationen von Wörtern, da sie auf der Ebene von Buchstaben operieren [Li+22]. Zum Beispiel generieren die Zeichen Embeddings für ähnlich geschriebene Wörter wie „mutation“, „mutationen“ oder „muutattionen“ ähnliche Embeddings, während dies für Wort Embeddings nicht notwendigerweise gelten muss.
Ein Beispiel für solche Zeichen Embeddings sind die *FlairEmbeddings*, die durch eine BiLSTM kontextabhängige Wortrepräsentationen erzeugen, indem sie die Eingabe als eine Sequenz von Zeichen modellieren und in dieser Arbeit verwendet werden [ABV18].

3.3 BIO/IOB-Format

Das Ziel des NER ist es, jede Teilsequenz, die eine benannte Entität beschreibt, mit dem entsprechenden Entitätslabel zu klassifizieren. Nach dem Sequence Labeling erhält jedes Element ein Label. Da Entitäten jedoch aus mehr als einem Element bestehen können, ist eine spezielle Notation erforderlich, um sie zu Entitäten zusammenzuführen.
In dieser Arbeit kommt dafür das *BIO/IOB-Format* zum Einsatz, das es ermöglicht, die Grenzen und die Struktur von Entitäten innerhalb des Textes eindeutig zu kennzeichnen [RM95]. Das *BIO/IOB-Format* ist ein klassisches Annotationsschema zur Markierung von benannten Entitäten in einer Folge von Token.
Besteht eine Entität aus genau einem Element, so wird dieses mit dem Präfix *B-* gekennzeichnet. Bei Entitäten, die aus mehreren Elementen bestehen, wird das erste Element mit dem Präfix *B-* und alle weiteren mit dem Präfix *I-* gekennzeichnet. Alle anderen Elemente, die zu keiner Entität gehören, bekommen das Label *O*. Dabei steht „B“ für „Begin“, „I“ für „Inside“ und „O“ für „Outside“. Auf das Präfix folgt dann das Label der Entität. Mit Hilfe dieser Annotation können die Entitäten aus den Elementen eindeutig konstruiert werden.
Die Abbildung 5 zeigt das BIO/IOB-Format am Beispieltext „*The mutation Delta F508 is associated ... c.140G>A transition ...*“, indem die zwei genetische Varianten „*Delta F508*“ und „*c.140G>A*“ identifiziert werden.

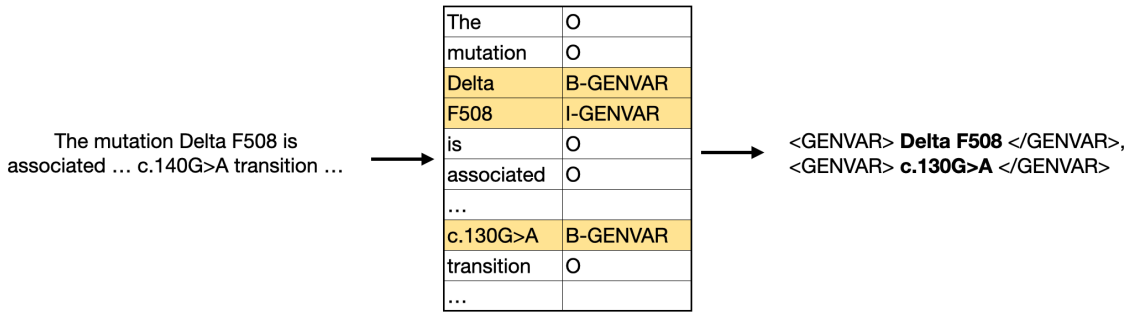

Abbildung 5: BIO/IOB-Labeling an einem Beispiel

3.4 Neuronale Netze

Deep Learning, also der Einsatz von mehrschichtigen neuronalen Netzen, hat sich in den letzten Jahren im Bereich der NER als State-of-the-Art-Methode etabliert [Raz+22]. Der große Vorteil gegenüber traditionellen NER-Methoden liegt darin, dass sie automatisiert eine abstrakte Darstellung der Textdaten lernen können, ohne auf manuelles, zeit- und kostenintensives Feature-Engineering angewiesen zu sein [Li+22].

3.4.1 Perceptron

Die Idee von selbstlernenden neuronalen Netzen basiert auf dem *Perceptron* das 1958 von Frank Rosenblatt vorgestellt worden ist [Kan03]. Dieses Modell wurde vom biologischen Neuron inspiriert, das ein elektrisches Signal ausgibt, sobald der eintreffende Reiz einen bestimmten Schwellenwert überschreitet. Perzeptronen können daher als binäre Klassifikatoren betrachtet werden, die bei Überschreiten des Schwellenwertes eine '1' und ansonsten eine '0' ausgeben. Um dies zu modellieren, wird die Heaviside-Funktion H als Aktivierungsfunktion verwendet.

$$H(x) = \begin{cases} 0 & \text{für } x < 0, \\ 1 & \text{für } x \geq 0. \end{cases} \tag{1}$$

In neuronalen Netzen wird das Konzept des Perceptrons verallgemeinert, indem beliebige Aktivierungsfunktionen zugelassen werden. Eine Aktivierungsfunktion ist eine Funktion, die die Ausgabe eines Neurons berechnet und entscheidet, wie stark ein Neuron auf die eingehenden Signale reagiert. Dadurch kann das neuronale Netz auch nicht-binäre Muster lernen. Es gibt eine Vielzahl von Aktivierungsfunktionen. Die bekanntesten sind Sigmoid, Tanh und ReLU.
Die Verbindungen zwischen den Neuronen werden durch Matrizen dargestellt, die *Gewichte* genannt werden. Mathematisch lässt sich ein Neuron durch eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ darstellen. Wobei f wie folgt definiert ist:

$$f(\mathbf{x}) = a(\mathbf{w} \cdot \mathbf{x} + b) \tag{2}$$

Hierbei wird der Gewichtsvektor $\mathbf{w} \in \mathbb{R}^n$ mit der Eingabe $\mathbf{x} \in \mathbb{R}^n$ elementweise multipliziert und der sogenannte *Bias* $b \in \mathbb{R}$ dazuaddiert. Anschließend wird auf das Ergebnis eine Aktivierungsfunktion a angewendet, um die Ausgabe zu generieren. Diese Architektur ist in der Abbildung 6 als gerichteter Graph visualisiert.

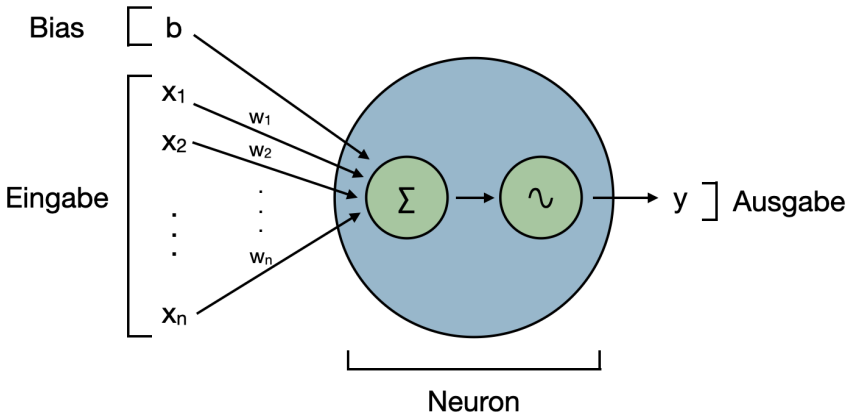

Abbildung 6: Aufbau eines Neurons

Die Gewichte werden während des Trainingsprozesses gelernt, um die Eingabe bestmöglich auf die gewünschte Ausgabe abzubilden. Das Training ist dabei ein Minimierungsproblem einer Fehlerfunktion E und ist definiert als:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L_{\mathbf{w}}(\hat{y}_i, y_i) \tag{3}$$

Die Fehlerfunktion berechnet dabei den Fehler L zwischen der tatsächlichen Ausgabe y_i und der gewünschten Ausgabe $\hat{y}_i$ für N Eingaben. Diese Art des Lernens, bei der das neuronale Netz während des Trainings durch die gewünschte Ausgabe korrigiert wird, wird als *überwachtes Lernen* bezeichnet.
Einzelne Neuronen sind in ihrer Lernfähigkeit sehr limitiert. Marvin Minsky und Seymour Papert haben schon 1969 in ihrem Buch „*Perceptrons: an Introduction to Computational Geometry.*“ [MP69] gezeigt, dass ein einzelnes Perzeptron nicht in der Lage ist eine XOR-Funktion zu lernen. Um komplexere Strukturen zu lernen, werden Neuronen miteinander kombiniert und in Schichten angeordnet. Solche mehrschichtigen Strukturen nennt man *neuronale Netze*. Es existieren verschiedene Arten von neuronalen Netzen, darunter das *Feed-Forward Neuronale Netz* und das Rekurrente Neuronale Netz (RNN).

3.4.2 Feed-Forward Neuronale Netze

Bei Feed-Forward Neuronalen Netzen, werden die Neuronen zu Schichten gruppiert und sequenziell angeordnet. Ein Neuron einer Schicht ist dabei mit allen Neuronen der nächsten Schicht verbunden. Definiert ist ein Feed-Forward Neuronales Netz f mit k-Schichten durch eine Komposition von Funktionen f_i:

$$f = f_k \circ \cdots \circ f_2 \circ f_1 \tag{4}$$

Die Ausgabe einer Schicht wird als Eingabe der nächsten Schicht verwendet. Der Funktionswert $f(x)$ ist die Ausgabe des neuronalen Netzes für die Eingabe x. Jede Funktion f_i

stellt dabei die Berechnung in der i-ten Schicht dar:

$$f_i(x) = a(w_i \cdot x + b_i) \tag{5}$$

Dabei bezeichnet w_i die Gewichtsmatrix, b_i den Biasvektor und a die Aktivierungsfunktion der i-ten Schicht. Die Funktion f_k wird die *Ausgabe Schicht* genannt und die Funktionen $f_1, ..., f_k - 1$ sind die *versteckten Schichten* des Neuronalen Netzes. In der Abbildung 7 ist als Beispiel ein 3-Schichten Feed-Forward Neuronales Netz dargestellt.

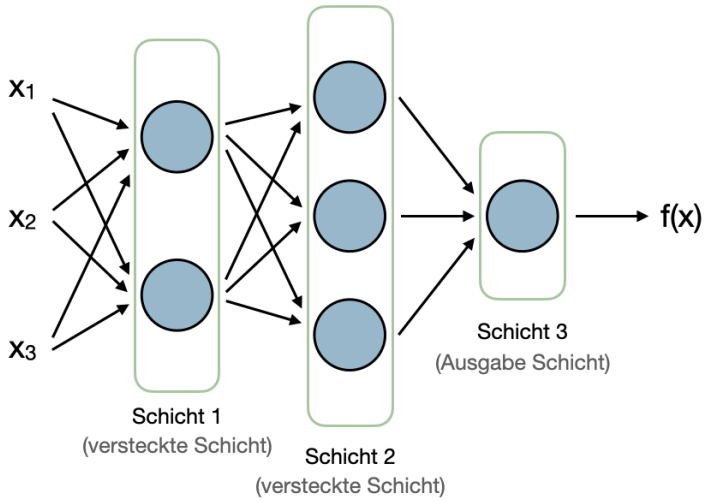

Abbildung 7: Aufbau eines Feed-Forward Neuronalen Netz
mit 3 Schichten

Das Ziel besteht darin, die Gewichte und den Bias so zu wählen, dass eine Funktion berechnet wird, die die Eingaben bestmöglich auf die gewünschte Ausgabe abbildet. Während ein einzelnes Neuron beispielsweise nicht in der Lage ist, die XOR-Funktion zu berechnen, lässt sich zeigen, dass ein Feedforward-Neuronales Netz mit mindestens einer versteckten Schicht in der Lage ist, jede Funktion beliebig genau zu approximieren. Dies ist auch bekannt als *Universal approximation theorem* und wurde 1989 von George Cybenko bewiesen [Cyb89].
Jedoch gibt es wesentliche Einschränkungen bei der Nutzung von Feed-Forward Neuronalen Netzen. Zum einen sind sie auf eine vordefinierte Eingabegröße beschränkt und können daher nicht mit Eingaben variabler Länge umgehen. Zum anderen betrachten sie jede Eingabe als unabhängig, ohne mögliche Abhängigkeiten zwischen den Datenpunkten zu berücksichtigen. Dies führt zu Problemen bei der Modellierung von sequentiellen Daten. Sequentielle Daten sind Daten, die in einer speziellen Reihenfolge angeordnet sind. Ein typisches Beispiel dafür sind Texte in einer Sprache, da zwischen den Worten eines Satzes Abhängigkeiten bestehen. Die Wörter eines Satzes bauen auf dem Kontext der vorhergehenden Wörter auf, und ihre Reihenfolge ist entscheidend für die Bedeutung des Satzes. Ändert man beispielsweise die Reihenfolge der Wörter in den Sätzen „*Die Katze*

beißt die Maus“ zu „*Die Maus beißt die Katze*“, so ändert sich auch die Bedeutung des Satzes, obwohl die Wörter gleich bleiben. Um dies bei der Verarbeitung zu berücksichtigen, bieten rekurrente neuronale Netze eine effektive Lösung.

3.4.3 Rekurrente Neuronale Netze

Rekurrente neuronale Netze (RNN) sind neuronale Netze, die dafür ausgelegt sind mit sequentiellen Daten zu arbeiten. Dies wird durch Rückkopplungsschleifen erreicht, die es dem Netzwerk ermöglichen, Informationen von einem Schritt der Verarbeitung zum nächsten weiterzugeben. Dabei wird dem RNN in jedem Verarbeitungsschritt die Ausgabe des vorhergehenden Schritts sowie das nächste Element in der Eingabesequenz gegeben. Da die Eingabe elementweise nacheinander verarbeitet wird, ist es außerdem in der Lage, mit Eingaben variabler Länge umzugehen.
Um die Funktionsweise von RNNs zu verstehen, siehe das Beispiel in Abbildung 8:

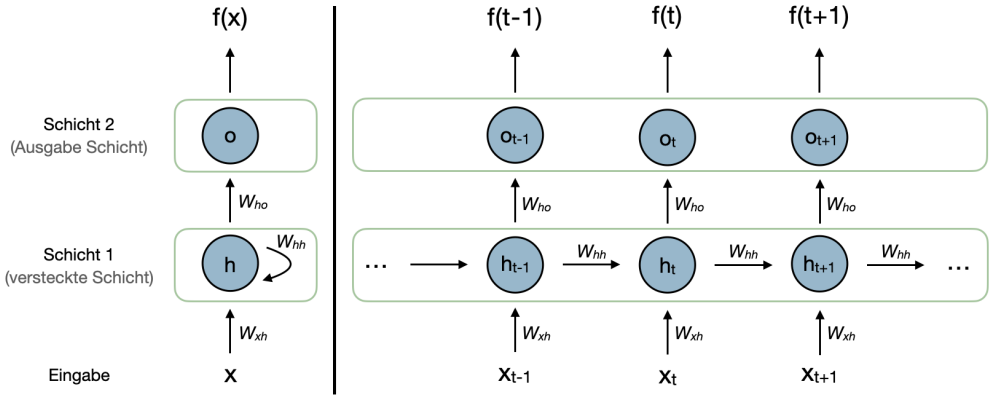

Abbildung 8: Einfaches rekurrentes neuronales Netz

Auf der Abbildung 8 sind zwei Darstellungsvarianten eines Rekurrenten Neuronalen Netzwerks zu sehen. Links ist das RNN in seiner kompakten Form dargestellt. Dieses Netzwerk enthält in der ersten Schicht ein Neuron, das eine Rückkopplungsschleife besitzt und die Ausgabe an die nächste Schicht weitergibt, die die Ausgabe berechnet. Auf der rechten Seite der Abbildung wird das RNN zeitlich 'entfaltet' dargestellt, wobei jeder Verarbeitungsschritt über die Zeit hinweg einzeln visualisiert wird. Konkret berechnet das RNN in jedem Verarbeitungsschritt t den Zustand des Neurons h und die Ausgabe $f(t)$ des Neurons o. Für eine Eingabesequenz $x_1, ..., x_T$ wird die Teilsequenz $x_1, ..., x_t$ bis zu einem bestimmten Zeitpunkt $t < T$ verarbeitet. Der Zustand des Neurons h im Schritt t wird dabei wie folgt berechnet:

$$h_t = a_1(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{6}$$

Die Ausgabe des RNNs $f(t)$ wird dann aus h_t anschließend so berechnet:

$$f(t) = a_2(W_{ho}h_t + b_o) \tag{7}$$

Dabei sind:

- a_1 und a_2 die Aktivierungsfunktion.
- h_t der Zustand des Neurons h zum Zeitpunkt t.
- W_{hh}, W_{xh} und W_{ho} die Gewichtsmatrizen.
- b_h, b_o die Bias-Vektoren.

In RNNs werden die Gradienten durch *Backpropagation Through Time* (BPTT) berechnet. Bei langen Sequenzen neigen diese Gradienten dazu, entweder zu verschwinden (werden sehr klein) oder zu explodieren (werden sehr groß).

3.4.4 Long Short-Term Memory (LSTM)

Das Long Short-Term Memory Modell (LSTM) [HS97] wurde 1997 in der Arbeit von Sepp Hochreiter und Jürgen Schmidhuber mit der Motivation vorgestellt, eine Lösung für das Problem der verschwindenden und explodierenden Gradienten von RNN zu finden. LSTMs sind eine Variante von RNNs, die das Netzwerk zusätzlich zum *hidden state* um einen *cell state* erweitern. Außerdem werden 'Gating'-Funktionen eingeführt, die den Informationsfluss innerhalb des Netzwerkes steuern, indem sie entscheiden, welche Informationen behalten und welche vergessen werden sollen. Dies ermöglicht es LSTMs Informationen über einen längeren Zeitraum zu behalten.
Eine LSTM besteht aus einer Reihe von LSTM-Zellen, die miteinander verbunden sind und die Eingabe sequenziell verarbeiten. Dabei übergibt jede LSTM-Zelle ihren *hidden state* und den *cell state*, die den internen Zustand der LSTM-Zelle erfassen, an die nachfolgende LSTM-Zelle weiter.
In jedem Verarbeitungsschritt t einer Eingabe $x = x_1 ... x_T$ werden in einer LSTM-Zelle die folgenden Berechnungen durchgeführt:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \\
g_t &= \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g) \\
o_t &= \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Dabei bezeichnet i das *input gate*, f das *forget gate*, o das *output gate*, c den *cell state* und h den *hidden state*. Die Sigmoid-Funktion σ, definition als $\sigma(x) = \frac{1}{1+e^{-x}}$, gibt Werte zwischen 0 und 1 zurück und ist für die Aktivierung der Gates verantwortlich. Der *cell state* ist eine Kombination aus Information von verganenen Verarbeitungsschritten und neuer Information, gesteuert durch das *forget gate* und das *input gate*. Der *hidden state* h_t ist die Ausgabe der LSTM-Zelle.

Die Abbildung 9 zeigt den Aufbau einer LSTM-Zelle.

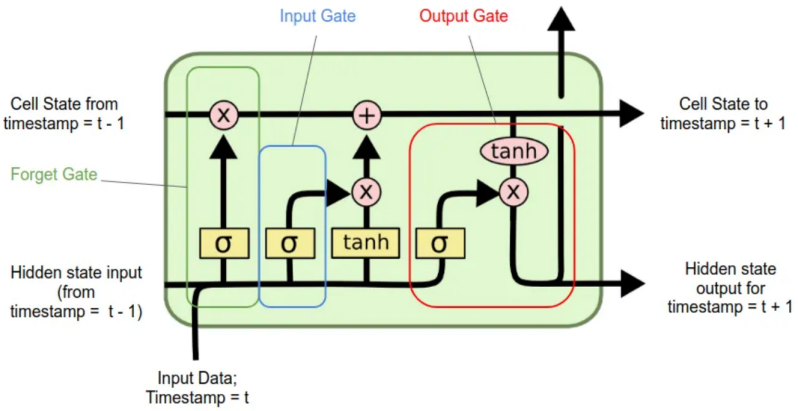

Abbildung 9: LSTM-Zelle

Ein Nachteil des rekursiven Ansatzes ist, dass er eine sequentielle Verarbeitung der Eingaben erfordert und somit keine Parallelisierung zulässt. Außerdem haben LSTMs trotz ihrer speziellen Struktur Schwierigkeiten Informationen über längere Zeiträume zu speichern und sehr langfristige Abhängigkeiten in Daten effektiv zu erfassen [Zha+20]. Ein möglicher Lösungsansatz für diese Problematik bietet der Attention-Mechanismus [DB14]. Dieser Mechanismus ermöglicht es, Wörter, die in einer Sequenz weit voneinander entfernt sind, über kürzere Netzwerkpfade miteinander zu verbinden, was bei RNNs oder LSTMs weniger effizient möglich ist.

3.4.5 Transformer

Die im Paper „Attention Is All You Need“ von Vaswani et al. [Vas+17] vorgestellte Transformer-Architektur ist ähnlich wie Rekurrente Neuronale Netze ein Modell zur Verarbeitung sequentieller Daten, verzichtet jedoch vollständig auf rekurrente Strukturen und setzt stattdessen auf den Self-Attention-Mechanismus zur Erfassung von Abhängigkeiten zwischen den Elementen der Eingabesequenz. Der Self-Attention-Mechanismus funktioniert dabei so, dass jedes Element der Eingabesequenz durch einen Query-Vektor, einen Key-Vektor und einen Value-Vektor repräsentiert wird und dann berechnet wird, wie relevant jedes Element (durch den Query-Vektor) für ein anderes Element (durch den Key-Vektor) ist. Diese Relevanz wird als Score ausgedrückt. Anschließend werden diese Scores genutzt, um die Value-Vektoren zu gewichten. Durch das Aufsummieren dieser gewichteten Value-Vektoren erhält man dann den finalen, kontextualisierten Vektor für jedes Element, der die gesamte Eingabesequenz berücksichtigt.
Der Self-Attention-Mechanismus lässt sich wie folgt berechnen:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{8}$$

Dabei ist K der Key-Vektor, Q der Query-Vektor und V der Value-Vektor. Das Skalarprodukt der Query- und Key-Vektoren wird durch die Wurzel der Dimension des Key-Vektors d_k geteilt, um die Skalierung zu normalisieren. Anschließend wird die Softmax-Funktion auf diese Werte angewendet, um die normalisierten Gewichte dann mit dem Value-Vektor zu kombinieren. *Multi-Head Attention* bezeichnet mehrere Self-Attention Komponenten, die als *Attention-Heads* bezeichnet werden, die parallel in verschiedenen Schichten laufen.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{wobei head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{9}$$

Jeder Attention-Head head_i wird durch die Anwendung der Attention-Funktion auf die gewichteten Vektoren QW_i^Q, KW_i^K, VW_i^V gebildet. Die Ausgaben aller Attention-Heads werden dann zusammengefügt und mit einer weiteren Gewichtsmatrix W^O gewichtet. Durch die Aufteilung der Berechnung kann sich jeder Attention-Head auf verschiedene Beziehungen innerhalb der Eingabesequenz konzentrieren. Die Abbildung 10 stellt den Aufbau des Multi-Head Attention Mechanismus graphisch dar.

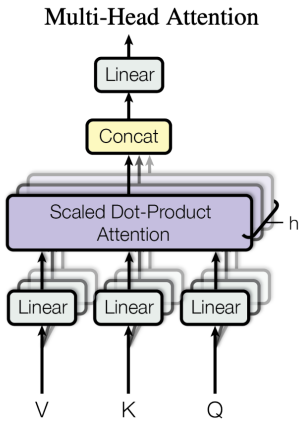

Abbildung 10: Multi-Head Attention [Vas+17]

Diese Multi-Head Attention Komponenten bilden in Kombination mit einem Feed-Forward Netzwerk einen sogenannten *Transformerblock*. Diese Transformerblöcke sind ein wichtiger Bestandteil der Encoder- und Decoder-Module des Transformers. Im Encoder wird die Eingabe verarbeitet und eine kontextualisierte Repräsentation der Daten generiert. Diese erzeugte kontextualisierte Repräsentation wird dann im Decoder mit der bis dahin erzeugten Ausgabesequenz kombiniert, um die Ausgabe zu generieren. Encoder und Decoder bestehen dabei aus mehreren Schichten von Transformerblock die hintereinander geschaltet werden. Die Abbildung 11 zeigt den Aufbau der Transformer-Architektur.

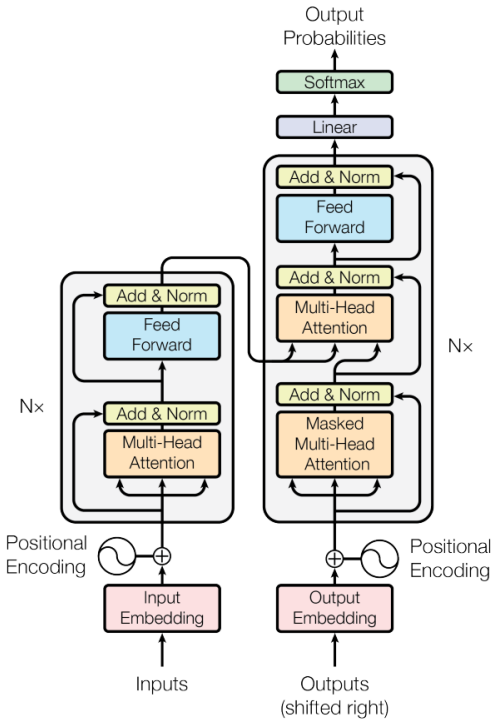

Abbildung 11: Die Transformer Architektur [Vas+17]

In der Architektur, wie sie von Vaswani et al. [Vas+17] vorgestellt wurde, werden jeweils sechs solcher Transformerblöcke im Encoder und Decoder verwendet, wobei jeder Block acht Attention-Heads enthält. Diese Anzahl ist jedoch nicht festgesetzt und variiert in verschiedenen Implementierungen der Transformer-Architektur.
Der Vorteil des Transformers gegenüber RNNs und LSTMs ist die parallele Verarbeitung der Sequenzen durch die Verwendung des Attention-Mechanismus. Auf diese Weise kann das Modell gleichzeitig auf alle Sequenzelemente zugreifen, so dass auch bei langen Sequenzen keine Informationen verloren gehen.

4 NER-Modelle

In diesem Kapitel werden die Modelle vorgestellt, die für die Entwicklung eines NER-Taggers zur Erkennung genetischer Variationen verwendet wurden. In den letzten Jahren haben sich Deep-Learning-Methoden als State-of-the-Art-Methode zur Lösung von NER-Aufgaben erwiesen [KT21].
Dazu wurden basierend auf dem HunFlair NER-Tagger, drei spezialisierte Modelle entwickelt. Das erste Modell verarbeitet annotierte, vollständige genetische Variationen und verwendet ein BiLSTM-CRF Modell zur Erkennung der Entitäten. Das zweite und dritte Modell nutzt einen Transformer-basierten Ansatz, speziell BioLinkBERT. Dabei verarbeitet das zweite Modell ebenfalls vollständige genetische Variationen, während das dritte Modell in Subklassen zerlegte genetische Variationen erhält. Die ersten beiden Modelle implementieren einen sogenannten „coarse-grained“ NER-Ansatz, bei dem es darum geht, genetische Variationen als Ganzes zu erkennen, während das dritte Modell einen „fine-grained“ NER-Ansatz verwendet, bei dem die genetischen Variationen in spezifischere Subklassen zerlegt werden und das Modell dann darauf trainiert wird, diese zu identifizieren.
Der Abschnitt 4.1 beschreibt die allgemeine Architektur von HunFlair, während die Abschnitte 4.1, 4.2 und 4.4 detailliert auf die einzelnen Modelle eingehen.

4.1 Beschreibung der HunFlair Architektur

HunFlair [Web+21] ist ein biomedizinischer NER-Tagger, der dafür spezialisiert wurde, die Entitäten *Chemikalien*, *Krankheiten*, *Zelllinien*, *Gene* und *Spezien* automatisiert aus Texten zu identifizieren. Er basiert auf dem 2019 veröffentlichten HUNER-Tagger und integriert dessen Ansatz in das Flair [Akb+19] NLP-Framework. Dabei wird für jeden Entitätstyp ein eigener NER-Tagger auf der Vereinigung aller Goldstandard-Korpora für diesen Entitätstyp trainiert. Im Gegensatz zu traditionellen NER-Taggern werden dabei keine manuell erstellten Features verwendet, deren Entwicklung zeitaufwändig und spezifisches Expertenwissen erfordert, wie sie z.B. in dem in [HXY15] vorgestellten NER-Tagger zum Einsatz kommen. Zusätzlich werden verschiedene Zeichen- und Wort Embeddings verwendet und miteinander kombiniert. Die Kombination verschiedener Embeddings kann zu einer besseren Repräsentation der semantischen und syntaktischen Eigenschaften von Texten führen, da jedes Embedding unterschiedliche Aspekte abdeckt [ABV18]. In [Lam+16] werden beispielsweise Zeichen Embeddings verwendet, um Unterschiede in der Schreibweise von Wörtern zu erfassen und mit Wort Embeddings kombiniert, um den Kontext der verwendeten Wörter zu verstehen.
Die Architektur von HunFlair und die Integration in das Flair-NLP Framework wird in Weber et al. [Web+21] detailliert beschrieben.

4.2 Modell 1: BiLSTM-CRF Tagger

Das erste Modell zur Implementierung eines genetischen Variaten NER-Tagger baut auf der BiLSTM-CRF Architektur auf, wie sie in [Lam+16] beschrieben wird. Ein Bidirektionales LSTM (BiLSTM) stellt eine Erweiterung der in Abschnitt 3.4.4 beschriebenen LSTM-Architektur dar und wird in [GS05] beschrieben. Im Gegensatz zu herkömmlichen LSTMs, verarbeitet ein BiLSTM die Eingabe nicht nur von links nach rechts, sondern auch von rechts nach links. Diese bidirektionale Verarbeitung erweist sich als sinnvolle Erweiterung, da, wie in Keraghel et al. [KMN24] beschrieben wird, rekurrente Modelle dazu neigen, Elemente, die näher am Ende der Sequenz sind, unverhältnismäßig stärker zu gewichten als Elemente, die weiter entfernt davon sind. Durch die Verwendung eines BiLSTM wird dieser Effekt ausgeglichen.
Die BiLSTM-Architektur wird in diesem Modell durch eine zusätzliche CRF-Schicht erweitert, die dazu dient, die Labels vorherzusagen. In [CL19] wurde gezeigt, dass es für Sequence-Labelling-Aufgaben, bei denen starke Abhängigkeiten zwischen den Labels bestehen, es besser ist, die Labels in Kombination mit einer CRF zu generieren, da diese die Beziehungen der Labels untereinander besser modelliert können.
In diesem Modell wird eine BiLSTM mit 256 versteckten Schichten verwendet, was es zu einem Deep-Learning-Ansatz macht. Ein Nachteil dieser vielschichtigen Modelle ist ihre Abhängigkeit von großen, qualitativ hochwertigen, annotierten Datensätzen, auf denen das Modell trainiert werden kann, so genannten Goldstandard-Korpora [Du+16].
Eine mögliche Lösung für dieses Problem ist der Einsatz von *Transfer Learning* [GB18]. Transfer Learning ist eine Methode im maschinellen Lernen, bei der ein Modell für einen Aufgabenbereich trainiert wird und dann wiederverwendet oder angepasst wird, um eine andere, aber verwandte Aufgabe zu lösen. Es werden dabei zwei Arten von Transfer Learning unterschieden: Feature-basiertes Transfer Learning und Fine-Tuning. In diesem Modell wird ein Feature-basierter Ansatz verfolgt, bei dem vortrainierte Embeddings als Eingabe für das BiLSTM-CRF-Modell verwendet werden. Hierbei wird während des Trainingsprozesses ausschließlich das BiLSTM-CRF-Modell auf die spezifische Aufgabe hin trainiert, ohne dabei die vortrainierten Embeddings zu verändern.
In diesem Modell werden dabei die Flair-Embeddings in Kombination mit dem klassischen Wort-Embedding FastText[4] verwendet, basierend auf den Empfehlungen aus [ABV18]. Die Architektur ist in Abbildung 12 dargestellt, wie sie in in [Hab+17] detailliert beschrieben wird.

[4] `https://fasttext.cc/docs/en/crawl-vectors.html`

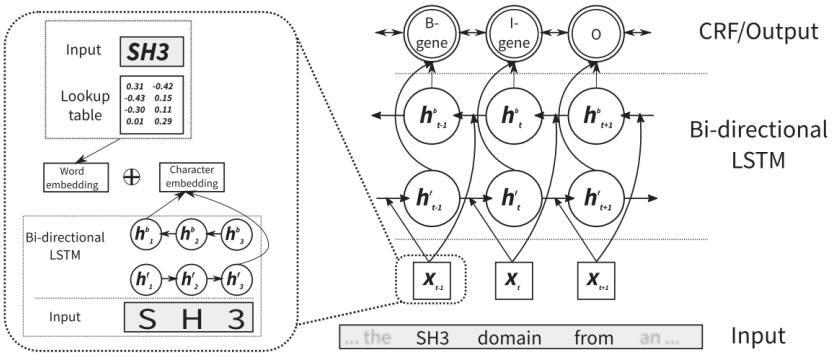

Abbildung 12: Architektur der BiLSTM-CRF aus [Hab+17]. Die Eingabe wird der Embedding Schicht übergeben, die eine vektorisierte Darstellung erzeugt und sie an die BiLSTM übergibt. Die BiLSTM erzeugt dabei für jedes Eingabewort x_t zwei kontexualisierte Repräsentation h_t^f und h_t^b, die konkatiniert werden und der CRF-Schicht übergeben wird.

4.3 Modell 2: BioLinkBERT Tagger

Das zweite Modell basiert auf der im Abschnitt 3.4.5 vorgestellten Transformer-Architektur. In diesem Modell wird der Fine-Tuning Ansatz des Transfer Learning angewendet, indem ein vortrainiertes Modell als Grundlage verwendet wird, das dann spezifisch für die Erkennung genetischer Varianten umtrainiert wird. Der Unterschied hierbei zum Feature-basierten Transfer Learning ist, das beim Fine-Tuning das gesamte vortrainierte Modell angepasst wird, während beim Feature-basierten Ansatz die vortrainierten Embeddings unverändert bleiben. Konkret wurde das verwendete Modell auf 21 GB Abstracts von PubMed[5], einer biomedizinischen Datenbank, vortrainiert. In den letzten Jahren hat sich gezeigt, dass mit vortrainierten Sprachmodellen in verschiedenen NLP-Bereichen State-of-the-Art-Ergebnisse erzielt werden können. Zum Beispiel wurde in der Arbeit von Carielli et al. [CA21] gezeigt, dass ein Transformer-basierter Ansatz BiLSTM- und CRF-basierte Modelle auf verschiedenen BioNER-Korpora bei der Erkennung biomedizinischer Entitäten übertrifft.
In diesem Modell wird BioLinkBERT [YLL22] verwendet, um einen NER-Tagger zu konstruieren. Dieses Modell basiert auf der in Devlin et al. [Dev+19] vorgestellten BERT (Bidirectional Encoder Representations from Transformers) Architektur. BERT ist ein bidirektionales Transformer-Modell und verwendet im Gegensatz zur ursprünglichen Transformer-Architektur, die sowohl einen Encoder als auch einen Decoder enthält, nur die Encoder-Komponente. Außerdem ist BERT in der Lage ist, bidirektionale Repräsentationen von Daten zu lernen. Dies wird durch die Verwendung einer 'Masked Language Model' Trainingsmethode erreicht, das einen Teil der Eingabetexte zufällig maskiert und das Modell dann darauf trainiert, diese maskierten Wörter basierend auf ihrem Kontext zu erraten. Diese Methode ermöglicht es dem Modell, den Kontext von

[5] `https://pubmed.ncbi.nlm.nih.gov`

beiden Seiten eines Wortes zu verstehen.
Die verwendeten Modell-Hyperparameter entsprechen der in Devlin et al. [Dev+19] beschriebenen $BERT_{Base}$-Konfiguration. Dabei werden 12 Transformerblöcke verwendet, wobei jeder Block 12 Attention-Heads nutzt. Außerdem wird für die Größe der versteckten Schichten eine Dimension von 768 gewählt.
Darüber hinaus wird eine Vortrainingsmethode verwendet, die es dem Modell ermöglicht, den Kontext über Dokumentgrenzen hinweg zu berücksichtigen, indem Verbindungen zwischen Dokumenten, wie z.B. Hyperlinks, in den Trainingskorpus integriert werden. Diese Trainingsmethode wird in Carielli et al. [CA21] beschrieben.

4.4 Modell 3: Fine-grained BioLinkBERT Tagger

Das dritte Modell basiert ebenfalls auf dem Transformer-basierten BioLinkBERT und unterscheidet sich vom zweiten Modell dadurch, dass es einen fine-grained NER-Ansatz verwendet. Die Verwendung eines fine-grained NER-Ansatzes bedeutet, dass das Modell darauf ausgerichtet ist, spezifische Subklassen der Entitäten zu erkennen. Im Vergleich zu den ersten beiden Modellen, die einen coarse-grained NER-Tagging-Ansatz verwenden, gibt es zwei Herausforderungen bei der Erstellung eines fine-grained NER-Taggers: die Auswahl der Entitätsklassen und die Erstellung der Trainingsdaten [Li+22]. Die Entitätsklassen wurden in Anlehnung an die im tmVar-Paper beschriebenen Kriterien [Wei+13] ausgewählt. Daraus wurden die folgenden 10 Entitätsklassen herausgearbeitet:

- *Reference sequence type* (Der Typ der Referenzsequenz: c,g,r,m,p)
- *Exon/intron* (Gibt an ob die Variation in einem Exon oder einem Intron liegt: IVS, Intron, Ex, Exon)
- *Mutation type* (Der Typ der Variation: z.B. del, ins, dup, tri, delins, indel)
- *Frame shift mutation* (Gibt an das die Variation eine Verschiebung des Leserasters von Genen auf der DNA verursacht: fs, fsX, fsx)
- *DNA/RNA nucleotide* (Die betroffenen Nukleotide: z.B. A, T, G, C)
- *Protein amino acid* (Die betroffenen Aminosäuren: z.B. glutamine, glu)
- *Mutation position* (Die Position der Variation)
- *Snp id* (ID für eine spezifische Single Nucleotide Polymorphisms)
- *Mutation component* (Die übrigen Komponenten der Variation, die nicht in die oben genannten Kategorien eingeordnet werden konnten)

Für das Training eines Modells, das diese spezifischen Entitätsklassen erkennen kann, sind passende Trainings- und Testdaten erforderlich. Da solche Daten in der spezifischen Segmentierung nicht vorhanden sind, wurden die bereits für das erste und zweite Modell verwendeten Korpora als Grundlage für den Trainings- und Testdatensatz genutzt. Dazu

wurden vier spezifische reguläre Ausdrücke entwickelt und auf diese Korpora angewendet, um die Datenpunkte in die gewünschten Subklassen zu zerlegen. Da die verwendete Notation in den Korpora nicht immer mit der HGVS-Nomenklatur übereinstimmt, wurden die regulären Ausdrücke aus der in [Dun+16] beschriebenen Notation entwickelt und durch Beobachtungen aus den verwendeten Korpora ergänzt. Ein Beispiel für einen der entwickelten regulären Ausdrücke ist der folgende:

```
(REF)? NUC_RANGE NUC? OP NUC
```

Dieser reguläre Ausdruck dient zur Erkennung von Substitutionen, Deletionen, Duplikationen, Insertionen und Inversionen die auf der DNA-/RNA-Ebene beschrieben werden und ist in verschiedene Teilausdrücke unterteilt.
Die Teilausdrücke sind wie folgt definiert:

```
REF         := [cgrm]\.
NUC         := [ATGCatgcu]+
OP          := [>|indel|delins|del|ins|inv|at|...]
POS         := ([\+]\-]? \d+|?)
NUC_POS     := ((POS)? [\+|\-]? POS | \* POS | POS \*)
NUC_RANGE   := NUC_POS (\_ NUC_POS)?
```

Der Ausdruck beginnt mit der optionalen Angabe eines Referenzsequenztyps. Der Typ kann entweder eine kodierende DNA-Sequenz (c), eine genomische Sequenz (g), eine RNA-Sequenz (r), eine mitochondriale Sequenz (m) oder eine Proteinsequenz (p) sein. Dann folgt eine Positionsangabe der betroffenen Stelle. Dies kann entweder als spezifische Basenposition erfolgen oder mithilfe des „-“ Operators als relative Position in Bezug auf eine angegebene Basenposition. Zur Angabe eines Positionsbereichs wird der Operator „_“ verwendet. Für eine relative Positionierung zum Stopcodon wird das Zeichen „*“ und für eine unbekannte Position das Zeichen „?“ verwendet. Dann folgt die Angabe der zu verändernden Nukleotiden, gefolgt von einem der Operatoren, die durch Symbole wie „>“„indel“ oder „delins“ dargestellt werden. Zum Schluss wird eine Nukleotidfolge angegeben, die die ursprüngliche Folge ersetzt.
Ein Beispiel für die Anwendung dieses regulären Ausdrucks ist die Zerlegung der genetischen Variation „*c.1930C>T*“, die in der Abbildung 13 dargestellt ist.

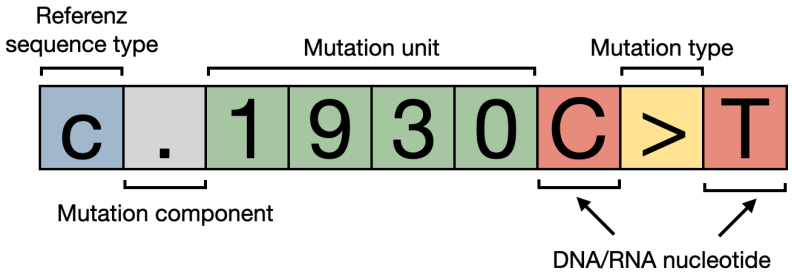

Abbildung 13: Zerlegung von ’c.1930C>T’ in die Entitätsklassen

Das „c“ wird dabei als der Referenzsequenztyp erkannt und „1930“ als die spezifische Position innerhalb der kodierenden DNA-Sequenz. Das „C“ und das „T“ sind die betroffenen Nukleotide und „>“ der Substitutionsoperator. Da der „.“ in keine der anderen Entitätsklassen eingeordnet werden kann, wird er als *Mutation component* klassifiziert.
Im nachfolgenden Kapitel werden Statistiken präsentiert, die zeigt, wie umfassend die entwickelten regulären Ausdrücke die genetischen Variationen aus den Korpora erfassen. Außerdem sind alle weiteren regulären Ausdrücke im Anhang 8 mit einer kurzen Beschreibung zu finden.
Ein weiterer Nebeneffekt dieser feineren Zerlegung der Entitäten ist, dass eine spätere Normalisierung der Entitäten erleichtert wird. Bei der Entitätsnormalisierung, auch Named Entity Linking (NEL) genannt, werden Entitäten mit Einträgen in einer externen Wissensdatenbank verknüpft. Eine Schwierigkeit besteht darin, dass eine Entität verschiedene Formen haben kann, da sie neben dem vollständigen Namen auch Teilnamen, Aliasnamen und Abkürzungen haben kann. Darüber hinaus kann eine Bezeichnung je nach Kontext eine andere Entität bezeichnen. Bei der eindeutigen Zuordnung von Entitäten kann eine feinere Aufgliederung der Entität hilfreich sein.

5 Experimente

In diesem Kapitel wird die Trainingsmethode für die im Kapitel 4 beschriebenen NER-Tagger vorgestellt, sowie die verwendeten Datensätze und die Evaluationsmetriken beschrieben. Jedes der drei NER-Modelle wird separat auf spezifischen Korpora trainiert und anschließend auf denselben Korpora getestet. Für den fine-grained BioLinkBERT Tagger wurden die Ausgaben angepasst, so dass die Ergebnisse mit den anderen vergleichbar sind. Die Evaluation der Modelle basiert auf Metriken wie Recall, Precision und F1-Score, wobei das Matching-Kriterium *exact match* verwendet wurde.
Der zugehörige Code ist auf GitHub unter folgendem Link verfügbar:
`https://github.com/rn-kay99/hunflair_tagger`.

5.1 Daten für die BiLSTM-CRF und BioLinkBERT Tagger

Um Datensätze für das Training und den Test des BiLSTM-CRF NER-Taggers (siehe Abschnitt 4.2) und des BioLinkBERT NER-Taggers (siehe Abschnitt 4.3) zu erhalten, wurden 20 biomedizinische Korpora auf genetische Variationen untersucht. Unter den untersuchten Korpora haben sich die Korpora *BioRed* [Luo+22], *tmVar* [Wei+22] und *Osiris* [Fur+08] als passend herausgestellt und werden als Datenbasis für die weiteren Experimente verwendet. Während BioRed Annotationen zu verschiedenen Entitäten wie genetischen Varianten, Genen, Krankheiten, Chemikalien, Spezien und Zelllinien enthält, hat tmVar3 Annotationen zu genetischen Variationen, Genen und Krankheiten. Osiris enthält Annotationen zu Genen und genetischen Variationen. Um die Vergleichbarkeit zu gewährleisten, wurden alle Annotationen, die nicht genetische Variationen repräsentieren, entfernt.
Jeder Korpus wurde in einen Trainingsdatensatz und einen unabhängigen Testdatensatz für die Auswertung unterteilt. Zusätzlich wurde ein Validierungsdatensatz erstellt, wobei zufällig ausgewählte 10% des Trainingsdatensatzes verwendet wurden. Der Validierungsdatensatz wird während des Modelltrainings verwendet, um zu beurteilen, ob das Modell sich verbessert oder verschlechtert hat. Die Aufteilung zwischen Trainingsdatensatz und Validierungsdatensatz ist dabei über die Experimente hinweg unverändert, um sicherzustellen, dass das Modell unter denselben Bedingungen evaluiert wird.
Die Abbildung 14 gibt eine Übersicht über die Größe und Verteilung der Korpora BioRED, tmVar3 und Osiris in Trainings-, Test- und Validierungsdatensätze.
Das Aufteilungsverhältnis in Trainings- und Testdatensätze für die Korpora tmVar3 und Osiris wurde manuell festgelegt. Für tmVar3 wurde ein Aufteilungsverhältnis von 80/20 gewählt, während bei Osiris, aufgrund der geringeren Größe des Korpus, eine 60/40 Aufteilung verwendet wurde, um die Daten zwischen Training und Test zu teilen. Im Gegensatz dazu wurde die vorhandene Aufteilung des BioRed-Korpus übernommen. Die Einteilung der Daten zwischen Training und Test erfolgte zufällig, um sicherzustellen, dass in den Datensätzen die Daten ähnlich verteilt sind. Außerdem wurde darauf geachtet, dass es keine Überschneidungen zwischen Trainings- und Testdatensätzen gibt.

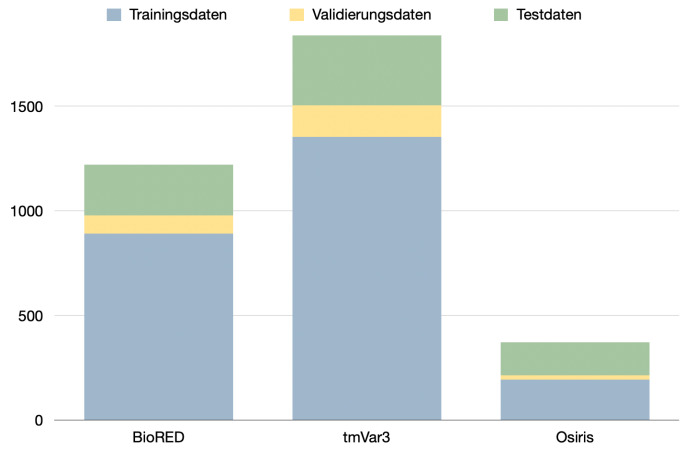

Abbildung 14: Übersicht über die Korpusgröße und die Verteilung in Trainings- und Test- und Validierungsdatensätze der Korpora BioRED (801 Trainingssätze, 89 Validierungssätze, 241 Testsätze), tmVar3 (1354 Trainingssätze, 150 Validierungssätze, 335 Testsätze), Osiris (192 Trainingssätze, 21 Validierungssätze, 159 Testsätze).

5.2 Daten für den fine-grained BioLinkBERT Tagger

Damit das fine-grained BioLinkBert Modell 4.4 auf Subklassen von genetischen Variationen trainiert werden kann, ist ein angepasster Datensatz erforderlich. Dafür wurden die im Abschnitt 4.4 beschriebenen regulären Ausdrücke entwickelt und auf die oben beschriebenen Korpora angewendet.
Die Tabelle 1 gibt einen Überblick über den Umfang des angepassten Datensatzes. Die Spalte „Korpusgröße“ gibt die Gesamtzahl der genetischen Variationen im Korpus an, während die Spalte „Abdeckung“ die Anzahl der mit den regulären Ausdrücken geparsten Variationen angibt. Die „Abdeckungsrate“ gibt das prozentuale Verhältnis zwischen Korpusgröße und Abdeckung an.

	BioRED	**tmVar3**	**Osiris**
Abdeckung	1062	1310	242
Korpusgröße	1381	1839	372
Abdeckungsrate	76,90%	71,23%	65,05%

Tabelle 1: Abdeckung der regulären Ausdrücke auf den Korpora

Genetische Variationen, die nicht durch die Anwendung der regulären Ausdrücke zerlegt werden konnten, wurden als Ganzes der Entitätsklasse „Mutation component“ zugeordnet und dem Datensatz hinzugefügt. Dieses Vorgehen wurde gewählt, um die Gesamtgröße der Korpora beizubehalten, insbesondere aufgrund der bereits begrenzten Größe der

Datensätze.
Auf dem BioRed-Korpora wurde eine höhere Abdeckungsrate auf als auf tmVar3 und Osiris erzielt, da es in diesen Korpora verhältnismäßig mehr Annotationen von genetischen Variationen gibt, die in natürlicher Sprache verfasst oder umschrieben wurden. Dies macht sie schwerer mit regulären Ausdrücken zu parsen.
Die Tabelle 2 gibt einen Überblick über die Anzahl der Datenpunkte pro Entitätsklasse, die nach der Anwendung der regulären Ausdrücke auf die Korpora erhalten wurden.

Entitätstyp	**BioRED**	**tmVar3**	**Osiris**
Reference sequence type	179	266	11
Exon/intron	8	33	5
Mutation type	295	585	98
Frame shift mutation	20	28	0
DNA/RNA nucleotide	563	1010	288
Protein amino acid	663	1067	107
Mutation position	698	1172	135
Snp id	160	134	24
Mutation component	528	1014	162

Tabelle 2: Verteilung der verschiedenen Entitätstypen

Ein häufig auftretendes Problem bei der fine-grained Named Entity Recognition ist das *data sparseness problem*. Dies tritt auf, da die Anzahl der Entitätsklassen bei der fine-grained Named Entity Recognition wesentlich größer ist und somit die Wahrscheinlichkeit größer ist, dass nicht genügend Trainingsdaten für jede spezifische Entitätsklasse vorhanden ist. In diesem Fall wurden beispielsweise im Osiris-Korpus keine Frame-Shift-Mutation-Komponenten erfasst, da dieser Korpus keine Frame-Shift-Mutationen enthält. Darüber hinaus gibt es im Osiris-Korpus kaum genetische Variationen mit Angabe des Referenzsequenztyps, d. h. des Typs der Basensequenz oder des Referenzgenoms, auf die sich die genetische Variation bezieht.

5.3 Trainingsdetails

Zum Trainieren der NER-Modelle wird ein iterativer Algorithmus verwendet, der sich *Mini-Batch Gradient Descent* nennt. Beim Mini-Batch Gradient Descent wird der Trainingsdatensatz in kleine Teilmengen von Datenpunkten aufgeteilt, die sich *Mini-Batches* nennen. Die Festlegung der Mini-Batch-Größe sowie der Anzahl vollständiger Iterationen über den gesamten Trainingsdatensatz, auch als *Trainingsepochen* bezeichnet, sind Parameter, die vor dem Training festgelegt werden müssen. In jeder Trainingsepoche wird dann der Fehler für jeden Datenpunkt im Mini-Batch berechnet, indem der Fehler zwischen den vorhergesagten Ausgaben des Modells und den tatsächlichen Labels verglichen wird. Die Modellparameter werden dann entsprechend angepasst, um den Fehler zu verringern. Zur Berechnung des Fehlers verwendet das BiLSTM-CRF-Modell den *Viterbi Loss* als

Verlustfunktion, während das BioLinkBERT und das Fine-grained BioLinkBERT-Modell den *Cross-Entropy Loss* als Verlustfunktion verwenden. Der Cross-Entropy-Loss für einen Datenpunkt lässt sich dabei wie folgt berechnen:

$$L(y, \hat{y}) = -\sum_{i=1}^{M} y_i \cdot \log(\hat{y}_i) \tag{10}$$

M ist die Anzahl der Entitätsklassen und $\hat{y}_i$ die Ausgabe des Modells für die Klasse i und y_i der tatsächliche Wert. Der Fehler wird berechnet, indem die negative logarithmische Wahrscheinlichkeit der korrekten Klasse summiert wird. Der Logarithmus in der Cross-Entropy-Loss-Funktion bewirkt, dass unsichere Vorhersagen des Modells stärker gewichtet werden. Dies geschieht, indem größere Unterschiede nahe 1 mit höherer Gewichtung berücksichtigt werden, während kleinere Unterschiede nahe 0 weniger stark ins Gewicht fallen.
Für die Bestimmung der Anzahl der Trainingsepochen wird eine Early-Stopping-Methode verwendet, bei der das Training automatisch endet, wenn keine signifikanten Verbesserungen mehr im Trainingsverlust zu beobachten sind oder die vorher festgelegte maximale Anzahl von Epochen erreicht wurde.
Zur Bestimmung der Hyperparameter, darunter maximale Epochen, Lernrate und die Mini-Batch-Größe, wurde die Methode des *Random Search* angewendet. Dabei wurden verschiedene Kombinationen von Hyperparametern zufällig aus einem vordefinierten Bereich ausgewählt und auf dem Validierungsdatensatz getestet.
Die Tabelle 3 enthält die Werte der verwendeten Hyperparameter sowie die vordefinierten Bereiche, aus denen die zufällig ausgewählten Werte ausgewählt wurden.

Hyperparameter	Wert	Vordefinierter Bereich
Maximale Epochen	200	[20, 100, 200, 300]
Lernrate	$2e^{-5}$	[$2e^{-2}$, $2e^{-3}$, $2e^{-4}$, $2e^{-5}$, $2e^{-6}$]
Mini-Batch-Größe	16	[8, 16, 32]

Tabelle 3: Hyperparameter des Modells mit vordefinierten Bereichen

Die eckigen Klammern geben dabei an, dass die Werte aus dem angegebenen diskreten Wertebereich ausgewählt wurden.
Um eine verlässliche Bewertung der Modellleistungen zu erhalten, wurde jedes Modell fünfmal auf einem Korpus trainiert und getestet. Aus diesen Werten wurden dann der Mittelwert und die Standardabweichung für die Metriken Precision, Recall und F1-Score berechnet. Der Mittelwert gibt dabei die durchschnittliche Leistung über die Durchläufe hinweg an, während die Standardabweichung angibt, wie stark die Ergebnisse streuen.
Eine häufige Ursache für eine schlechte Performance eines maschinellen Lernmodells ist *Overfitting*. Overfitting tritt auf, wenn das Modell während der Trainingsphase zu sehr auf die Daten optimiert wurde und dadurch zwar auf den Trainingsdaten gut abschneidet, aber schlecht auf ungesehene Daten generalisieren kann. Um dies zu vermeiden, werden

beim Training von NER-Modellen verschiedene Techniken eingesetzt, wie z.B. *Shuffling*. Dabei werden die Mini-Batches nach jeder Epoche in eine zufällige Reihenfolge gemischt, um damit zu vermeiden, dass das Modell möglicherweise speziell auf die Reihenfolge der Daten trainiert wird.

5.4 Evaluationsmetriken

Um die Qualität der Modellvorhersagen zu bewerten, werden die Metriken *Recall*, *Precision* und *F1-Score* berechnet und miteinander verglichen, die aus den *True Positives*, *False Positives* und *False Negatives* ermittelt werden.
In diesem Kontext werden die Fälle, in denen das Modell eine Tokensequenz als genetische Variation annotiert und diese Annotation korrekt ist, als *True Positives* bezeichnet, während die falschen Klassifikationen die *False Positives* sind. Ein *False Negative* tritt auf, wenn das Modell eine Tokensequenz nicht als genetische Variation annotiert, obwohl diese eine solche beschreibt.
Der Recall lässt sich dann definieren als das Verhältnis der korrekt identifizierten genetischen Variationen (True Positives) zur Gesamtanzahl der tatsächlichen genetischen Variationen, einschließlich derjenigen, die fälschlicherweise vom Modell nicht erkannt wurden (True Positives + False Negatives):

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{11}$$

Die Precision ist definiert das Verhältnis der korrekt klassifizierten genetischen Variationen (True Positives) zur Gesamtanzahl der vom Modell als genetische Variationen klassifizierten Token, einschließlich derjenigen, die fälschlicherweise als solche erkannt wurden (True Positives + False Positives):

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{12}$$

Während der Recall misst, wie effektiv das Modell die Entitäten identifizieren kann, gibt die Precision an, wie viele der vom Modell als genetische Variationen klassifizierten Entitäten tatsächlich korrekt sind.
Der F1-Score ist das harmonische Mittel aus Precision und Recall und ist nützlich, da es für ein Modell einerseits wichtig ist, möglichst viele Fälle zu erfassen (Recall) und gleichzeitig die Anzahl der falsch positiven Ergebnisse zu minimieren (Precision). Er wird definiert als:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

Der Algorithmus 1 beschreibt konzeptionell die Berechnung dieser Werte im Pseudocode. Dafür werden die Arrays *predictedValues* und *goldValues* so aus einer Liste von Modellannotationen bzw den tatsächlichen Annotationen konstruiert, sodass sie elementweise miteinander verglichen werden können. Weitere Einblicke in die Konstruktion dieser

Arrays gibt der Algorithmus 3 der im Anhang zu finden ist.
Um zu entscheiden, wann eine Vorhersage korrekt ist, muss ein Matching-Kriterium bestimmt werden. In den Experimenten wird eine Vorhersage des Modells als korrekt gewertet, wenn die vorhergesagten Entitätsgrenzen genau mit den Begrenzungen der tatsächlichen Entität im Text übereinstimmen. Dies bedeutet, dass nicht nur der Startpunkt, sondern auch der Endpunkt der vorhergesagten Entität genau den Start- und Endpunkt der tatsächlichen Entität im Text entsprechen müssen. Zusätzlich dazu muss auch der vorhergesagte Entitätstyp exakt mit dem tatsächlichen Entitätstyp übereinstimmen. Diese Art der Bewertung wird als *exact match* bezeichnet.
Die Verwendung dieses Matching-Kriteriums hat den Vorteil, dass zum einen sichergestellt werden kann, dass das Modell genau die Entitäten lernt, wie sie in den Trainingsdaten definiert werden und dass zum anderen die Vergleichbarkeit der Ergebnisse mit anderen Modellen erleichtert wird. Daneben gibt es jedoch noch viele weitere Matching-Kriterien, wie *Left match*, *Right match* oder *Partial match* [Tsa+06].

Algorithm 1 Berechnung von Precision, Recall und F1-Score für die Modellvorhersagen

procedure EVALUATE(predictedValues, goldValues)
    $true\_positives \leftarrow 0$
    $false\_positives \leftarrow 0$
    $false\_negatives \leftarrow 0$
    for i **in** length($predictedValues$) **do**
        if $goldValues[i] ==$ „genetic_variant“ $== predictedValues[i]$ **then**
            $true\_positives \mathrel{+}= 1$
        else if $goldValues[i] ==$ „genetic_variant“ $\neq predictedValues[i]$ **then**
            $false\_negatives \mathrel{+}= 1$
        else
            $false\_positives \mathrel{+}= 1$
        end if
    end for
    $recall \leftarrow \frac{true\_positives}{true\_positives + false\_negatives}$
    $precision \leftarrow \frac{true\_positives}{true\_positives + false\_positives}$
    $f1\_score \leftarrow \frac{2 \cdot (precision \cdot recall)}{precision + recall}$

    return $recall, precision, f1\_score$
end procedure

5.5 Evaluation der fine-grained und coarse-grained Ergebnisse

Um die Ergebnisse des fine-grained BioLinkBERT Modells mit den der anderen Modellen vergleichen zu können, ist es notwendig, die Annotationen in ein gleichartiges Format zu bringen. Die Annotationen der fine-grained und coarse-grained Tagger unterscheiden sich dabei wie folgt:

Gegeben sei ein Text $T = \{t_1, \ldots, t_n\}$, wobei t_i das i-te Token repräsentiert.
Eine Annotation ist ein Paar (X, y), wobei $X = (x_1, \ldots, x_n)$ eine Tokenfolge ist und x_i das i-te Token in T bezeichnet.
Der coarse-grained Tagger, also das BiLSTM-CRF und BioLinkBERT Modell, annotieren eine Tokenfolge mit $y \in Y = \{genetic\_variant\}$, wobei der fine-grained Tagger eine Tokenfolge mit $y \in$ Y' = {reference_sequence_type, ...} (siehe Abschnitt 4.4 für vollständige Auflistung der Entitätsklassen) annotiert. Dabei besteht eine coarse-grained annotierte Entität im Allgemeinen aus mehreren fine-grained annotierten Entitäten.
Um die verschiedenen Arten von Annotationen miteinander vergleichen zu können, müssen die fine-grained annotierten Entitäten wieder zusammengeführt werden. Dieser Vorgang wird Algorithmus 2 beschrieben.

Algorithm 2 Zusammenführung der fine-grained annotierten Entitäten

```
1:  procedure MergeFineGrainedEntities(inputFile)
2:      tokens ← readTokens(inputFile)
3:      predictedEntities ← []
4:      currentEntity ← []
5:      for token in tokens do
6:          if getAnnotation(token) ≠ O then
7:              if length(currentEntity) == 0 then
8:                  setAnnotation(token, B-genetic_variant)
9:              else
10:                 setAnnotation(token, I-genetic_variant)
11:             end if
12:             currentEntity.append(token)
13:         else
14:             if length(currentEntity) > 0 then
15:                 predictedEntities.append(currentEntity)
16:                 currentEntity ← []
17:             end if
18:         end if
19:     end for
20:     if length(currentEntity) > 0 then
21:         predictedEntities.append(currentEntity)
22:     end if
23:     return predictedEntities
24: end procedure
```

Dafür liest der Algorithmus zunächst die annotierten Token aus einer Eingabedatei ein und iteriert über die Tokens.
Auf Token-Ebene wird ein Token zusätzlich zum Entitätstyp mit einem Präfix versehen, der angibt, ob es sich um den Anfang einer Entität (B-), einer Fortsetzung einer Entität (I-) oder um keine Entität (O) handelt (hierbei wird das in Abschnitt 3.3 beschriebene

BIO-Labeling verwendet). Das erste Token mit einer anderen Annotation als „O“ wird mit „B-genetic_variant“ annotiert. Die darauf folgenden Token werden mit „I-genetic_variant“ annotiert. Dies bedeutet, dass die fine-grained Entitätstyp-Informationen wie „mutationUnit“ verworfen werden und durch die Oberklasse „genetic_variant“ ersetzt werden. Diese Token werden dann dem *currentEntity* Array hinzugefügt, das dazu dient aufeinanderfolgende Tokens, die genetischen Variationen entsprechen, zu einer gemeinsamen Entität zusammenzuführen. Wenn ein Token mit der Annotation „O“ auftritt, markiert dies das Ende einer Tokensequenz, die eine genetische Variation beschreibt, falls vorherige Token mit anderen Annotationen als „O“ vorhanden waren. In diesem Fall werden die Tokens im *currentEntity* Array zusammengeführt und dem *predictedArray* hinzugefügt, welches die vorhergesagten Entitäten speichert. Der Algorithmus gibt schließlich das *predictedEntity* Array zurück.
Es ist zu beachten, dass der in Pseudocode beschriebene Algorithmus 2 zur Zusammenführung von in Subklassen zerlegten Entitäten einen einfacheren Lösungsansatz verfolgt und bestimmte Einschränkungen aufweist. Der Algorithmus berücksichtigt insbesondere nicht den Fall, in dem zwei genetische Variationen unmittelbar aufeinander folgen. Stattdessen führt er alle aneinander angrenzenden annotierten Token zu einer einzigen Entität zusammen. In den verwendeten Korpora dieser Arbeit wurden jedoch keine Fälle von unmittelbar aufeinander folgenden genetischen Variationen beobachtet. Aus diesem Grund wurde entschieden, im Rahmen dieser Arbeit den einfacheren Algorithmus anzuwenden.

5.6 Technische Details

Die Experimente liefen auf dem *guppi1* Server der Humboldt-Universität, der mit vier NVIDIA GeForce GPUs ausgestattet ist: zwei GeForce GTX 1080 Ti (11178 MiB), eine GeForce GTX TITAN X (12212 MiB) und eine TITAN X (Pascal) (12196 MiB) und unter openSUSE Leap 15.3 läuft.
Es wurde Python in der Version 3.9.16 verwendet.

6 Ergebnisse

Im Folgenden werden in den Tabellen 4 bis 6 die Ergebnisse der NER-Tagger vorgestellt und mit den der Baseline Modelle verglichen.
Für das BioRed-Korpus wurde PubMedBERT-CRF [Luo+22] als Baseline-Modell verwendet. PubMedBERT-CRF basiert auf dem transformerbasierten BERT-Modell und nutzt ein Conditional Random Field (CRF), um die Abhängigkeiten zwischen aufeinanderfolgenden Tokens zu modellieren.
Für die tmVar3- und Osiris-Korpora wurde das im Abschnitt 2 vorgestellte tmVar3-Modell verwendet. Die Wahl mehrerer Baseline-Modelle ist darauf zurückzuführen, dass kein einzelnes veröffentlichtes Modell gefunden wurde, das jeweils auf den Korpora BioRed, tmVar3 und Osiris evaluiert. Die Ergebnisse der Baseline-Modelle für die verschiedenen Datensätze wurden direkt aus den entsprechenden Papern entnommen. Eine manuelle Installation und Konfiguration der Baseline-Modelle, um die Modellperformance zu bestimmen, wurde aufgrund des Umfangs dieser Arbeit nicht durchgeführt.
Die Werte für das BiLSTM-CRF Modell, das BioLinkBERT Modell und das fine-grained BioLinkBERT Modell sind Mittelwerte aus 5 Versuchsdurchführungen (wie im Abschnitt 5.3 beschrieben) und die Standardabweichung ist in Klammern angegeben. Die besten Ergebnisse sind jeweils fett markiert.

Tabelle 4: Performance der Modelle auf dem BioRed Korpus

Modell	Precision (± std)	Recall (± std)	F1-Score (± std)
PubMedBERT-CRF	0,847	**0,871**	0,859
BiLSTM-CRF	0,835 (± 0.05)	0,783 (± 0.04)	0,808 (± 0.04)
BioLinkBERT	**0,859** (± 0.02)	0,866 (± 0.03)	**0,863** (± 0.01)
Fine-grained BioLinkBERT	0,657 (± 0.06)	0,817 (± 0.03)	0,706 (± 0.04)

Tabelle 5: Performance der Modelle auf dem tmVar3 Korpus

Modell	Precision (± std)	Recall (± std)	F1-Score (± std)
tmVar 3.0	0,940	**0,889**	**0,914**
BiLSTM-CRF	0,856 (± 0.03)	0,659 (± 0.05)	0,744 (± 0.03)
BioLinkBERT	**0,954** (± 0.05)	0,682 (± 0.04)	0,795 (± 0.04)
Fine-grained BioLinkBERT	0,644 (± 0.05)	0,758 (± 0.06)	0,696 (± 0.06)

Tabelle 6: Performance der Modelle auf Osiris Korpus

Modell	Precision (± std)	Recall (± std)	F1-Score (± std)
tmVar 3.0	**0,986**	0,850	**0,913**
BiLSTM-CRF	0,859 (± 0.04)	0,785 (± 0.05)	0,820 (± 0.02)
BioLinkBERT	0,897 (± 0.02)	**0,865** (± 0.03)	0.899 (± 0.03)
Fine-grained BioLinkBERT	0,637 (± 0.05)	0,714 (± 0.04)	0,673 (± 0.04)

Die Trainingslaufzeit aller Modelle zusammen belief sich auf ungefähr 300 Stunden. Dabei variieren die Trainingszeiten der einzelnen Modelle. Es ist zu beobachten, dass mit der verwendeten GPU, das Training des BioLinkBERT-Modells im Vergleich zum BiLSTM-CRF-Modell um ca. 20% schneller war, während das Fine-grained BioLinkBERT Modell sogar um ca. 35% schneller war als das BiLSTM-CRF-Modell.[6]
Die Tabelle 4 zeigt, dass das Modell BioLinkBERT auf dem BioRed-Korpus den besten F1-Score erzielt, während die Tabellen 5 und 6 zeigen, dass das Baseline-Modell tmVar3.0 auf den beiden anderen Korpora den besten F1-Score erzielt. Auf dem Osiris Korpus liefert das BioLinkBERT-Modell ähnlich gute Werte wie das Baseline-Modell, während es auf dem tmVar3 Korpus einen deutlich niedrigeren Recall aufweist, nämlich 0,682 im Vergleich zu 0,889.
Außerdem lässt sich feststellen, dass das Transformator-basierten BioLinkBERT Modell auf den Korpora konstant besser abschneiden als das BiLSTM-CRF-Modell. Berechnet man den durchschnittlichen F1-Score über die drei Korpora für das BiLSTM-CRF Modell, dann erhalt man 0,791, während er für das BioLinkBERT bei 0,853 liegt.
Das fine-grained BioLinkBERT schneidet in den Ergebnissen im Vergleich zu allen anderen Modellen schlechter ab. In sämtlichen untersuchten Korpora erzielt es niedrigere Präzisions-, Recall- und F1-Score-Werte im Vergleich zu den Baseline-Modellen, dem BiLSTM-CRF Modell und dem BioLinkBERT Modell. Durchschnittlich erreicht das fine-grained BioLinkBERT Modell über die drei Korpora einen F1-Score von 0,692.
Die geringe Standardabweichung ($< 0{,}1$) bei den Precision-, Recall- und F1-Score-Werten deutet darauf hin, dass die Ergebnisse der Modelle auf den Korpora relativ stabil sind und nicht signifikant von größeren Variationen beeinflusst wird.

[6] Auf dem BioRed Korpus brauchte das Training des BiLSTM-CRF-Modells ca. 480 Minuten, für das BioLinkBERT-Modell ca. 380 Minuten und das Fine-grained BioLinkBERT-Modell ca. 315 Minuten.
Auf dem tmVar3 Korpus brauchte das BiLSTM-CRF-Modell ca. 450 Minuten, das BioLinkBERT-Modell ca. 350 Minuten und das Fine-grained BioLinkBERT-Modell ca. 280 Minuten.
Auf dem Osiris Korpus brauchte das BiLSTM-CRF-Modells ca. 115 Minuten, das BioLinkBERT-Modell ca. 65 Minuten und das Fine-grained BioLinkBERT-Modell etwa 50 Minuten.

7 Diskussion

Um die auftretenden Probleme zu verstehen, wurde eine Fehleranalyse durchgeführt, für die eigens Fehlerklassen konzipiert wurden. Durch die Kategorisierung der Fehler konnten systematische Fehlerquellen der Modelle identifiziert werden.
Anschließend wurde analysiert, wie sich die Fehlerquellen zwischen den Modellen unterscheiden und mögliche Ursachen, sowie Verbesserungsvorschläge für zukünftige Arbeiten diskutiert.

7.1 Fehleranalyse

Für die Fehleranalyse wurden die Annotationsfehler für jede Modellarchitektur in jedem Korpus manuell überprüft[7]. Die identifizierten Fehler können grob in drei Kategorien eingeteilt werden: Erstens genetische Variationen, die vom Modell nicht erkannt wurden. Zweitens Variationen, die zwar erkannt wurden, bei denen jedoch die Entitätsgrenzen nicht korrekt gesetzt wurden. Und drittens Zeichenfolgen, die irrtümlicherweise als genetische Variationen annotiert wurden, obwohl sie keine solchen darstellen.
Die nicht erkannten Variationen lassen sich weiter differenzieren: Zum einen handelt es sich um Variationen, die eine standardisierte Form haben, beispielsweise „*-79C>T*“ (Nicht erkannt, standardisierte Form), und zum anderen um komplexer beschriebene Variationen, die durch natürlichsprachliche Formulierungen wie „*G–>A substitution at codon 1763*“ erschwert werden (Nicht erkannt, natürlichsprachlich).
Die erkannten Variationen, bei denen die gesetzten Entitätsgrenzen abweichen, wurden in zwei Unterkategorien unterteilt: Zum einen handelt es sich um die Fälle, in denen die Anpassung der Start- und/oder Endposition um ±1 die korrekte Annotation ergibt (Gering abweichende Entitätsgrenzen). Zum Beispiel würde aus „(11381G/C“ durch eine solche Anpassung „11381G/C“ werden. Zum anderen die Fehler, in denen eine Verschiebung der Start- und/oder Endpositionen um mehr als ±1 erforderlich ist, um die korrekten Entitätsgrenzen zu erreichen (Erheblich abweichende Entitätsgrenzen).
Und drittens Zeichenfolgen, die irrtümlicherweise als genetische Variationen annotiert wurden, obwohl sie keine solchen darstellen. Auch hier wurden 2 Fälle unterschieden. Zum einen Fälle, in denen die erkannten Zeichenfolgen natürlichsprachlich sind, jedoch keine genetischen Variationen repräsentieren (Falsch erkannt, natürlichsprachlich), wie zum Beispiel „arginine deletion“. Zum anderen die Fälle, in denen die erkannten Zeichenfolgen nicht natürlichsprachlich sind und keine genetische Variationen sind (Falsch erkannt, sonstiges), wie im Fall von „573X“.
Im Osiris Korpus konnte außerdem eine Fehlerklasse beobachtet werden in denen das Modell eine genetische Variation korrekt erkennt, die im Osiris-Korpus jedoch inkonsistent oder unvollständig annotiert wurde (Inkonsistente Korporaannotation). Insbesondere tritt dies auf, wenn dieselbe genetische Variation mehrmals im Text vorkommt, aber nur einmal im Korpus annotiert wurde. Diese Fälle könnten bei der Bewertung der Modellleistung berücksichtigt werden, da sie keine Fehler an sich darstellen.

[7] Aus den 5 Modellen, die für jeden Korpus pro Modellarchitektur trainiert wurden, wurde jeweils das mit dem höchsten F1-Score für die Analyse ausgewählt.

Die Fehlerkategorien wurde dabei so konstruiert, dass jedem Annotationsfehler genau eine Fehlerkategorie zugeordnet werden konnte.
Die Tabelle 7 zeigt die Annotationsfehler des BiLSTM-CRF-Modells, die auf den Korpora identifiziert wurden, aufgeteilt in die verschiedenen Kategorien.

Fehlerkategorie	**BioRED**	**tmVar3**	**Osiris**
Nicht erkannt, natürlichsprachlich	7	45	3
Nicht erkannt, standardisierte Form	23	43	14
Gering abweichende Entitätsgrenzen	7	2	6
Erheblich abweichende Entitätsgrenzen	14	6	8
Falsch erkannt, natürlichsprachlich	2	0	3
Falsch erkannt, sonstiges	1	2	3
Inkonsistente Korporaannotation	0	0	26

Tabelle 7: Fehlerquellen des BiLSTM-CRF Modells

Bei der Analyse der Annotationenfehler lässt sich beobachten, dass das BiLSTM-CRF-Modell in den meisten Fällen, in denen es Vorhersagen trifft, korrekt liegt. Dies spiegelt sich auch in dem relativ hohen Precision Wert von über 85% auf allen Korpora wieder. Die Analyse zeigt, dass das Modell einfache und standardisierte Formen genetischer Variation gut erkennt. Auf dem BioRed Korpus wurden 23 (von 200), auf dem tmVar Korpus 43 (von 235) und auf dem Osiris Korpus wurden 14 (von 136) solcher Fehler identifiziert. In einigen Fällen zeigt das Modell jedoch Schwierigkeiten, dies mit hoher Zuverlässigkeit zu tun, und erkennt die gleiche genetische Variation in verschiedenen, aber ähnlichen Kontexten nicht konsistent. Zum Beispiel wird die Variation „G118“ im Satz „...more abundant than the G118 allele“ erkannt, aber nicht in einem der folgenden Sätze „...stability between A118 and G118 alleles“, obwohl die Satzstruktur sehr ähnlich ist. Darüber hinaus sind bei komplexeren standardisierten genetischen Variationen häufiger größere Abweichungen bei der Festsetzung der Entitätsgrenzen zu beobachten, z. B. die Annotation „+652-653delAG“ anstelle von „g.ORF15 + 652-653delAG“. Es fällt ebenfalls auf, dass genetische Variationen, die aus sehr vielen Zeichen bestehen und in natürlicher Sprache beschrieben werden, oftmals übersehen werden, wie z. B. „adenosine, guanosine, cytidine, and thymidine in position 118“.
Die folgende Tabelle 8 zeigt die Annotationsfehler des BioLinkBERT Modells auf den Korpora.

Fehlerkategorie	BioRED	tmVar3	Osiris
Nicht erkannt, natürlichsprachlich	5	38	2
Nicht erkannt, standardisierte Form	14	32	8
Gering abweichende Entitätsgrenzen	9	3	3
Erheblich abweichende Entitätsgrenzen	9	9	4
Falsch erkannt, natürlichsprachlich	2	1	1
Falsch erkannt, sonstiges	3	2	2
Inkonsistente Korporaannotation	0	0	8

Tabelle 8: Fehlerquellen des BioLinkBERT Modells

Dabei lässt sich beobachten, dass kaum Annotationenfehler gemacht werden, die nicht auch vom BiLSTM-CRF Modell gemacht werden. Dies zeigt, dass das BiLSTM-CRF Modell als Verbesserung hinsichtlich der Annotationsqualität angesehen werden kann, was sich in einem insgesamt besseren F1-Score auf den Korpora widerspiegelt. Die Verbesserung zeigt sich vor allem darin, dass genetische Variationen, die in standardisierter Form auftreten, zuverlässiger erkannt werden, insbesondere solche, die in der Struktur nicht häufig im Trainingskorpus vorkommen, wie z.B. „del-(439-443)“. Auf dem BioRED-Korpus konnten nur 14 statt 23, auf dem tmVar3-Korpus 32 statt 43 und auf dem Osiris-Korpus nur 8 statt 14 Fehler der Art gefunden werden.
Bei genetischen Variationen, die in natürlicher Sprache beschrieben werden, haben sowohl das BioLinkBERT-Modell wie auch das BiLSTM-CRF-Modell Schwierigkeiten, diese zuverlässig zu erkennen, insbesondere wenn sie sehr lang sind. Hierbei zeigt sich jedoch eine leichte Verbesserung beim BioLinkBERT-Modell. Die Anzahl der Fehler hat sich auf dem BioRed-Korpus um 2, auf dem tmVar3-Korpus um 7 und auf dem Osiris-Korpus um 1 verringert.
Eine interessante Beobachtung ist, dass sich das BioLinkBERT-Modell an die inkonsistenten Fälle der Korpusannotation angepasst hat. Im Vergleich zum BiLSTM-CRF-Modell annotiert das BioLinkBERT-Modell seltener genetische Variationen, wenn diese nicht auch im Osiris-Korpus annotiert wurden. Während das BiLSTM-CRF-Modell 24 Fehler aufgrund inkonsistenter Korpusbezeichnungen hatte, wies das BioLinkBERT-Modell nur 8 solcher Fehler auf dem Osiris Korpus auf.
Die Tabelle 9 zeigt die Annotationsfehler des fine-grained BioLinkBERT Modells, die auf den Korpora identifiziert wurden.

Fehlerkategorie	BioRED	tmVar3	Osiris
Nicht erkannt, natürlichsprachlich	7	14	4
Nicht erkannt, standardisierte Form	6	12	11
Gering abweichende Entitätsgrenzen	16	10	10
Erheblich abweichende Entitätsgrenzen	8	15	10
Falsch erkannt, natürlichsprachlich	33	42	3
Falsch erkannt, sonstiges	43	53	12
Inkonsistente Korporaannotation	0	0	15

Tabelle 9: Fehlerquellen des fine-grained BioLinkBERT Modells

Das fine-grained BioLinkBERT Modell zeigt einerseits eine Verbesserung, da es deutlich mehr genetische Variationen erkennt. Insbesondere gelingt es dem Modell, lange natürlichsprachige genetische Variationen zu identifizieren, die vom BiLSTM-CRF Modell und dem BioLinkBERT Modell nicht erkannt werden. Ein Beispiel dafür ist die genetischen Variation „single-nucleotide deletion at position 329“ aus dem tmVar3 Testdatensatz. Während diese Variation von den anderen beiden Modellen nicht erkannt wird, identifiziert das fine-grained BioLinkBERT Modell zumindest teilweise die Komponente „deletion at position 329“. Das fine-grained BioLinkBERT Modell hat jedoch Schwierigkeiten bei der korrekten Setzung der Entitätsgrenzen für lange genetische Variationen, die in natürlicher Sprache beschrieben werden. Kurze, standardisierte Formen genetischer Variationen erkennt das Modell in den meisten Fällen wie die beiden anderen Modelle. Jedoch annotiert es häufig Zeichenketten fälschlicherweise als genetische Variation. Das spiegelt sich im niedrigen Precision Wert des Modells von unter 70% auf allen Korpora wieder. Auffällig ist dabei, dass das fine-grained BioLinkBERT-Modell wiederholt den Annotationsfehler macht, alleinstehende Zahlen und Begriffe, die auf genetische Variationen hinweisen, wie „nucleotide“, sowie Proteinnamen zu annotieren. Dies macht den größten Teil der Fehler aus und erklärt den relativ niedrigen Precision Wert.
Ein weiter häufig vorkommender Fehler, ist dass die genetischen Variationen bevorzugt so annotiert werden, wie sie durch die Regex für das Trainingsdatensatz geparst wurden. Wenn Klammern oder andere Trennzeichen in und um den Variationen an Stellen vorkommen, die in der Regex-Konstruktion nicht vorgesehen waren, neigt das Modell dazu, diese nicht mit zu annotieren, was zu Fehlern in der Entitätsgrenzsetzung führt.
Zusätzlich wurde zum besseren Verständnis der Fehler des fine-grained BioLinkBERT-Modells eine Fehleranalyse auf Subklassenebene durchgeführt. Dabei wurden die Annotationsfehler untersucht, bevor sie mit der im Abschnitt 5.5 beschriebenen Methode zusammengeführt wurden[8].
Die Tabelle 10 zeigt die Häufigkeit mit der jede Subklasse vom fine-grained BioLinkBERT Modell auf den jeweiligen Korpora falsch annotiert wurde. Dabei wurden die Fehlannotationen auf dem Osiris-Korpus, die durch inkonsistente Korpusannotation zustande kamen, nicht mit einberechnet.

[8]Dabei wurde die Fehleranalyse für das jeweils beste fine-grained BioLinkBERT Modell für jeden Korpus für die Analyse ausgewählt.

Die Werte in Klammern geben an, wie oft die Subklasse insgesamt vom Modell vorhergesagt wurde.

Subklasse	BioRED	tmVar3	Osiris
Reference sequence type	0 (28)	2 (43)	0 (0)
Exon/intron	0 (1)	1 (19)	0 (0)
Mutation type	3 (65)	5 (85)	4 (97)
Frame shift mutation	0 (0)	0 (0)	0 (0)
DNA/RNA nucleotide	1 (79)	10 (141)	10 (187)
Protein amino acid	13 (83)	4 (78)	1 (52)
Mutation position	0 (51)	2 (79)	12 (134)
Snp id	4 (52)	1 (28)	0 (16)
Mutation component	99 (160)	113 (233)	16 (37)

Tabelle 10: Fehlannotationen der Subklassen des fine-grained BioLinkBERT Modells

Die Tabelle zeigt, dass die Mehrheit der Fehlklassifikationen (228) auf die Subklasse „mutation component“ zurückzuführen ist, während die Subklassen „Reference sequence type“, „Mutation type“, „Mutation position“ und „Snp id“ vergleichsweise geringe Fehlklassifikationen aufweisen (alle <15). Auffällig ist außerdem, dass das fine-grained BioLinkBERT-Modell kein Token als „frame shift mutation“ annotiert hat.

7.2 Erkenntnisse und Vergleich der Modelle

Die Ergebnisse der Fehleranalyse deuten darauf hin, dass genetische Variationen sowohl in standardisierter Form als auch in einfacheren natürlichsprachlichen Formulierungen mit einer hoher Wahrscheinlichkeit von den Modellen erkannt werden können. Schwierigkeiten ergeben sich bei der zuverlässigen Erkennung komplex beschriebener genetischer Variationen in standardisierten Formen und langer genetischer Variationen, die in natürlicher Sprache beschrieben werden. Der Vergleich der Modelle zeigt, dass das BioLinkBERT-Modell eine Verbesserung in der Annotationsqualität gegenüber dem BiLSTM-CRF Modell darstellt.
Das fine-grained BioLinkBERT-Modell zeigt einen vielversprechenden Ansatz zur Erkennung langer genetischer Variationen in natürlicher Sprache, ein Bereich, in dem sowohl das BiLSTM-CRF-Modell als auch das BioLinkBERT-Modell noch Schwierigkeiten haben. Es gibt jedoch auch hierbei noch Schwierigkeiten, da die festgelegten Entitätsgrenzen oft noch von den tatsächlichen Entitätsgrenzen abweichen. Diese Schwächen könnten auf verschiedene Faktoren zurückzuführen sein.
Eine mögliche Fehlerquelle könnte die derzeitige Form der Regex sein, die nur etwa 65-75% der genetischen Variationen in den Korpora abdeckt. Dies hat zur Folge, dass ein größerer Teil der genetischen Variationen der fine-grained Subklasse „mutation component“ zugeordnet wird, wodurch diese Subklasse viele sehr unterschiedliche Bestandteile enthält. Aus der Tabelle 10 lässt sich entnehmen, dass etwa 75% der Fehlklassifizierungen auf die fehlerhafte Zuordnung zur Subklasse „mutation component“ zurückzuführen sind.

Eine weitere mögliche Ursache könnte in der Zusammensetzung der fine-grained zu Coarse-grained Entitäten liegen, die wahrscheinlich eine höhere Komplexität und zusätzliche Logik erfordern. Beispielsweise könnte eine Fehlklassifizierung in Situationen auftreten, in denen eine Entität identifiziert wird, die keine Informationen über den Mutationstyp, die Position, Nukleotide oder Proteine enthält. Daher kann es sinnvoll sein, bei der Zusammensetzung der Entität die Informationen zu berücksichtigen, die aus den fine-grained annotierten Komponenten gewonnen werden können.
Zusätzlich könnte ein Post Processing zur Verbesserung der Performance beitragen, wie es auch im tmVar3-Modell durchgeführt wird. Im tmVar3-Modell werden die Entitätsgrenzen im Nachhinein angepasst, unter anderem, um fehlende schließende Klammern hinzuzufügen. Ähnliche Abweichungen in den Entitätsgrenzen bei Klammern und anderen Trennzeichen konnten auch beim fine-grained BioLinkBERT-Modell beobachtet werden.
Eine weitere Überlegung wäre, ein anderes Matching-Kriterium als das im Abschnitt 5.4 beschriebene „exact match“ zu verwenden, da dieses Matching-Kriterium möglicherweise nicht die tatsächliche Leistung der Modelle abbildet. Zum Beispiel können Inkonsistenzen in den Korporaannotationen zu einer Verschlechterung der Modellergebnisse führen, wie sie beispielsweise im Osiris-Korpus häufig beobachtet wurden. In Tsai et al. [Tsa+06] wird angegeben, dass die Übereinstimmung zwischen den Annotatoren für biomedizinische NER bei der Annotation von Texten zwischen 87% und 89% liegt. Ein Beispiel ist die annotierte genetische Variation im Osiris-Korpus „(C2029T nucleotide substitution)“. Wenn das Modell stattdessen „C2029T nucleotide substitution“ erkennt, unterscheiden sich die Annotationen durch das Weglassen der Klammern. In solchen Situationen könnte ein alternatives Matching-Kriterium, dass weniger strikt ist in Betracht gezogen werden, um eine realistischere Bewertung der Modellleistung zu erhalten.
Um sicherzustellen, dass die Evaluationsergebnisse der Modelle zuverlässig sind, wurden für jede Modellarchitektur fünf verschiedene Modelle auf den Korpora trainiert und evaluiert. Anschließend wurden der Durchschnitt und die Standardabweichung der Ergebnisse berechnet.

8 Zusammenfassung

In dieser Arbeit wurden drei verschiedene Modelle, nämlich das BiLSTM-CRF, das BioLinkBERT und das fine-grained BioLinkBERT, zur Entwicklung eines NER-Taggers für genetische Variationen vorgestellt, implementiert und auf den drei Korpora BioRed, tmVar3 und Osiris trainiert und getestet. Die Leistung dieser Modelle wurde dabei mit den Baseline-Modellen tmVar3 und PubMedBERT verglichen. Es hat sich gezeigt, dass das Modell BioLinkBERT auf dem BioRed-Korpus den besten F1-Score erzielt, während auf dem Osiris- und tmVar3-Korpus das tmVar3.0-Modell den besten F1-Score erzielt.
Außerdem wurden die in der Einleitung aufgeworfenen Fragestellungen untersucht: Erstens, wie präzise können die entwickelten Modelle genetische Variationen erkennen und zweitens, welchen Einfluss hat ein fine-grained NER-Ansatz auf die Erkennung genetischer Variationen?
Dabei konnte festgestellt werden, dass das BiLSTM-CRF Modell einen durchschnittlichen F1-Score über die drei Korpora von 0,791 erzielt, während das BioLinkBERT Modell einen durchschnittlichen F1-Score von 0,853 erreicht. Das fine-grained BioLinkBERT Modell zeigt zwar einen niedrigeren durchschnittlichen F1-Score von 0,692, jedoch zeigt der fein granulierte Ansatz positive Auswirkungen auf die Erkennung langer genetischer Variationen in natürlicher Sprache. Dies ist ein Bereich, in dem sowohl das BiLSTM-CRF Modell als auch das BioLinkBERT-Modell noch Schwierigkeiten haben.

Literaturverzeichnis

[Akb+19] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter u. a. „FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP“. In: (Juni 2019), S. 54–59. DOI: 10.18653/v1/N19-4010.

[ABV18] Alan Akbik, Duncan Blythe und Roland Vollgraf. „Contextual String Embeddings for Sequence Labeling“. In: *Proceedings of the 27th International Conference on Computational Linguistics*. International Committee on Computational Linguistics. Zalando Research, Muhlenstraße 25, 10243 Berlin: International Committee on Computational Linguistics, Aug. 2018, S. 1638–1649.

[AX19] Felipe Almeida und Geraldo Xexéo. „Word Embeddings: A Survey“. In: *CoRR* abs/1901.09069 (2019). arXiv: 1901.09069.

[Ber+10] B. E. Bernstein, J. A. Stamatoyannopoulos, J. F. Costello, B. Ren, A. Milosavljevic u. a. „The NIH Roadmap Epigenomics Mapping Consortium“. In: *Nature Biotechnology* 28.10 (2010), S. 1045–1048. DOI: 10.1038/nbt1010-1045.

[Bre+19] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang und Martin Zinkevich. „Data Validation for Machine Learning“. In: *Proceedings of Machine Learning and Systems 2019, MLSys 2019*. Stanford, CA, USA: mlsys.org, 2019.

[BST17] H. K. Brittain, R. Scott und E. Thomas. „The rise of the genome and personalised medicine“. In: *Clinical medicine (London, England)* 17.6 (2017), S. 545–551. DOI: 10.7861/clinmedicine.17-6-545.

[Cap+07] J. Gregory Caporaso u. a. „MutationFinder: a high-performance system for extracting point mutation mentions from text“. In: *Bioinformatics* 23.14 (2007), S. 1862–1865. DOI: 10.1093/bioinformatics/btm235.

[CA21] [Vorname] Carielli und [weitere Autoren]. „Titel des Papers“. In: *[Name des Journals]* [Bandnummer].[Ausgabennummer] (2021), [Seitenzahlen]. DOI: [DOIdesPapers].

[Chi98] Nancy A. Chinchor. „Overview of MUC-7“. In: *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*. 1998.

[CL19] Hyun Cho und Hyunju Lee. „Biomedical named entity recognition using deep neural networks with contextual information“. In: *BMC Bioinformatics* 20.1 (2019), S. 735. DOI: 10.1186/s12859-019-3321-4.

[Col+11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu u. a. „Natural Language Processing (almost) from Scratch“. In: (2011). arXiv: 1103.0398 [cs.LG].

[Con15] The 1000 Genomes Project Consortium. „A global reference for human genetic variation“. In: *Nature* 526 (1. Okt. 2015), S. 68–74. DOI: 10.1038/nature15393.

[Cyb89] G. Cybenko. „Approximation by superpositions of a sigmoidal function“. In: *Mathematics of Control, Signals, and Systems* 2 (1989), S. 303–314. DOI: 10.1007/BF02551274.

[Dev+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[Du+16] X Du, Y Cai, S Wang und L Zhang. „Overview of deep learning“. In: *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE. 2016, S. 159–164.

[Dun+16] Johan T. den Dunnen, Raymond Dalgleish, Donna R. Maglott, Reece K. Hart, Marc S. Greenblatt u. a. „HGVS Recommendations for the Description of Sequence Variants: 2016 Update“. In: *Human Mutation* 37.6 (2016). For the 25th Anniversary Commemorative Issue, [Seitenzahlen]. DOI: 10.1002/humu.22981.

[DB14] Kyunghyun Cho Dzmitry Bahdanau und Yoshua Bengio. „Neural Machine Translation by Jointly Learning to Align and Translate“. In: *arXiv preprint arXiv:1409.0473* (2014).

[Fur+08] Laura I Furlong, Holger Dach, Martin Hofmann-Apitius und Ferran Sanz. „OSIRISv1.2: a named entity recognition system for sequence variants of genes in biomedical literature.“ In: *BMC Bioinformatics* 9 (2008), S. 84. DOI: 10.1186/1471-2105-9-84.

[GB18] John M Giorgi und Gary D Bader. „Transfer learning for biomedical named entity recognition with neural networks“. In: *Name der Zeitschrift/Konferenz* Bandnummer.Ausgabenummer (2018). Received on February 12, 2018; revised on April 25, 2018; editorial decision on May 25, 2018; accepted on May 29, 2018, Seitenzahlen. DOI: DOI-Nummer.

[GS05] Alex Graves und Jürgen Schmidhuber. „Framewise phoneme classification with bidirectional LSTM and other neural network architectures“. In: *Neural Networks* 18.5-6 (2005), S. 602–610. DOI: 10.1016/j.neunet.2005.06.042.

[Hab+17] Maryam Habibi, Leon Weber, Mariana Neves, David Luis Wiegandt und Ulf Leser. „Deep learning with word embeddings improves biomedical named entity recognition“. In: *Bioinformatics* 33.14 (Juli 2017), S. i37–i48.

[Har54] Zellig S. Harris. „Distributional Structure“. In: *WORD* 10.2-3 (1954), S. 146–162. DOI: 10.1080/00437956.1954.11659520.

[HS97] Sepp Hochreiter und Jürgen Schmidhuber. „Long short-term memory“. In: *Neural computation* 9.8 (1997), S. 1735–1780.

[Hor+04] Florence Horn u. a. „Automated extraction of mutation data from the literature: application of MuteXt to G protein-coupled receptors and nuclear hormone receptors“. In: *Bioinformatics (Oxford, England)* 20.4 (2004), S. 557–568. DOI: 10.1093/bioinformatics/btg449.

[HXY15] Zhiheng Huang, Wei Xu und Kai Yu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. 2015. arXiv: 1508.01991 [cs.CL].

[Kan03] Laveen Kanal. „Perceptron“. In: (Jan. 2003), S. 1383–1385.

[KMN24] Imed Keraghel, Stanislas Morbieu und Mohamed Nadif. „A survey on recent advances in Named Entity Recognition“. In: *arXiv preprint arXiv:2401.10825* 1 (2024). Version 1, S. 1. arXiv: 2401.10825 [cs.CL].

[KT21] Veysel Kocaman und David Talby. „Biomedical Named Entity Recognition at Scale“. In: *Lecture Notes in Computer Science*. Bd. 12661. 2021.

[Kor21] M. V. Koroteev. *BERT: A Review of Applications in Natural Language Processing and Understanding*. 2021. arXiv: 2103.11943 [cs.CL].

[LMP01] John Lafferty, Andrew McCallum und Fernando Pereira. „Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data“. In: *WhizBang! Labs–Research* (2001).

[Lam+16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami und Chris Dyer. „Neural Architectures for Named Entity Recognition“. In: *CoRR* abs/1603.01360 (2016). arXiv: 1603.01360.

[Li+22] Jing Li, Aixin Sun, Jianglei Han und Chenliang Li. „A Survey on Deep Learning for Named Entity Recognition“. In: *IEEE Transactions on Knowledge and Data Engineering* 34.1 (Jan. 2022), S. 50–70. ISSN: 2326-3865. DOI: 10.1109/tkde.2020.2981314.

[Luo+22] Ling Luo u. a. „BioRED: a rich biomedical relation extraction dataset“. In: *Briefings in Bioinformatics* 23.5 (2022), bbac282. DOI: https://doi.org/10.1093/bib/bbac282.

[Mal+20] E. R. Malone, M. Oliva, P. J. B. Sabatini und et al. „Molecular profiling for precision cancer therapies“. In: *Genome Medicine* 12.1 (2020), S. 8. DOI: 10.1186/s13073-019-0703-1.

[MAM08] Alireza Mansouri, Lilly Suriani Affendey und Ali Mamat. „Named Entity Recognition Approaches“. In: *IJCSNS International Journal of Computer Science and Network Security* 8.2 (Feb. 2008), S. 6.

[McD+04] Ryan T. McDonald u. a. „An entity tagger for recognizing acquired genomic variations in cancer literature“. In: *Bioinformatics* 20.17 (2004), S. 3249–3251. DOI: https://doi.org/10.1093/bioinformatics/bth350.

[Mik+13] Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[MP69] Marvin Minsky und Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[OTK02] Tomoko Ohta, Yuka Tateisi und Jin-Dong Kim. „The GENIA Corpus: an Annotated Research Abstract Corpus in Molecular Biology Domain“. In: *Proceedings of the Human Language Technology Conference (HLT 2002)* (Jan. 2002), S. 82–86.

[RM95] Lance Ramshaw und Mitch Marcus. „Text Chunking using Transformation-Based Learning“. In: (1995).

[Raz+22] Shaina Raza, Deepak John Reji, Femi Shajan und Syed Raza Bashir. „Large-scale application of named entity recognition to biomedicine and epidemiology“. In: *PLOS Digital Health* (Dez. 2022). DOI: 10.1371/journal.pdig.0000152.

[Son+21] Bosheng Song, Fen Li, Yuansheng Liu und Xiangxiang Zeng. „Deep learning methods for biomedical named entity recognition: a survey and qualitative comparison“. In: *Briefings in Bioinformatics* 22.6 (Juli 2021), bbab282. ISSN: 1477-4054. DOI: 10.1093/bib/bbab282. eprint: https://academic.oup.com/bib/article-pdf/22/6/bbab282/41089553/bbab282.pdf.

[Sun+22] Mujeen Sung, Minbyul Jeong, Yonghwa Choi, Donghyeon Kim, Jinhyuk Lee u. a. „BERN2: an advanced neural biomedical named entity recognition and normalization tool“. In: *CoRR* abs/2201.02080 (2022). arXiv: 2201.02080.

[TW02] Lorraine Tanabe und W John Wilbur. „Tagging gene and protein names in biomedical text“. In: *Bioinformatics* 18.8 (2002), S. 1124–1132. DOI: 10.1093/bioinformatics/18.8.1124.

[Tsa+06] Richard Tzong-Han Tsai, Shih-Hung Wu, Wen-Chi Chou, Yu-Chun Lin, Ding He u. a. „Various criteria in the evaluation of biomedical named entity recognition“. In: *BMC Bioinformatics* 7.1 (Feb. 2006), S. 92. DOI: 10.1186/1471-2105-7-92.

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones u. a. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[Web+21] Leon Weber, Mario Sänger, Jannes Münchmeyer, Maryam Habibi, Ulf Leser u. a. „HunFlair: an easy-to-use tool for state-of-the-art biomedical named entity recognition“. In: *Bioinformatics* 37.17 (Jan. 2021), S. 2792–2794. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab042. eprint: https://academic.oup.com/bioinformatics/article-pdf/37/17/2792/50339120/btab042_supplementary_data.pdf.

[Wei+13] Chih-Hsuan Wei u. a. „tmVar: a text mining approach for extracting sequence variants in biomedical literature“. In: *Bioinformatics* 29.11 (2013), S. 1433–1439. DOI: 10.1093/bioinformatics/btt156.

[Wei+22] Chih-Hsuan Wei u. a. „tmVar 3.0: an improved variant concept recognition and normalization tool“. In: *Bioinformatics (Oxford, England)* 38.18 (2022), S. 4449–4451. DOI: 10.1093/bioinformatics/btac537.

[YLL22] Michihiro Yasunaga, Jure Leskovec und Percy Liang. *LinkBERT: Pretraining Language Models with Document Links*. 2022. arXiv: 2203.15827 [cs.CL].

[Zha+20] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin u. a. „Do RNN and LSTM have Long Memory?“ In: *Proceedings of the 37th International Conference on Machine Learning*. Hrsg. von Hal Daumé III und Aarti Singh. Bd. 119. Proceedings of Machine Learning Research. PMLR, Juli 2020, S. 11365–11375.

[Zho+04] GuoDong Zhou, Jie Zhang, Jian Su, Dan Shen und ChewLim Tan. „Recognizing names in biomedical texts: a machine learning approach“. In: *Bioinformatics* 20.7 (Feb. 2004), S. 1178–1190. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bth060. eprint: https://academic.oup.com/bioinformatics/article-pdf/20/7/1178/48905293/bioinformatics_20_7_1178.pdf.

Appendix

Anhang A

Reguläre Ausdrücke zur Erstellung des fine-grained BioLinkBERT Korpus

Hier sind die regulären Ausdrücke, die die zur Erstellung des Korpus für den fine-grained BioLinkBERT NER-Tagger verwendet wurden. Die dargestellten Ausdrücke sind leicht vereinfacht, um das Verständnis zu erleichtern und weichen in der tatsächlichen Implementierung in einigen Details leicht ab.
Die regulären Ausdrücke setzen sich aus mehreren Teilausdrücken zusammen.
Die Teilausdrücke sind wie folgt definiert:

```
REF          := [cgrm]\.
P_REF        := [p]\.
NUC          := [ATGCatgcu]
OP           := [>|indel|delins|del|ins|inv|at|...]
FRAME_SHIFT  := [fs|fsX|X]
POS          := ([\+]\-]? \d+|?)
NUC_POS      := ((POS)? [\+|\-]? POS | \* POS | POS \*)
NUC_RANGE    := NUC_POS (\_ NUC_POS)?
PROTEIN      := [Arg|Ser|Val|...]
```

Die verwendeten regulären lassen sich dann aus den Teilausdrücken wie folgt definieren:

1. **Regex 1:** `(REF)? NUC_RANGE NUC? OP NUC`
   Dieser reguläre Ausdruck wird zur Erkennung von Substitutionen, Deletionen, Duplikationen, Insertionen und Inversionen auf der DNA-/RNA-Ebene verwendet. Er ermöglicht die optionale Angabe des Referenzsequenztyps (REF), gefolgt von der Positionsangabe oder einem Positionsbereich (NUC_RANGE). Optional kann vor der Operation (OP) die vorherige Nukleotidsequenz (NUC) spezifiziert werden, die dann auf den angegebenen Bereich angewendet wird.
   Beispiel: „`c.263C>T`“ oder „`c.715_717delGTC`“

2. **Regex 2:** `(REF)? NUC NUC_POS? OP NUC (OP REF? NUC_POS)?`
   Dieser reguläre Ausdruck wird ebenfalls zur Erkennung von Substitutionen, Deletionen, Duplikationen, Insertionen und Inversionen auf DNA/RNA-Ebene verwendet. Hier wird jedoch eine andere Struktur erkannt, bei der zuerst das/die zu ersetzende(n) Nukleotid(e) angegeben wird, gefolgt von einer optionalen Positionsangabe. Es können auch komplexere Substitutionen erkannt werden, bei denen zuerst das/die zu ersetzende(n) Nukleotid(e), dann die Position und schließlich ein Referenznukleotid und eine optionale Positionsangabe angegeben werden.
   Beispiel: „`c.A436C`“ oder „`G>T at c.898`“

3. **Regex 3:** `(P_REF)? PROTEIN NUC_POS? OP? PROTEIN?`
   Dieser reguläre Ausdruck wird zur Erkennung von Substitutionen, Deletionen, Duplikationen, Insertionen und Inversionen auf der Protein-Ebene verwendet. Dabei ist für

die Proteinbezeichnung sowohl der dreibuchstabige, einbuchstabige Aminosäurecode und die Verwendung des vollen Namens als Proteinbezeichnung zulässig.
Beispiel: „`p.Ser326Cys`“ oder „`p.T286M`“

4. **Regex 4:** `rs[0-9]+`
   Dieser reguläre Ausdruck wird zur Erkennung der SNP IDs verwendet.
   Beispiel: „`rs25531`“

Algorithmus zur Konstruktion der 'predictedValues' und 'goldValues' Arrays
Im Folgenden wird der Algorithmus beschrieben, mit dessen Hilfe die Arrays 'predictedValues' und 'goldValues' aus einer Liste von Modellannotationen bzw. den tatsächlichen Annotationen konstruiert werden. Dies ermöglicht einen elementweisen Vergleich der beiden Arrays.

Algorithm 3 Erhalte eine Liste der Entitäten aus den Token Annotationen

```
procedure GetValueArrays(predictedEntities, goldEntities, type)
    if type == „fine_grained“ then
        predictedEntities ← MergeFineGrainedEntities(predictedEntities)
    end if
    allEntities ← MergeArrays(predictedEntities, goldEntities)
    predictedValues ← []
    goldValues ← []
    for entity in allEntities do
        if entity in predictedEntity then
            predictedValues.append(entity)
        else
            predictedValues.append(O)
        end if
        if entity in goldEntities then
            goldValues.append(entity)
        else
            goldValues.append(O)
        end if
    end for
    return predictedValues, goldValues
end procedure
```

Akronyme

Bi-LSTM-CRF Bidirektionale LSTM-CRF. 4, 8

CRF Conditional Random Field. 5

NER Named Entity Recognition. 3, 4

NLP Natural Language Processing. 6

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird. Berlin, den 26. Februar 2024

.....