

# KI basierte Vorhersage von komplexen Produkt-System-Topologien

*Eine Abschlussarbeit zum Master of Science.*

**Kaan Timucin Dönmez**, Freie Universität Berlin, Deutschland

Matrikelnummer: 5335113

kaad01@zedat.fu-berlin.de

14.06. 2023

**Betreuer:**

**Mathias Uta**<sup>1</sup>, Siemens Energy AG Erlangen, Deutschland

**Gutachter:**

**Prof. Dr. Daniel Goehring**<sup>2</sup>, Freie Universität Berlin, Deutschland

**Prof. Dr. Tim Landgraf**<sup>3</sup>, Freie Universität Berlin, Deutschland

**Zitation:**

**Kaan Timucin Dönmez**, *KI basierte Vorhersage von komplexen Produkt-System-Topologien*, Freie Universität Berlin, Master Thesis, 2023,

**Version:** 1

---

<sup>1</sup>SE GT PRM EL DGO, Siemens Energy

<sup>2</sup>Dahlem Center for Machine Learning and Robotic, Autonomous Cars

<sup>3</sup>Dahlem Center for Machine Learning and Robotic, Autonomous Cars

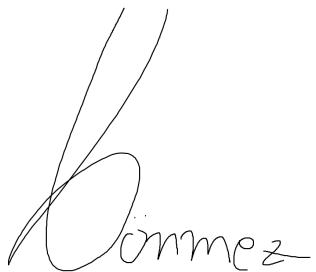
# Eigenständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

Berlin (Deutschland), 14.06.2023

A handwritten signature in black ink, appearing to read "Kaan Timucin Dönmez".

---

Kaan Timucin Dönmez

## **Zusammenfassung**

Diese Masterarbeit konzentriert sich auf die Entwicklung und Anwendung von Graph Neural Networks (GNNs) und Knowledge-Graph-Embedding-Methoden (KGEs) zur Vorhersage von Verbindungen in komplexen Produkt-System-Topologien. Das Hauptziel besteht darin, den Konfigurationsprozess von Hochspannungsanlagen zu optimieren. Trotz Herausforderungen wie dem Clever-Hans-Phänomen und der Notwendigkeit einer anwendungsspezifischen Modellanpassung, zeigte das Modell in den durchgeföhrten Experimenten vielversprechende Ergebnisse. Techniken wie gewichteter Loss und alternatives Negative Sampling trugen zur Verbesserung der Modellleistung bei. Das finale Modell kann durchschnittlich 91% der Verbindungen korrekt vorhersagen. Obwohl 56% der Vorhersagen fälschlicherweise Verbindungen annehmen, ist eine Präzision von 44% ausreichend, um den Konfigurationsprozess für Ingenieure zu beschleunigen. Diese Arbeit bietet somit eine solide Grundlage, die in zukünftigen Forschungsarbeiten weiter verbessert werden kann.

## **Abstract**

This Master's thesis is centered around the development and application of Graph Neural Networks (GNNs) and Knowledge-Graph-Embedding methods (KGEs) for predicting connections in complex product-system topologies. The primary objective is to optimize the configuration process of high-voltage installations. Despite challenges such as the Clever Hans phenomenon and the need for application-specific model adaptation, the model demonstrated promising results in the conducted experiments. Techniques such as weighted loss and alternative negative sampling contributed to the enhancement of the model's performance. The final model is capable of correctly predicting an average of 91% of the connections. Although 56% of the predictions falsely assume connections, a precision of 44% is sufficient to accelerate the configuration process for engineers. This work thus provides a solid foundation that can be further improved in future research.

# Inhaltsangabe

<b>1 Einführung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Stand der Technik . . . . .	2
1.3 Ziel . . . . .	3
<b>2 Grundlagen</b>	<b>4</b>
2.1 Knowledge Graph . . . . .	4
2.2 Domäne . . . . .	6
2.3 Link Prediction . . . . .	9
2.4 Machine Learning . . . . .	10
2.4.1 Induktives und transduktives Lernen . . . . .	11
2.4.2 Training von Machine Learning-Modellen . . . . .	11
2.4.3 Neuronale Netzwerke . . . . .	12
2.5 Embedding . . . . .	14
2.6 GraphSAGE . . . . .	16
2.6.1 GraphSAGE: Kontext und Motivation . . . . .	16
2.6.2 Funktionsprinzip von GraphSAGE . . . . .	16
<b>3 Implementation</b>	<b>18</b>
3.1 Datensatz . . . . .	18
3.2 Preprocessing . . . . .	20
3.3 Das Modell . . . . .	21
3.4 Training des neuronalen Netzwerks . . . . .	26
3.4.1 Die Klassifikationsfunktion . . . . .	26
3.4.2 Erstellung von positiven und negativen Beispielen . . . . .	28
3.4.3 Lossfunktion und Optimierung des Modells . . . . .	28
3.4.4 Iterationen zur Modellverbesserung und dessen Validierung . . . . .	29
3.5 Einfügen in den Graphen . . . . .	31
<b>4 Experimente und Auswertung</b>	<b>33</b>
4.1 Negative Sampling . . . . .	34
4.1.1 Experiment . . . . .	34
4.1.2 Auswertung . . . . .	35
4.2 Anpassung der Lossfunktion . . . . .	37
4.2.1 Experiment . . . . .	37
4.2.2 Auswertung . . . . .	37
4.3 Anpassung der Modellarchitektur . . . . .	38
4.3.1 Experiment . . . . .	38
4.3.2 Auswertung . . . . .	39
<b>5 Fazit</b>	<b>41</b>
5.1 Diskussion der Ergebnisse . . . . .	41
5.2 Ausblick . . . . .	42

# Abbildungen

1.1 Aktivitätsdiagramm der Schaltanlagenkonfiguration . . . . .	2
2.1 Ausschnitt aus einem KG . . . . .	5
2.2 Schaltanlage und Feld . . . . .	6
2.3 Feld mit Gas-isolierte Schaltanlage (GIS)-Modulen . . . . .	7
2.4 Ontologie . . . . .	8
2.5 Schaltanlage als Knowledge Graph . . . . .	9
2.6 Einfaches Neuronales Netzwerk . . . . .	12
2.7 Schematische Darstellung eines Perceptrons . . . . .	13
2.8 One-Hot-Vektor . . . . .	14
2.9 Embedding . . . . .	15
2.10 GraphSAGE-Algorithmus . . . . .	16
2.11 Illustration von GraphSAGE . . . . .	17
3.1 Darstellung eines Input-Baums . . . . .	20
3.2 Dropout in NN . . . . .	24
3.3 Classifier . . . . .	27
3.4 Precision und Recall . . . . .	31
4.1 Trainingsverlauf . . . . .	33
4.2 ROC-AUC-Score . . . . .	34
4.3 Experiment: Negative Sampling Training . . . . .	35
4.4 Experiment: Zusammengelegte LossFunktion . . . . .	36
4.5 Experiment: Gewichtete Lossfunktion . . . . .	38
4.6 Experiment: Modellarchitektur . . . . .	39
4.7 Experiment: Anpassung der Modellarchitektur . . . . .	40
5.1 Erweiterte Ontologie des Use Case . . . . .	43
5.2 Ausschnitt aus dem Knowledge Graph (KG) des erweiterten Use Case	44

# List of Source Codes

3.1 Code der GNN Architektur . . . . .	23
3.2 Vektorkonstruktion für „substation“-Knoten . . . . .	24
3.3 Die forward-Funktion des Modells . . . . .	26
3.4 Erstellung aller möglichen Kanten . . . . .	32

## Abkürzungsverzeichnis

- AUC** Area Under the Curve  
**CNN** Convolutional Neural Network  
**ELU** Exponential Linear Unit  
**GCN** Graph Convolutional Network  
**GIS** Gas-isolierte Schaltanlage  
**GNN** Graph Neural Network  
**KG** Knowledge Graph  
**KGE** Knowledge-Graph-Embedding  
**ML** Machine Learning  
**PST** Produkt-System-Topologie  
**PyG** PyTorch Geometric  
**ReLU** Rectified Linear Unit  
**ROC** Receiver Operating Characteristic

# Gliederung

**Einführung** ... Kapitel 1 dient als Einführung und gibt einen kurzen Überblick über das Arbeitsfeld und die Zielsetzung dieser Masterarbeit.

**Hintergrund** ... Kapitel 2 beleuchtet die relevanten Konzepte und Begriffe. Darüber hinaus wird diese Arbeit in den Kontext von verwandten Arbeiten gestellt, um einen umfassenden Überblick über das Forschungsfeld zu bieten.

**Implementierung** ... Kapitel 3 widmet sich einer detaillierten Beschreibung der verwendeten Daten und des entwickelten Modells. Es wird auch auf die Implementierung des spezifischen Anwendungsfalls eingegangen.

**Experimente und Auswertung** ... In Kapitel 4 wird das erstellte Modell evaluiert. Es werden die Definitionen der durchzuführenden Experimente beschrieben und die Ergebnisse dieser Experimente ausgewertet.

**Fazit und Ausblick** ... Kapitel 5 zieht ein abschließendes Fazit aus der Arbeit, diskutiert die Ergebnisse und skizziert mögliche zukünftige Arbeiten. Dieses Kapitel dient als Abschluss der Arbeit und gibt einen Ausblick auf mögliche zukünftige Forschungsrichtungen.

# 1 Einführung

## 1.1 Motivation

Komplexe Produkte, die aus zahlreichen Komponenten bestehen, bilden ein Produkt-System. Wenn man diesem System eine Ordnung zuweist, um die Verbindungen zwischen den Komponenten zu verdeutlichen, spricht man von einer Produkt-System-Topologie (PST), die im Grunde genommen ein KG<sup>1</sup> ist. Solche PSTs werden von vielen Firmen für verschiedene Produkte eingesetzt (siehe Abbildung 2.4), da sie eine geordnete und übersichtliche Darstellung von Daten bieten. Aus diesen PSTs können Unternehmen ihre Produkte konfigurieren. Bei einer solchen Konfiguration müssen zahlreiche Regeln und komplexe Strukturen berücksichtigt werden, weshalb diese oft manuell durchgeführt werden. Dieser Prozess ist zeitaufwendig und ressourcenintensiv. Es besteht daher ein großes Interesse daran, diese Konfiguration zu erleichtern, zum Beispiel durch Vorhersagen auf Basis von Kontextinformationen über die gewünschte Konfiguration.

Vorhersagen in KGs sind essentiell und haben in den letzten Jahren zunehmend an Interesse gewonnen, sei es für Freundschaftsvorschläge in sozialen Medien [32] oder Filmempfehlungen von Streamingdiensten [16]. Die am häufigsten verwendeten Vorhersagen sind Linkpredictions<sup>2</sup>. Eine gute Linkprediction sollte auf Basis von wenig Kontext genau vorhersagen können, ob eine Verbindung zwischen zwei Entitäten<sup>3</sup> besteht oder nicht.

In einfacheren Systemen können gute Vorhersagen durch die Ableitung von Regeln erreicht werden. Ein Beispiel hierfür ist die Regel „Der Feind meines Feindes ist mein Freund“. Durch diese Regel lassen sich neue, zuvor nicht vorhandene Relationen vorhersagen. In komplexen Systemen hingegen wird auf komplexere Algorithmen zurückgegriffen. Maschinelles Lernen hat sich in diesem Kontext in den letzten Jahren als vielversprechend erwiesen, befindet sich jedoch noch in einem frühen Stadium der Entwicklung.

In diesem Kontext ist der spezifische Use Case<sup>4</sup> (siehe das Aktivitäts Diagramm 1.1) dieser Arbeit die Konfiguration von Hochspannungsschaltanlagen. Hochspannungsschaltanlagen sind komplexe Systeme, die aus einer Vielzahl von Komponenten bestehen<sup>5</sup> und sich zu einem Produkt-System zusammenschließen. Die in dieser Arbeit präsentierten Technologien und Methoden könnten dazu beitragen, den Konfigurationsprozess von Hochspannungsschaltanlagen effizienter und einfacher zu gestalten. Insbesondere könnten Methoden des maschinellen Lernens und der Linkprediction

---

<sup>1</sup>siehe Definition 2.1

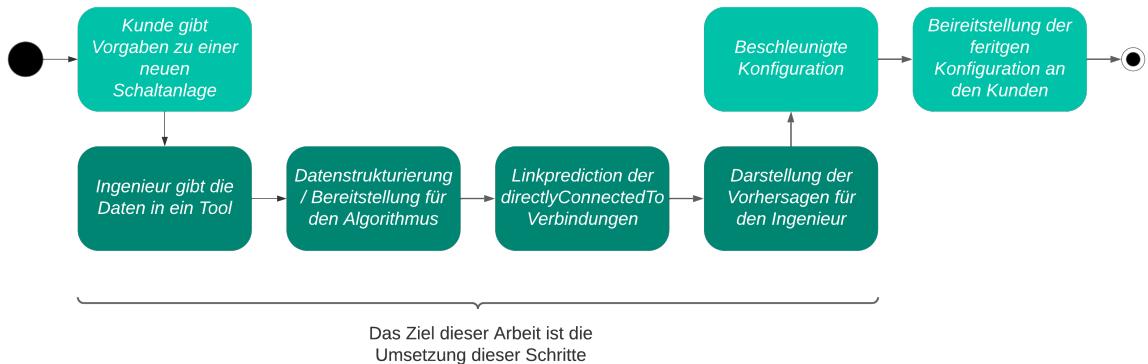
<sup>2</sup>Vorhersagen von Relationen

<sup>3</sup>Komponenten in einem PST

<sup>4</sup>deutsch Anwendungsfall

<sup>5</sup>siehe Kapitel 2.2

eingesetzt werden, um Kontextinformationen effektiv zu nutzen und präzise Vorhersagen über die Konfiguration der Anlage zu treffen.



**Abbildung 1.1: Aktivitätsdiagramm der Schaltanlagenkonfiguration:** Die in dunkelgrün hervorgehobenen Schritte verdeutlichen den Fokus dieser Arbeit und zeigen, wie der Einsatz von maschinellem Lernen und Link Prediction den Prozess optimiert und beschleunigt. Beginnend mit der Eingabe der Kundenanforderungen, wird die Datenstrukturierung und Bereitstellung für den Link Prediction Algorithmus durchgeführt. Der Algorithmus erzeugt Vorhersagen zu den GIS-Modulen (siehe Kapitel 2.2), die den Ingenieuren präsentiert werden, um die Konfigurationsgeschwindigkeit zu erhöhen. Schließlich wird die fertige Konfiguration dem Kunden bereitgestellt. (Quelle: eigene Darstellung)

## 1.2 Stand der Technik

KGs sind ein effektives Mittel zur Darstellung von Beziehungen in der realen Welt und werden daher auch für PSTs eingesetzt. In den letzten Jahren wurden verschiedene Knowledge-Graph-Embedding (KGE)-Ansätze entwickelt, um alle Komponenten von KGs in Vektorform darzustellen und die latenten Eigenschaften der Komponenten zu repräsentieren. KGE-Techniken werden in verschiedenen Anwendungsbereichen wie Linkprediction, KG-completion und Entity-classification eingesetzt.

Zu den frühen KGE-Modellen gehören Translation-based Modelle wie TransE [2], die Beziehungen zwischen Entitäten durch Vektoroperationen wie Addition und Subtraktion darstellen. Obwohl sie schnell und einfach zu implementieren sind, zeigen sie Einschränkungen in ihrer Expressivität. Im Gegensatz dazu verwenden Bilineare Modelle wie DistMult [33] und ComplEx [31] Matrixzerlegung und komplexe Zahlen, um Beziehungen zwischen Entitäten dataillierter zu modellieren. Diese Modelle sind leistungsfähiger, aber auch rechenintensiver als die Translation-basierten Modelle.

Neural Network-based Modelle nutzen neuronale Netze (siehe Definition 2.4.3) zur Modellierung von Beziehungen zwischen Entitäten in KGs. Hierbei sind Convolutional Neural Network (CNN)s und Graph Neural Network (GNN)s zwei vielversprechende Ansätze, die in jüngster Zeit entwickelt wurden. CNNs wurden bereits erfolgreich für die Modellierung von Bilddaten eingesetzt, und ihre Anwendung auf Graphdaten bietet interessante Möglichkeiten für die Informationsverarbeitung. GNNs können auf strukturierten Graphdaten arbeiten und bieten Vorteile wie End-to-End<sup>6</sup> überwachtes Lernen und verbesserte Leistung in verschiedenen Klassifikationsaufgaben.

Die Kombination von KGE-Methoden und GNNs eröffnet vielversprechende Möglichkeiten zur Verbesserung der Leistung bei Linkprediction und KG-completion. Verschiedene GNN-basierte KGE-Ansätze wurden vorgeschlagen, darunter das Graph Convolutional Network (GCN) und der Relational Graph Convolutional Network (R-GCN)-Ansatz. Allerdings haben diese Modelle noch Schwierigkeiten, semantische Informationen effektiv zu nutzen, und ihre Anwendung auf spezielle Domänen ist wenig erforscht oder evaluiert.

Da jeder Datensatz bzw. Knowledge Graph und jeder Use Case unterschiedlich ist, lässt sich kein eindeutiges State-of-the-Art-Modell bestimmen.

### 1.3 Ziel

Die Zielsetzung der Arbeit besteht darin, einen auf den spezifischen Anwendungsfall der Konfiguration von Hochspannungsschaltanlagen abgestimmten Machine Learning Algorithmus zu entwickeln. Im Mittelpunkt steht dabei die effektive Nutzung von Kontextinformationen zur Vorhersage von Verbindungen zwischen den Komponenten.

---

<sup>6</sup>End-to-End-Lernen bezieht sich auf einen Prozess, bei dem ein System alle Aufgaben, von der Eingabe bis zur Ausgabe, automatisch und ohne manuelle Zwischenschritte durchführt.

## 2 Grundlagen

In folgendem Kapitel wird diese Arbeit in den Kontext verwandter Arbeiten gesetzt und wichtige Begriffe wie auch Konzepte erklärt.

### 2.1 Knowledge Graph

Ein KG ist eine Datenstruktur, die darauf abzielt, Wissen in Form von Entitäten und ihren Beziehungen zueinander darzustellen. Dies wird in der Form eines gerichteten Graphen realisiert, sodass das Wissen maschinenlesbar wird. Knowledge Graphen werden genutzt, um bei logischen Schlussfolgerungen zu helfen und finden Anwendung in Bereichen wie Sprachverständnis, Empfehlungssysteme und anderen analytischen Aufgaben [10] [20]. Um das Konzept eines Knowledge Graphen besser zu verstehen, werden im Folgenden die grundlegenden Komponenten erläutert und ein Beispiel (siehe Abb. 2.1) gegeben:

#### Entitäten

Entitäten sind die grundlegenden Bausteine eines Knowledge Graphen und repräsentieren Personen, Orte, Objekte, Konzepte oder andere abstrakte oder konkrete Objekte, die in einem bestimmten Wissensbereich von Interesse sind, wie z.B. Hochspannungsanlagen. Entitäten können durch Attribute (z.B. Volumen) näher beschrieben werden.

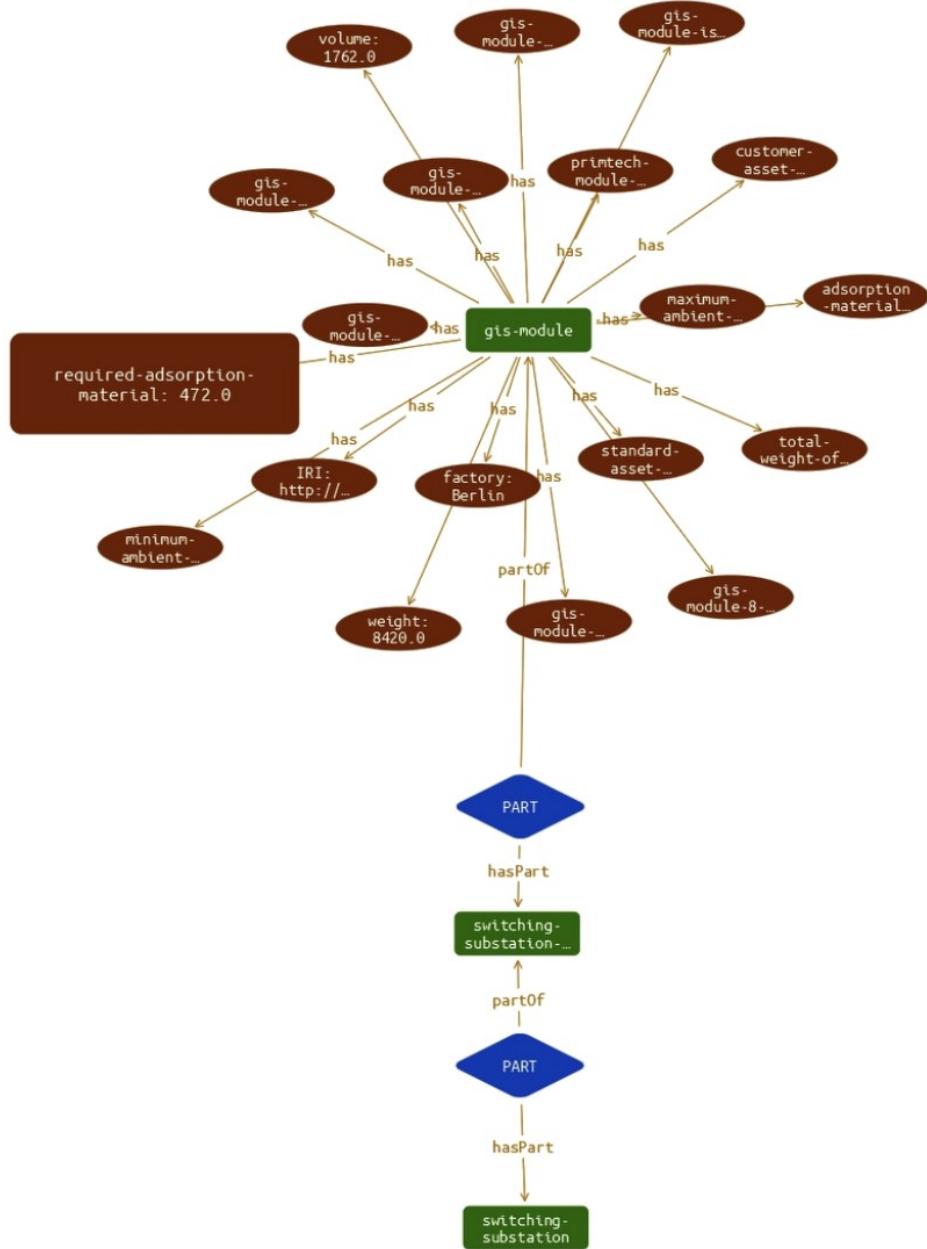
#### Relationen

Relationen sind die Verbindungen zwischen den Entitäten und beschreiben, wie diese Entitäten miteinander in Beziehung stehen. Relationen können unterschiedliche semantische Bedeutungen haben, wie z.B. „ist-Typ-von“, „hat-Teil“ oder andere semantische Konzepte.

#### Gerichteter Graph

Ein gerichteter Graph ist eine graphische Datenstruktur, bei der die Kanten zwischen den Knoten (in diesem Fall Entitäten) eine Richtung haben. Im Kontext eines KG bedeutet dies, dass die Beziehungen zwischen den Entitäten eine Richtung aufweisen, was zu einer klareren Repräsentation von Wissen führt. An dem KG 2.1 sind die Richtungen der Beziehungen sehr deutlich durch Pfeilköpfe dargestellt.

Ansätze für KG sind seit längerer Zeit bekannt. Der spezifische Begriff „Knowledge Graph“ allerdings wurde im Kontext von Googles Lancierung des „Google Knowledge Graph“ im Jahr 2012 populär [11]. Seitdem hat der Begriff an Bekanntheit gewonnen und wird häufig synonym zu den Ausdrücken „Ontologie“ und „Knowledge Base“ verwendet. Allerdings unterscheiden viele Forscher und Praktiker nicht signifikant zwischen diesen Begriffen, wie in [6] diskutiert wird.



**Abbildung 2.1: Ausschnitt aus einem KG:** Grüne Rechtecke sind Entitäten und repräsentieren Komponenten einer Hochspannungsanlage. Braune Kreise stellen Attribute dar und gehören zu der Entität „gis-module“ (deutlich durch die „has“ Pfeile). Relationen werden jeweils durch ein blaues Parallelogramm repräsentiert aus dem zwei Pfeile auf Entitäten zeigen, welche dadurch in Beziehung gesetzt werden. Um die Abbildung übersichtlich zu halten wurden die Beschriftungen gekürzt. Das hervorgehobene braune Rechteck dient lediglich zur Verdeutlichung der Attributinhalte eines GIS-Moduls. (Quelle: eigene Darstellung)

Im weiteren Verlauf dieser Arbeit wird bewusst der Ausdruck „Knowledge Graph“ verwendet, um eine klare Differenzierung von den anderen Begrifflichkeiten sicherzustellen. Der Begriff „Ontologie“ hat in diesem Zusammenhang eine andere Bedeutung: Er beschreibt die Struktur eines Datensystems, während ein KG diese Struktur mit spezifischen Instanzdaten ausfüllt und somit konkretisiert.

In den letzten Jahren hat die Forschung auf dem Gebiet der Knowledge Graphen erhebliche Fortschritte gemacht. Einige der wichtigsten Arbeiten in diesem Bereich sind die Einführung des DBpedia-Projekts [1], das darauf abzielt, strukturierte Informationen aus Wikipedia zu extrahieren und als KG zur Verfügung zu stellen, und die Entwicklung des YAGO-Projekts [28], das einen großen KG auf Basis von Wikipedia, WordNet und anderen Quellen erstellt.

Der aktuelle Stand der Technik im Bereich Knowledge Graphen umfasst sowohl die Erstellung als auch die Verbesserung bereits vorhandener Graphen und die Vervollständigung von KGs. Dazu gehören Methoden zur Extraktion von Wissen aus Texten, zur Integration verschiedener Wissensquellen und zur Anwendung von Machine Learning Techniken, um Vorhersagen und Schlussfolgerungen aus den in Knowledge Graphen enthaltenen Informationen abzuleiten.

## 2.2 Domäne

In dieser Arbeit liegt der Fokus auf der Konfiguration von Schaltanlagen<sup>1</sup> (siehe Abb. 2.2a). Schaltanlagen sind komplexe Systeme, die als Knotenpunkte und Verteilerstellen für elektrische Energie in verschiedenen Spannungsebenen fungieren. Sie bestehen aus einer Vielzahl von Betriebsmitteln, wie Schaltgeräten, Messeinrichtungen und weiteren Komponenten, die an einem zentralen Ort zusammengeführt werden.



(a) Schaltanlage



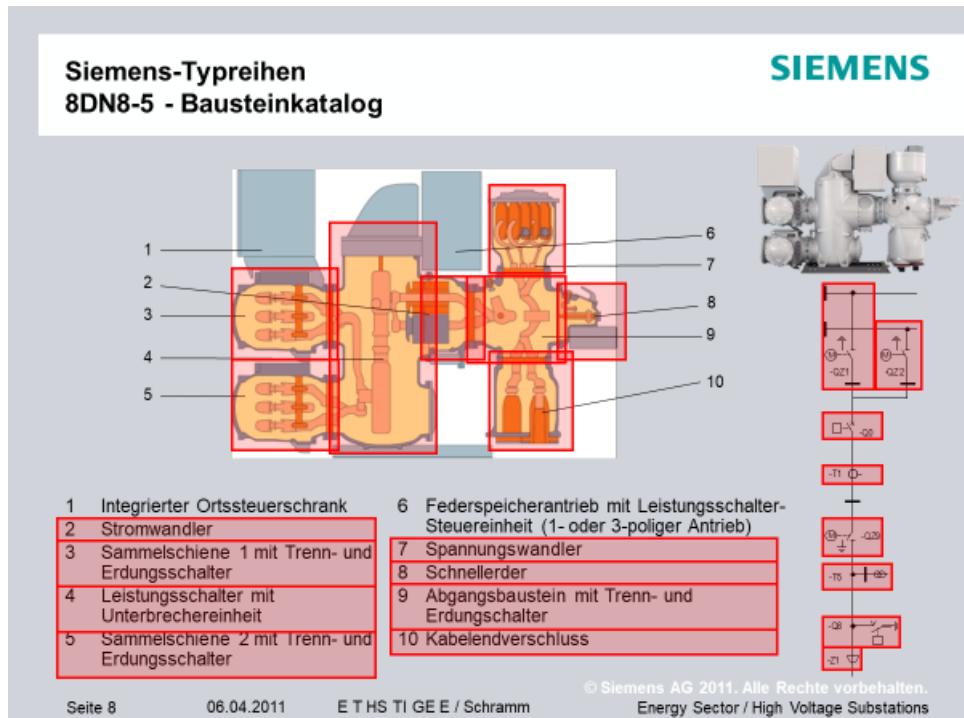
(b) Feld einer Schaltanlage

Abbildung 2.2: **Schaltanlage und Feld:** Bilder von einer realen Schaltanlage (a) und einem realen Feld (b). (Quelle: Siemens Energy, 2023)

<sup>1</sup>engl. switchgear

Die Hauptaufgabe einer Schaltanlage besteht darin, elektrische Verbindungen zwischen Freileitungen, Kabeln, gasisolierten Übertragungsleitungen und Transformatoren herzustellen oder zu trennen. Im Zentrum einer Schaltanlage befinden sich eine oder mehrere Sammelschienen, von denen aus die Verteilung des elektrischen Stroms auf die einzelnen Schaltfelder<sup>2</sup> (siehe Abb. 2.2b) erfolgt, einschließlich Einspeisefeldern und Abgangsfeldern.

Jedes Schaltfeld besteht aus mehreren Komponenten, auch GIS<sup>3</sup>-Module genannt. Abb. 2.3 veranschaulicht beispielhaft den Aufbau eines solchen Feldes und die Anordnung der GIS-Module. Wie in der Abbildung zu sehen ist, sind die Module miteinander verbunden.



**Abbildung 2.3: Feld mit GIS-Modulen:** Dies ist ein exemplarisches Feld bestehend aus unterschiedlichen GIS-Modulen. Die Module sind innerhalb des Feldes visuell dargestellt und benachbarte Module sind durch *directlyConnectedTo*-Verbindungen miteinander verknüpft. Ein Beispiel für solch eine Verbindung ist die zwischen dem Stromwandler (2) und dem Leistungsschalter mit Unterbrechereinheit (4). (Quelle: Siemens Energy, 2023)

Das Hauptziel dieser Arbeit ist es, die Verbindungen<sup>4</sup> zwischen den GIS-Modulen vorherzusagen.

<sup>2</sup>engl. switchgear bay

<sup>3</sup>engl. Gas insulated switchgear

<sup>4</sup>*directlyConnectedTo*-Relationen, siehe Abb. 2.4

Es ist wichtig zu beachten, dass GIS-Module als Teile von Schaltfeldern wiederum aus Baugruppen bestehen, die konfiguriert werden können. Dieser Aspekt wird in dieser Arbeit nicht im Detail behandelt, bietet jedoch einen interessanten Ansatzpunkt für zukünftige Untersuchungen (siehe Kapitel 5.2).

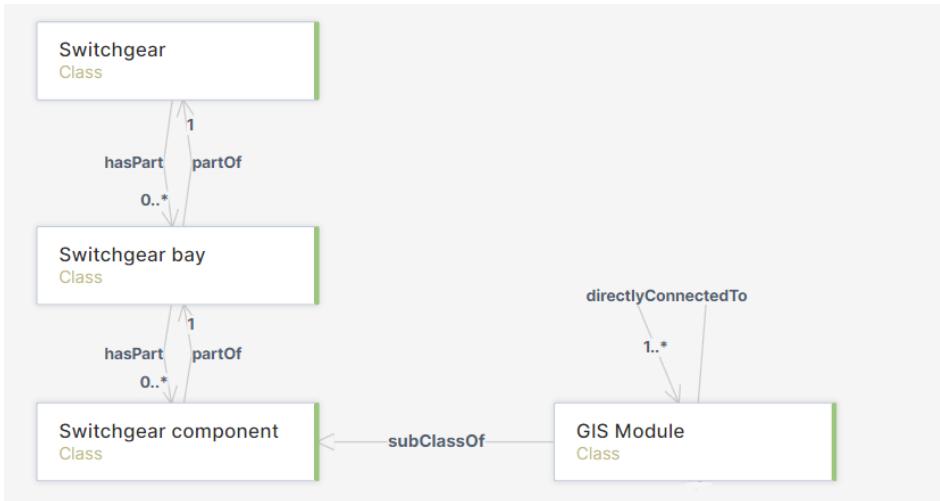


Abbildung 2.4: **Ontologie:** Die Ontologie des zu betrachtenden Knowledge Graphen. Eine Schaltanlage kann aus mehreren Feldern bestehen. Felder können aus mehreren Komponenten bestehen. Komponenten sind GIS-Modul. Diese sind miteinander verbunden. (Quelle: eigene Darstellung)

Die in dieser Arbeit verwendete Ontologie ist in Abb. 2.4 dargestellt und bietet einen Überblick über die Struktur und die Beziehungen zwischen den verschiedenen Komponenten innerhalb der Schaltanlagen. Eine Schaltanlage wird durch ihren Namen und mehrere Attribute beschrieben, wie zum Beispiel den Kundennamen, das Land der Herstellung, etc. Ebenso werden die GIS-Module durch ihre Namen und verschiedene Attribute charakterisiert. Schaltfelder verfügen lediglich über ein Attribut, ihre Bezeichnung (siehe Abb. 2.5).

Die Abbildung 2.5 veranschaulicht einen Ausschnitt des Knowledge-Graphen, der ein Schaltfeld repräsentiert. Das Schaltfeld ist Teil einer Schaltanlage und besteht aus mehreren Komponenten, die miteinander verbunden sind. Diese Verbindungen müssen von Ingenieuren manuell berechnet und eingefügt werden.

Das Ziel dieser Arbeit ist es, durch den Einsatz von Link Prediction-Methoden die Verbindungen zwischen den GIS-Modulen vorherzusagen und anschließend in den KG zu integrieren. Dies ermöglicht eine effizientere und zuverlässigere Konfiguration der Schaltanlagen und führt zu einer Einsparung von Ressourcen.

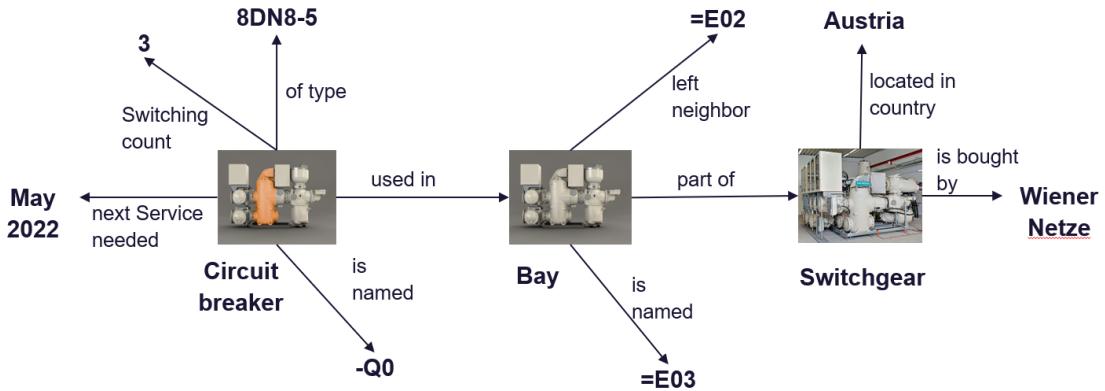


Abbildung 2.5: **Schaltanlage als KG:** Ausschnitt einer Schaltanlage als KG, basierend auf der Ontologie in Abb. 2.4. Der Circuit Breaker ist ein GIS-Modul. Sowohl Schaltanlage als auch GIS-Modul werden durch diverse Attribute näher beschrieben. Felder haben lediglich einen Namen als Attribut. (Quelle: In Anlehnung an Siemens Energy, 2023)

## 2.3 Link Prediction

Link Prediction repräsentiert eine bedeutende Herausforderung im Kontext von Knowledge Graphen sowie allgemeinen Graphen. Dabei wird versucht, durch die Analyse bereits existierender Informationen im Knowledge Graphen sowie durch Erkennen von Regeln, Mustern und Strukturen, zukünftige oder versteckte Beziehungen zwischen Entitäten zu prognostizieren. Da Graphen in vielen Bereichen des digitalen Raums präsent sind, findet Link Prediction vielfältige Anwendungen, etwa in sozialen Netzwerken für Freundesempfehlungen [32], bei Streaming-Diensten für Filmempfehlungen [16] und in der Vervollständigung von Knowledge Graphen, die wiederum in zahlreichen Bereichen Anwendung findet.

Die jüngsten Jahre brachten bemerkenswerte Fortschritte in der Forschung zu Link Prediction hervor. Hervorzuheben sind etwa die Arbeiten von Zhang und Chen [34], die Graph Neural Networks (GNNs) zur Vorhersage von Verknüpfungen in Knowledge Graphen einführten. GNNs bieten einen innovativen Ansatz zur Modellierung strukturierter Daten, der die Extraktion und Nutzung von Informationen aus lokalen und globalen Strukturen in Graphen ermöglicht. Zhang und Chen demonstrieren in ihrer Arbeit, dass GNNs eine bessere Leistung als herkömmliche Methoden wie Matrix-Faktorisierung [16] und Random-Walk-basierte [32] Ansätze liefern.

Die aktuellen Techniken zur Link prediction umfassen ein breites Spektrum an Methoden und Algorithmen, einschließlich Machine-Learning-Techniken wie translation-basierte Modelle (z.B. TransE [2]), bilineare Modelle (z.B. DistMult [33], ComplEx [31]) und Graph Neural Networks (z.B. R-GCN [25], GraphSAGE [8]). Diese Methoden unterscheiden sich in ihrer Fähigkeit, verschiedene Arten von Beziehungen, semantische Informationen und strukturelle Muster in Knowledge Graphen zu erfassen und zu modellieren.

Trotz der vielversprechenden Entwicklungen in dem Feld der Link Prediction in Knowledge Graphen bleibt dieses Gebiet ein relativ neues Forschungsfeld. Es besteht weiterhin ein großer Bedarf an Forschung, um die Effizienz und Robustheit dieser Ansätze weiter zu verbessern und auf verschiedene Anwendungsfälle zu adaptieren.

## 2.4 Machine Learning

Machine Learning (ML) ist ein Teilgebiet der künstlichen Intelligenz, das darauf abzielt, Computern beizubringen, Muster und Zusammenhänge in Daten zu erkennen und darauf basierend Entscheidungen oder Vorhersagen zu treffen. In der Regel werden Algorithmen und Modelle entwickelt, die anhand von Trainingsdaten lernen und anschließend auf unbekannte Daten angewendet werden können, um Vorhersagen zu treffen. Im Kontext von Knowledge Graphen ist ML von großer Bedeutung, insbesondere bei der link prediction.

ML lässt sich in drei Kategorien von Lernmethoden unterteilen:

### Überwachtes Lernen:

Beim überwachten Lernen besteht der Trainingsdatensatz aus Eingabe-Ausgabe-Paaren, wobei der Ausgabewert (auch bekannt als Label oder Target) für jede Eingabe bekannt ist. Der Algorithmus lernt Muster in den Eingabedaten zu erkennen und den Zusammenhang zwischen Eingabe und Ausgabe abzubilden. Im Kontext der Link Prediction in Knowledge Graphen bedeutet dies, dass ein Modell anhand von Beispielen von vorhandenen Verbindungen (positive edge) und nicht vorhandenen Verbindungen (negative edge) trainiert wird, um zu lernen, ob eine Verbindung zwischen zwei Entitäten besteht oder nicht.

### Unüberwachtes Lernen:

Beim unüberwachten Lernen sind die Ausgabewerte (Labels) für die Trainingsdaten nicht bekannt. Der Algorithmus versucht, die Struktur und Muster in den Daten eigenständig zu erkennen und zu modellieren. Im Kontext von Knowledge Graphen kann unüberwachtes Lernen beispielsweise zur Identifizierung von Clustern von ähnlichen Entitäten oder zur Extraktion von latenten Merkmalen aus den Daten verwendet werden.

### Teilweise überwachtes Lernen:

Teilweise überwachtes Lernen kombiniert Elemente aus überwachtem und unüberwachtem Lernen. Dabei sind einige Beispiele im Trainingsdatensatz beschriftet, während andere nicht beschriftet sind. Dieses Lernverfahren kann in Situationen nützlich sein, in denen beschriftete Daten teuer oder schwierig zu beschaffen sind. Im Kontext von Knowledge Graphen kann teilweise überwachtes Lernen dazu verwendet werden, fehlende Verbindungen oder Attributwerte unter Berücksichtigung sowohl der beschrifteten als auch der unbeschrifteten Beispiele vorherzusagen.

Für den in dieser Arbeit gestellten Anwendungsfall und Datensatz wird ein überwachtes Training benutzt.

### 2.4.1 Induktives und transduktives Lernen

Induktives Lernen bezieht sich auf die Aufgabe des Lernens von allgemeinen Regeln aus Trainingsdaten, um Vorhersagen auf neuen, bisher ungesiehenen Daten zu treffen. Im Gegensatz dazu bezieht sich transduktives Lernen auf die Aufgabe des Lernens von Vorhersagen für spezifische, unbekannte Datenpunkte, basierend auf den Eigenschaften der bereits bekannten Datenpunkte. Während induktives Lernen allgemeine Modelle entwickelt, die auf neuen Daten generalisiert werden können, konzentriert sich transduktives Lernen auf die Vorhersage von Labels für eine spezifische Menge von Datenpunkten.

In dieser Arbeit soll ein Modell entwickelt werden, das in der Lage ist, neue ungesiehe Knoten und Kanten sowie deren lokale und globale Rollen im Graphen zu erkennen. Daher wird ein induktiver Ansatz verwendet.

### 2.4.2 Training von Machine Learning-Modellen

ML-Modellen unterlaufen im Allgemeinen ein Training. Dabei wird das Modell anhand der Trainingsdaten optimiert, um eine bestimmte Zielfunktion<sup>5</sup> zu minimieren. Der Begriff „Lernen“ bezieht sich auf die Optimierung des Modells während des Trainings. Die Zielfunktion misst den Unterschied zwischen den Vorhersagen des Modells und den tatsächlichen Zielwerten der Trainingsdaten. Im Kontext der Link Prediction in Knowledge Graphen könnte die Zielfunktion beispielsweise den Unterschied zwischen den vorhergesagten Wahrscheinlichkeiten für das Vorhandensein von Verbindungen und den tatsächlichen Verbindungen messen.

Es gibt verschiedene Optimierungsverfahren, um die Zielfunktion zu minimieren und das Modell anzupassen. Ein weit verbreitetes Optimierungsverfahren ist der gradient descent [24], bei dem die Modellparameter schrittweise in Richtung des negativen Gradienten der Zielfunktion angepasst werden, um das Minimum zu finden. Es gibt auch mehrere Varianten des gradient descent, wie den stochastischen gradient descent (SGD) und adaptive Methoden wie Adam [12], die darauf abzielen, schneller und effizienter zu konvergieren.

Darüber hinaus spielen Regularisierungstechniken eine wichtige Rolle beim Training von ML-Modellen, um Overfitting zu verhindern. Overfitting tritt auf, wenn das Modell zu stark an die Trainingsdaten angepasst ist und nicht gut auf unbekannte Daten generalisiert.

---

<sup>5</sup>engl. loss function

### 2.4.3 Neuronale Netzwerke

Ein bedeutender Bereich des Machine Learning umfasst neuronale Netzwerke, die aus Schichten von Neuronen bestehen, die miteinander verbunden sind. Sie sind in der Lage, komplexe nichtlineare Muster in Daten zu erkennen, indem sie die Gewichtungen ihrer Verbindungen während des Trainings anpassen. Neuronale Netzwerke können für verschiedene Arten von Aufgaben verwendet werden, einschließlich Klassifikation, Regression, Clustering und Anomalieerkennung. Speziell im Kontext von Knowledge Graphen werden GNNs verwendet, die in der Lage sind, Informationen über die Graphstruktur zu erfassen und zu nutzen.

Ein neuronales Netzwerk besteht aus mehreren Schichten<sup>6</sup>, wobei jede Schicht eine Reihe von Neuronen oder Knoten enthält. Jedes Neuron in einer Schicht ist mit jedem Neuron in der nächsten Schicht verbunden (siehe Abb. 2.6). Jede dieser Verbindungen hat ein Gewicht, das den Einfluss des einen Neurons auf das andere bestimmt. Darüber hinaus hat jedes Neuron eine Bias-Einheit und eine Aktivierungsfunktion, die bestimmt, wie die Ausgabe des Neurons berechnet wird.

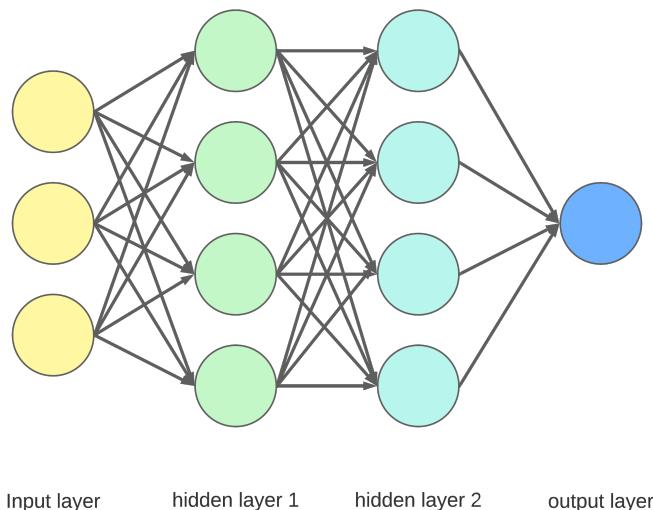
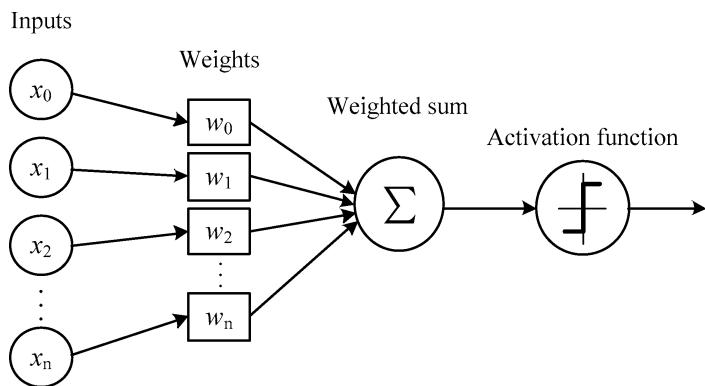


Abbildung 2.6: **Einfaches Neuronales Netzwerk:** Das Modell besteht aus vier Schichten: Input layer, Output layer und zwei Hidden layer. Als Input nimmt es drei Werte und gibt einen Wert als Output zurück. (Quelle: eigene Darstellung)

Um zu verdeutlichen, wie Informationen durch das Netzwerk fließen, kann der sogenannte „Forward Pass“ herangezogen werden. Während eines Forward Passes werden die Eingabedaten durch das Netzwerk geleitet. Sie beginnen in der Eingabeschicht und durchlaufen alle Zwischenschichten (die sogenannten „Hidden layers“), bis sie schließlich die Ausgabeschicht erreichen. Bei jedem Schritt wird die Ausgabe eines Neurons berechnet, indem die Eingaben mit den Gewichten summiert und dann durch die Aktivierungsfunktion geleitet werden.

<sup>6</sup>engl. layer

Ein Perceptron, die grundlegende Einheit eines neuronalen Netzwerks, illustriert diesen Prozess, siehe Abbildung 2.7. Es nimmt mehrere Eingaben entgegen, multipliziert jede Eingabe mit einem zugehörigen Gewicht und summiert diese gewichteten Eingaben zusammen. Ein Biaswert wird hinzugefügt und die resultierende Summe wird durch eine Aktivierungsfunktion geleitet, um die endgültige Ausgabe zu generieren. Dieser Prozess stellt den Forward Pass in einem Neuron dar.



**Abbildung 2.7: Schematische Darstellung eines Perceptrons:** Ein Perceptron repräsentiert ein Neuron in einem neuronalen Netzwerk. Eingaben ( $x_1, x_2, \dots, x_n$ ) werden jeweils mit den zugehörigen Gewichten ( $w_1, w_2, \dots, w_n$ ) multipliziert und dann summiert. Zu dieser Summe wird ein Bias addiert. Die resultierende Summe wird durch eine Aktivierungsfunktion (z.B. die sigmoid Funktion) geleitet, um die Ausgabe zu erzeugen. Dieser Prozess illustriert den Forward Pass in einem Perceptron. (Quelle: [9])

Das Resultat eines Forward Passes ist die Ausgabe des Netzwerks, welche die Vorhersage basierend auf den gegebenen Eingabedaten darstellt.

Verschiedene Arten von neuronalen Netzwerken sind geeignet für unterschiedliche Arten von Aufgaben und Daten:

- ▶ Feedforward-Neuronale Netzwerke (FFNN): Dies ist die einfachste Art von neuronalem Netzwerk, bei dem die Informationen von der Eingabeschicht über die versteckten Schichten zur Ausgabeschicht fließen.
- ▶ Convolutional Neural Networks (CNN): CNNs sind besonders geeignet für Bildverarbeitungsaufgaben, da sie die räumliche Struktur der Daten erfassen können. Sie sind bekannt für ihre Fähigkeit, Bildklassifikation und andere bildbezogene Aufgaben durchzuführen.
- ▶ Recurrent Neural Networks (RNN): RNNs können sequenzielle Daten verarbeiten und sind daher geeignet für Aufgaben wie Textanalyse oder Zeitreihenvorhersage.
- ▶ Graph Neuronale Netze (GNN): Diese Netze sind in der Lage, direkt auf Graphen zu arbeiten und können daher die Struktur des Graphen nutzen, um Vorhersagen zu treffen. Sie sind besonders geeignet für Aufgaben im Zusammenhang mit Knowledge Graphen, wie z.B. Link Prediction oder Knotenklassifikation.

## 2.5 Embedding

Ein Embedding ist eine Darstellung eines Objekts und dessen Beziehungen als Vektor in einem niederdimensionalen Raum. Im Kontext von neuronalen Netzwerken müssen alle Eingaben numerisch sein, was erfordert, dass Knoten und Attribute als Vektoren mit fester Länge dargestellt werden. Eine häufig verwendete Methode zur Darstellung kategorischer Daten ist das One-Hot-Encoding.

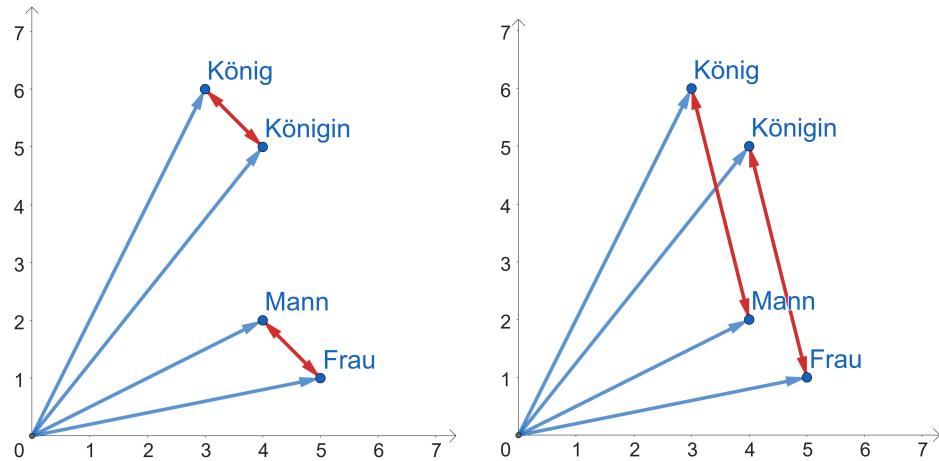
One-Hot-Encoding ist ein Verfahren, bei dem für jede Kategorie ein Vektor erstellt wird, der die Länge der einzigartigen Elemente in der Kategorie hat und mit Nullen gefüllt ist, mit Ausnahme der Position, die dem Wert der Kategorie entspricht, die auf Eins gesetzt wird. Das bedeutet, dass der Vektor für jede Kategorie nur eine einzige Eins hat und der Rest Nullen sind. Beispielsweise würde bei der Darstellung der vier Wörter Mann, Frau, König und Königin die One-Hot-Vektor auf folgende Weise konstruiert werden:

Mann	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0
1	0	0	0		
Frau	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	1	0	0
0	1	0	0		
König	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	1	0
0	0	1	0		
Königin	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	1
0	0	0	1		

Abbildung 2.8: **One-Hot-Vektor:** Die Länge der Vektoren beträgt 4, entsprechend der Anzahl der Wörter. Jedes Wort ist durch eine eigene Spalte repräsentiert. Es existiert keine Korrelation zwischen den verschiedenen Repräsentationen. (Quelle: eigene Darstellung)

Das Problem bei One-Hot-Encoding ist, dass man einen großen Vektor mit vielen Nullen und einer Eins erhält, was bei Variablen mit einer Vielzahl einzigartiger Merkmale zu einem sehr großen Vektor führt. Darüber hinaus haben die Kategorien im Vektorraum keine semantische Beziehung zueinander, was bedeutet, dass die Distanz zwischen den Vektoren keine Information über die Ähnlichkeit der Kategorien liefert.

Im Gegensatz dazu berücksichtigt der resultierende Vektorraum beim Embedding die semantischen Zusammenhänge zwischen den Kategorien [17]. Embedding-Vektoren, die sich im Raum näher zueinander befinden, weisen eine höhere semantische Ähnlichkeit auf (siehe Abb. 2.9). Ein niederdimensionaler Vektorraum ermöglicht es, die Komplexität der Daten zu reduzieren und die Rechenanforderungen für maschinelle Lernverfahren wie Link Prediction zu verringern, während gleichzeitig die semantischen Zusammenhänge zwischen den Kategorien beibehalten werden.



**Abbildung 2.9: Embedding:** Die Darstellung des Vektorraums ist auf den zweidimensionalen Raum reduziert. Die Wörter „Mann“ und „Frau“ bzw. „König“ und „Königin“ liegen näher beieinander als die Wörter „Königin“ und „König“, da sie semantisch stärker korreliert sind. Zudem ist der Abstand bzw. der Abstandsvektor zwischen diesen beiden Wortpaaren der selbe. Dies deutet auf den selben semantischen Unterschied und die selbe Korrelation zwischen den Wortpaaren hin. Das Gleiche gilt für die Wortpaare „König“, „Mann“ und „Königin“, „Frau“. (Quelle: eigene Darstellung)

Im Kontext der Link Prediction wird die Nähe von Knoten-Embeddings im Vektorraum genutzt, um semantische Zusammenhänge zu erfassen und auf dieser Grundlage eine Vorhersage über die Beziehung zwischen den Knoten zu treffen. Das übergeordnete Ziel dieser Arbeit besteht darin, ein Modell zu entwickeln, das diese Embedding-Funktion erlernen kann. Im nächsten Abschnitt wird ein entsprechender Ansatz vor gestellt.

## 2.6 GraphSAGE

### 2.6.1 GraphSAGE: Kontext und Motivation

GraphSAGE (Graph Sample and Aggregate)[8] stellt eine Methode für das induktive Lernen auf Graphen dar. Seine Stärken liegen insbesondere im Umgang mit umfangreichen oder dynamischen Graphen, die fortlaufend durch neue Knoten und Kanten erweitert werden. Für derartige Graphen wäre es unverhältnismäßig ressourcen- und zeitaufwendig, das gesamte Modell für jeden neu hinzukommenden Knoten zu retrainieren.

Im Gegensatz dazu ermöglicht GraphSAGE das effiziente Extrahieren von Informationen aus den Attributen der Knoten, wodurch es Embeddings für bisher ungesehene Daten erstellen kann. Anstelle eines separaten Embeddings für jeden einzelnen Knoten, lernt das Modell eine Funktion, die Embeddings basierend auf den Nachbarnknoten generiert.

### 2.6.2 Funktionsprinzip von GraphSAGE

Das GraphSAGE-Verfahren baut auf einer mehrschichtigen Struktur auf, welche auf einer definierten Abfolge von Aggregationsfunktionen beruht. Diese Funktionen bündeln Informationen aus der unmittelbaren Umgebung eines Knotens, um einen Embedding-Vektor für den betrachteten Knoten zu erzeugen. Der folgende Pseudo-Code stellt den Algorithmus der Embedderstellung dar, welcher anschließend erklärt wird.

---

**Algorithm 1:** GraphSAGE embedding generation (i.e., forward propagation) algorithm

---

**Input :** Graph  $\mathcal{G}(\mathcal{V}, \mathcal{E})$ ; input features  $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$ ; depth  $K$ ; weight matrices  $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$ ; non-linearity  $\sigma$ ; differentiable aggregator functions AGGREGATE $_k, \forall k \in \{1, \dots, K\}$ ; neighborhood function  $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

**Output :** Vector representations  $\mathbf{z}_v$  for all  $v \in \mathcal{V}$

```

1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$  ;
2 for  $k = 1 \dots K$  do
3   for  $v \in \mathcal{V}$  do
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ ;
5      $\mathbf{h}_v^k \leftarrow \sigma \left( \mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k) \right)$ 
6   end
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ 
8 end
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 

```

---

Abbildung 2.10: **GraphSAGE-Algorithmus:** Pseudocode der den Algorithmus für die Erstellung von Embeddings veranschaulicht. (Quelle: [8])

Der grundlegende Ablauf in GraphSAGE kann in drei Kernschritte unterteilt werden:

- 1 Sampling:** In dieser Phase wird für jeden Knoten  $v$  eine feste Anzahl von Nachbarn  $N(v)$  zufällig ausgewählt. Dieser Vorgang wird für jede Schicht des Modells durchgeführt, wobei die für die nachfolgende Schicht  $k \geq$  ausgewählten Nachbarn jeweils die Nachbarn der zuvor ausgewählten Knoten sind.
- 2 Aggregation:** Mit den aggregierten Informationen  $AGGREGATE_k()$  aus den ausgewählten Nachbarn wird der Vektor des Knotens  $h_{N(v)}^k$  berechnet. GraphSAGE bietet Unterstützung für diverse Aggregationsfunktionen, beispielsweise Mittelwert, Summe, Maximum oder benutzerdefinierte Funktionen.
- 3 Kombination:** Die aus der Nachbarschaft aggregierten Informationen werden zusammen mit den Eigenmerkmalen des Knotens in der Funktion  $CONCAT()$  vereint. Dies geschieht typischerweise durch eine vollständig verbundene Schicht, die sowohl die Merkmale des Knotens als auch die aggregierten Merkmale der Nachbarn berücksichtigt.

Diese Schritte werden für jede Schicht des Modells iterativ wiederholt, wobei jeweils die Ausgabe der vorherigen Schicht als Eingabe für die nächste Schicht dient. In Abb. 2.11 ist eine visuelle Illustration des Algorithmus dargestellt. Am Ende dieses Prozesses entsteht für jeden Knoten im Graphen ein Vektor, der für diverse Anwendungsfälle wie Knotenklassifikation, Linkvorhersage oder Clustering genutzt werden kann.

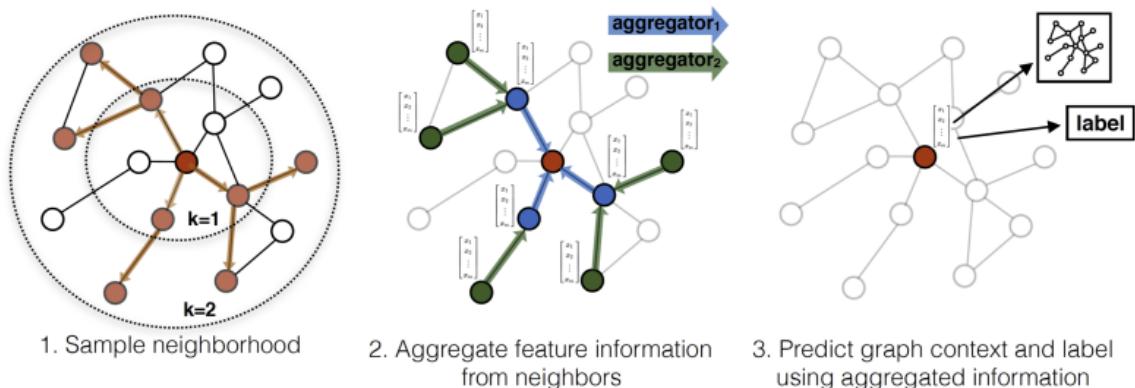


Abbildung 2.11: **Illustration von GraphSAGE:** Diese Darstellung veranschaulicht die Berechnung eines Embeddings für einen Knoten. Links ist das Sampling mit  $k = 2$  dargestellt. In der Mitte wird der Aggregations- schritt gezeigt. Rechts ist der Knoten mit seinem finalen Embedding- Vektor zu sehen, aus dem sowohl ein Label als auch der Kontext im Graphen erschlossen werden kann. (Quelle: [8])

Mit dem Verständnis dieser Grundlagen kann nun tiefer in die technischen Aspekte und die Umsetzung dieser Forschungsarbeit eingetaucht werden. Im nächsten Kapitel liegt der Fokus auf der Implementierung des Modells.

# 3 Implementation

In diesem Kapitel wird die Implementierung des gesamten Modells detailliert beschrieben. Dies beinhaltet die Beschreibung der Daten, das Preprocessing, die Architektur des Modells, das Training, die Integration der Vorhersagen in den Graphen und die Umsetzung in Python 3.11. Die Implementierung wurde mithilfe der folgenden Bibliotheken durchgeführt:

- ▶ pytorch 2.0.0
- ▶ pytorch-geometric 2.3.0
- ▶ scikit-learn 1.2.0
- ▶ pandas 1.5.3
- ▶ matplotlib==3.7.1

## 3.1 Datensatz

Der Datensatz, der als Basis für diese Arbeit dient, wurde aus einem Siemens Energy firmeninternen Knowledge Graphen extrahiert, der alle Komponenten der Schaltanlagen umfasst. Durch gezielte Queries<sup>1</sup> wurden die für das Modell relevanten Daten extrahiert und in .csv-Dateien<sup>2</sup> gespeichert. Dies stellt sicher, dass nur der relevante Teil des Knowledge Graphen für die weitere Analyse und Modellierung verwendet wird.

Dieser Datensatz besteht aus insgesamt sechs .csv-Dateien - drei für die Knoten und drei für die Kanten. Aus diesen Dateien wurde ein neuer Knowledge Graph erstellt, der sich vom firmeninternen Knowledge Graphen unterscheidet und speziell für diese Arbeit konzipiert wurde. Nach der Bereinigung von Duplikaten und unbrauchbaren Daten umfasst der erstellte Knowledge Graph 60.255 Knoten und 169.425 Kanten. Die genaue Verteilung der Knoten 3.1 und Kanten 3.2 im Datensatz wird in den folgenden Tabellen dargestellt.

<sup>1</sup>Queries, aus dem Englischen für „Anfragen“, bezeichnen in der Informatik Abfragen von Daten aus einer Datenbank oder einem Informationssystem. Sie sind in einer speziellen Abfragesprache, wie z.B. SQL (Structured Query Language), formuliert und ermöglichen es, gezielte Informationen aus umfangreichen Datenbeständen zu extrahieren.

<sup>2</sup>CSV (Comma Separated Values) ist ein Dateiformat, das Daten in einer tabellenartigen Struktur speichert.

Knoten	Schaltanlage	Feld	GIS-Modul
Anzahl	394	2664	57197
Kante	s_hasPart	f_hasPart	directlyConnectedTo
Anzahl	2652	57197	109576

Tabelle 3.1: Verteilung der Knoten im Datensatz.

Kante	s_hasPart	f_hasPart	directlyConnectedTo
Anzahl	2652	57197	109576
Kante	s_hasPart	f_hasPart	directlyConnectedTo
Anzahl	2652	57197	109576

Tabelle 3.2: Verteilung der Kanten im Datensatz.

*s\_hasPart* steht für die Relation: Schaltanlage hasPart Feld.  
*f\_hasPart* bezeichnet die Relation: Feld hasPart GIS-Modul.

Jede Entität im Datensatz wird durch eine eindeutige, nicht-numerische ID repräsentiert. Zusätzlich werden Schaltanlagen und GIS-Module durch verschiedene Attribute näher spezifiziert. einige Merkmale wurden für das Lernen nicht berücksichtigt, da deren Integration in das Datenmodell die Leistung der Vorhersage verringerten. Merkmale mit negativem Effekt wurden deshalb auf experimenteller Basis aus dem Modell entfernt.

Der erste Schritt vor dem eigentlichen Preprocessing besteht darin, die .csv-Dateien mithilfe der Pandas-Bibliothek<sup>3</sup> einzulesen und die Knoten zu modellieren. Dabei wird zunächst ein Mapping erstellt, das jeder Entität eine numerische ID zuweist. Danach wird für jede Schaltanlage und jedes GIS-Modul ein OneHotEncoder eingesetzt, um aus den Attributen einen maschinlesbaren Vektor zu erstellen.

Nachdem die Knoten modelliert wurden, werden die Kanten in Vektoren überführt, wobei das zuvor erstellte Mapping der Knoten verwendet wird. Anschließend wird ein *HeteroData*-Objekt erstellt, das zur Erzeugung eines heterogenen Graphen dient, der den spezifischen Anforderungen dieser Arbeit entspricht. Die in dieser Arbeit verwendeten Machine Learning Modelle benötigen jedoch einen ungerichteten Graphen als Eingabe, obwohl sie für heterogene Graphen konzipiert sind. Daher wird aus dem erstellten gerichteten Graphen ein ungerichteter Graph erzeugt, indem für jede Kante eine entsprechende Kante in entgegengesetzter Richtung hinzugefügt wird. Diese werden als *rev* Kanten (*rev\_hasPart*) bezeichnet.

Zusammenfassend kann gesagt werden, dass die Daten aus dem firmeninternen Knowledge Graphen extrahiert, in eine maschinlesbare Form überführt und für die Verwendung in ML Modellen vorbereitet werden. Durch die gezielte Bereinigung und Auswahl der Daten wird sichergestellt, dass das Modell auf relevanten und aussagekräftigen Daten trainiert wird. Dieser Prozess ermöglicht eine effiziente und zuverlässige Modellierung der Schaltanlagen und trägt zur Optimierung der Ressourcennutzung bei.

<sup>3</sup>Die Bibliothek pandas ist eine Open-Source-Datenanalyse- und -manipulationsbibliothek für die Programmiersprache Python. Sie bietet Datenstrukturen und Funktionalitäten zur Manipulation und Analyse von strukturierten Daten.

Mit der Strukturierung und Vorverarbeitung des Datensatzes wurde ein solider Ausgangspunkt für die folgenden Modellierungsschritte geschaffen. Als nächstes folgt eine tiefergehende Betrachtung des Preprocessing. Hierbei werden die spezifischen Maßnahmen erläutert, die zur weiteren Aufbereitung des Datensatzes und zur Gewährleistung seiner Eignung für das angestrebte Modell vorgenommen wurden.

## 3.2 Preprocessing

Die zentrale Zielsetzung dieser Arbeit liegt in der Prognose, zwischen welchen GIS-Modulen eine *directlyConnectedTo*-Relation besteht, und zwar auf Grundlage eines vorgegebenen Input-Baums (Abbildung 3.1). Diese Bäume stellen ausgewählte Ausschnitte des KG dar, die die Felder mit ihren relevanten Entitäten und Relationen abbilden, allerdings ohne Einbezug der *directlyConnectedTo*-Relationen. Um eine solche Prognoseleistung zu erbringen, muss das Modell auf der Basis dieser speziellen Bäume trainiert werden. Die Konstruktion und das Laden dieser Bäume erfolgen dabei mithilfe des *TreeLoaders*.

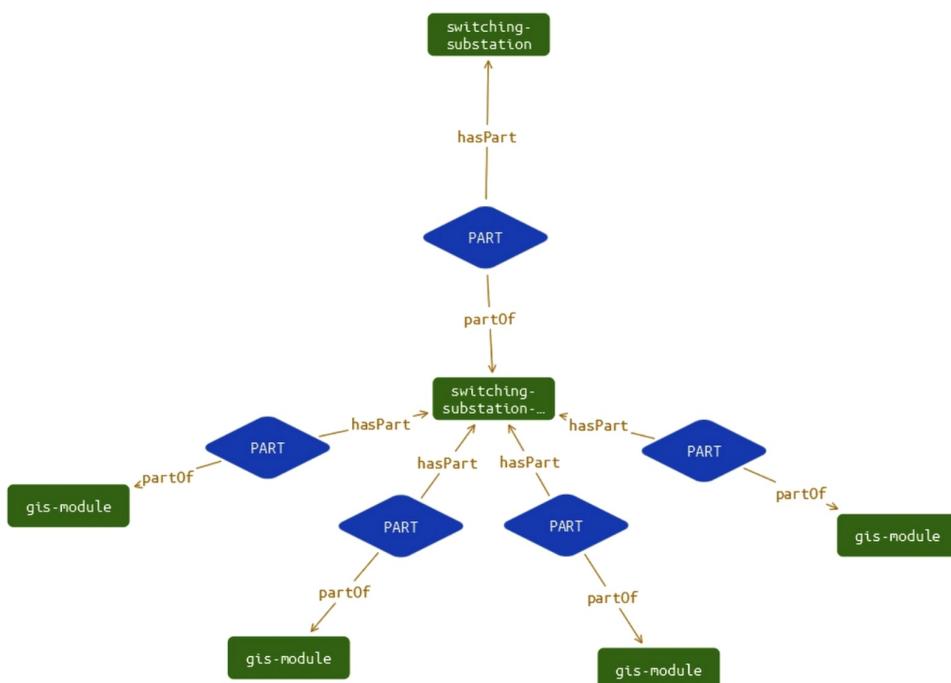


Abbildung 3.1: **Darstellung eines Input-Baums:** Dieses Diagramm bietet eine vereinfachte Darstellung eines Input-Baums. Attribute der GIS-Module und der Schaltanlage sowie die *rev*-Relationen sind in dieser Ansicht ausgelassen. Grüne Rechtecke repräsentieren die Entitäten, während blaue Parallelogramme die Relationen symbolisieren. Ein charakteristisches Merkmal eines Input-Baums ist die Abwesenheit von *directlyConnectedTo*-Verbindungen. (Quelle: eigene Darstellung)

Der *TreeLoader* fungiert als ein neuer Datensatz, für den aus jedem Feld einen Baum konstruiert und gespeichert wird. Dabei wird über alle Felder iteriert und für jedes Feld wird über die zuvor erwähnte *rev\_hasPart*-Relation die jeweilige Schaltanlage abgerufen. Anschließend werden alle zugehörigen GIS-Module hinzugefügt.

Jeder dieser Bäume wird als separates *HeteroData*-Objekt in einer Liste gespeichert.

Anschließend wird diese Liste mittels der Methode *random\_split* in einem Verhältnis von 70 zu 30 in einen Trainings- und einen Validierungsdatensatz unterteilt. Der Trainingsdatensatz dient als Grundlage für das Training des Modells und die Anpassung seiner Parameter, während der Validierungsdatensatz zur Evaluierung der Modellleistung genutzt wird und vom Modell während des Trainings nicht berücksichtigt wird. Beim Laden der Bäume aus den Datensätzen werden Batches bzw. Mini-batches vermieden. Jeder Baum wird dem Modell einzeln übergeben (Batch-Größe von 1).

Insgesamt ermöglicht dieser spezielle Preprocessing-Schritt die Konstruktion von Trainings- und Validierungsdatensätzen, die auf die spezifischen Anforderungen dieser Arbeit abgestimmt sind. Durch die Berücksichtigung der spezifischen Struktur der Daten und die gezielte Aufbereitung dieser Daten wird sichergestellt, dass das Modell optimal trainiert und validiert werden kann.

Im nächsten Kapitel liegt der Fokus auf der Architektur des Modells, welches verwendet wird, um die Verbindungen zwischen den GIS-Modulen vorherzusagen. Hierbei werden die spezifischen Komponenten und Techniken, die in dieser Arbeit zum Einsatz kommen, genauer betrachtet und erläutert, wie sie zum Erreichen der Ziele dieser Arbeit beitragen.

## 3.3 Das Modell

Im Zentrum dieser Arbeit steht die Nutzung und Anpassung von aktuellen, etablierten Modellen<sup>4</sup>. Ein bedeutender Bestandteil dieses Modells ist das *SAGEConv* Layer, das als Herzstück des Modells dient. Das *SAGEConv* Layer erfüllt die zwei Hauptkomponenten: dem Aggregations- und dem Aktualisierungsschritt, die beide in der GraphSAGE-Methodik<sup>5</sup> beschrieben sind.

Der Parameter  $k$ , welcher im Kapitel 2.6 definiert wurde, bezieht sich auf die Anzahl der Hops, welche der Anzahl der *SAGEConv*-Schichten entspricht. Würde man beispielsweise 3 *SAGEConv*-Schichten verwenden, wäre  $k = 3$ .

---

<sup>4</sup>state-of-the-art

<sup>5</sup>siehe Kapitel 2.6

Für die Aggregation wird der Durchschnitts-Aggregator (mean aggregator) verwendet. Dieser berechnet den Durchschnittswert der Nachbarschaft eines bestimmten Knotens. Der Aktualisierungsschritt, andererseits, bildet eine lineare Kombination aus den repräsentativen Vektoren der Nachbarn und der neuen Repräsentation des Knotens selbst 3.1. Die Gewichtungen in dieser linearen Kombination werden im Laufe des Trainingsprozesses angepasst und optimiert.

$$x'_i = W_1 x_i + W_2 \cdot \text{mean}_{j \in N(i)} x_j \quad (3.1)$$

**Formel 3.1: Aggregationsformel:**  $x'_i$  ist die Repräsentation des Knotens in dem Schritt  $i$ . Der *mean* Teil ist der Aggregationsschritt in dem der Durchschnittswert der Nachbarschaft berechnet wird.  $W_1$  und  $W_2$  sind die Wichtungsvektoren, welche im Laufe des Trainings angepasst werden. (Quelle: In Anlehnung an [30])

Ein zentraler Aspekt, nämlich das Message Passing<sup>6</sup> wird durch das PyTorch Geometric (PyG)-Framework<sup>7</sup> übernommen. Das Sampling der Nachbarschaften der Knoten, wurde ersetzt durch das Sampling der Bäume, welches im vorherigen Kapitel beschrieben wurde.

In diesem Modell arbeiten das Sampling und das Message Passing zusammen, um effiziente und genaue Embeddings für die Knoten in einem Graphen zu erstellen. Das Sampling bestimmt, welche Knoten Informationen austauschen, und das Message Passing bestimmt, wie diese Informationen ausgetauscht und zur Aktualisierung der Knotenzustände verwendet werden.

Im weiteren Verlauf dieses Kapitels erfolgt eine ausführlichere Erläuterung des Modells anhand des Quellcodes 3.1.

---

<sup>6</sup>Im Kontext der Graphentheorie und des ML bezieht sich Message Passing auf den Prozess des Austausches von Informationen zwischen den Knoten eines Graphen. Dabei sendet jeder Knoten Informationen an seine Nachbarknoten und empfängt von diesen Informationen, wodurch eine umfassendere Darstellung des Knotens und seiner Rolle innerhalb des gesamten Graphen erstellt wird.

<sup>7</sup>PyG ist eine Erweiterung von PyTorch, die speziell für die Arbeit mit graphischen Daten konzipiert wurde. Es bietet eine Vielzahl von nützlichen Funktionen, einschließlich effizienter Methoden für Message Passing und Sampling, die in diesem Modell genutzt werden.

```
1 class GNN(torch.nn.Module):
2     def __init__(self, hidden_channels, dropout=0.2):
3         super().__init__()
4         self.dropout = dropout
5         self.conv1 = SAGEConv(hidden_channels, hidden_channels)
6         self.conv2 = SAGEConv(hidden_channels, hidden_channels)
7         self.conv3 = SAGEConv(hidden_channels, hidden_channels)
8     def forward(self, x, edge_index):
9         x = F.elu(self.conv1(x, edge_index))
10        x = F.dropout(x, p=self.dropout, training=self.training)
11        x = F.elu(self.conv2(x, edge_index))
12        x = F.dropout(x, p=self.dropout, training=self.training)
13        x = F.elu(self.conv3(x, edge_index))
14        x = F.dropout(x, p=self.dropout, training=self.training)
15    return x
```

Quellcode 3.1: Code der GNN Architektur

Die **GNN-Klasse** (Zeile 1) leitet sich von der Basisklasse für alle neuronalen Netzwerkmodule in PyTorch, `torch.nn.Module`, ab. Die Initialisierungsmethode `init` (Zeile 2) konfiguriert die Klasse, indem sie die Anzahl der ausgehenden Kanten jedes Knotens im neuronalen Netzwerk auf 64 festlegt (`hidden_channels`) und den Dropout-Wert auf 0.2 setzt, eine Technik zur Vorbeugung von Overfitting. Diese Werte wurden durch experimentelle Optimierung bestimmt.

Die Methode `forward` (Zeile 7) definiert den forward-pass des Netzwerks. Der Input-Graph durchläuft alle drei `SAGEConv`-Schichten (Zeilen 8, 10 und 12) und wird nach jedem Durchlauf durch eine Exponential Linear Unit (ELU)-Aktivierungsfunktion transformiert.

Die ELU[5] ist eine spezielle Aktivierungsfunktion, die in neuronalen Netzwerken Anwendung findet. Sie unterscheidet sich von anderen Aktivierungsfunktionen, wie der Rectified Linear Unit (ReLU), durch ihre Charakteristik für negative Eingangswerte. Anstelle von null nimmt die ELU für negative Eingangswerte Werte kleiner als null an und strebt asymptotisch den Wert -1 an. Bei positiven Eingangswerten hingegen verhält sich die ELU identisch zur ReLU. Die ELU kann somit das ‘‘Dying ReLU’’-Problem vermeiden, bei dem Neuronen dauerhaft inaktiv werden können. Zudem fördert die ELU die Beschleunigung des Trainings von neuronalen Netzen und verbessert die Konvergenz.

Nach jeder Aktivierung erfolgt die Anwendung einer Dropout-Regulierung auf die Ausgabe (Zeilen 9, 11 und 13).

Die Dropout-Regulierung[27] ist eine Methode zur Vermeidung von Overfitting (Überanpassung) bei Modellen. Bei großen neuronalen Netzen, die auf verhältnismäßig kleinen Datensätzen trainiert werden - wie im hier diskutierten Fall - kann das Risiko

von Overfitting bestehen. Dies äußert sich dadurch, dass das Modell Schwierigkeiten hat, auf neue Daten zu generalisieren, und somit eine suboptimale Performance aufweist.

Ein Ansatz zur Bekämpfung dieses Problems besteht darin, mehrere Modelle auf demselben Datensatz zu trainieren und anschließend den Durchschnitt der Vorhersagen zu bilden. Allerdings ist dies in der Praxis oft nicht praktikabel, da es ein aufwändiges Trainieren, Optimieren und Speichern mehrerer Modelle erfordert.

Als Alternative zur Umsetzung dieses Konzeptes wird die Dropout-Methode eingesetzt. Während des Trainings werden zufällig ausgewählte Ausgaben (Outputs) einer Schicht ignoriert, auch als „dropping“ bezeichnet (siehe Abbildung 3.2). Dies simuliert effektiv die Existenz von mehreren unterschiedlichen Modellen und führt zu einer Vielfalt in der Repräsentation von Daten, wodurch das Modell widerstandsfähiger gegen Fehler und Verzerrungen wird.

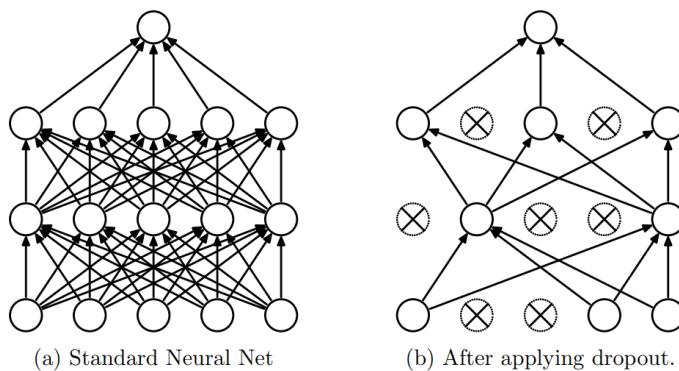


Abbildung 3.2: **Dropout in neuronalen Netzwerken:** Links: Ein Standard neuronales Netz mit 2 hidden layer. Rechts: Ein Beispiel für ein ausgedünntes Netz, das durch Anwendung von Dropout auf das Netzwerk links erzeugt wurde. Gekreuzte Einheiten wurden entfernt. (Quelle: [27])

Die Modellarchitektur in dieser Arbeit umfasst mehr als nur eine einfache GNN Struktur. Es ist entscheidend, den Eingangsgraphen korrekt zu definieren, der durch einen Knotenmerkmalsvektor  $x$  und eine Kantenindexmatrix bestimmt wird, wie in der *forward*-Funktion des Modells (Quellcode 3.3) gezeigt.

Jeder Knoten in diesem Modell ist durch einen Merkmalsvektor repräsentiert, der sowohl attributbasierte Informationen als auch eine ID-basierte Embedding enthält. Dies wird durch eine Kombination aus einer linearen Transformation und einem Embedding erreicht, wie im folgenden Codeausschnitt (Quellcode 3.2) gezeigt:

```

1 self.lin_sub = torch.nn.Linear(data['substation'].x.size(1),
2                                hidden_channels)
3 self.emb_sub = torch.nn.Embedding(data["substation"].num_nodes,
4                                   hidden_channels)

```

Quellcode 3.2: Vektorkonstruktion für „substation“-Knoten

Die lineare Schicht `self.lin_sub` transformiert die Knoteneigenschaften (welche zu diesem Zeitpunkt als One-Hot-Vektor vorliegen), indem sie eine lineare Transformation auf den Eingangsvektor anwendet, um die Dimension des Vektors auf `hidden_channels` zu reduzieren. Parallel dazu transformiert die Einbettungsschicht `self.emb_sub` die Knoten-IDs in einen eingebetteten Vektorraum der Dimension `hidden_channels` um aus diesen möglichen Informationen zu extrahieren, welche dem Modell helfen könnten. Die Ausgabe dieser beiden Schichten wird addiert, um den endgültigen Merkmalsvektor für die „Schaltanlage“-Knoten zu erhalten.

Es ist jedoch zu beachten, dass in diesem Modell für die „Felder“-Entitäten auf die Verwendung von Attributvektoren verzichtet wird, da deren Attribute keine vorteilhaften Informationen für das Training liefern.

In der Anwendung des GNN auf die gegebenen Daten besteht eine Besonderheit: Die Kantenindexmatrix darf nicht alle Kantenindizes für alle Kantenarten enthalten. Diese Praxis ist für andere Modelle häufig [13], da Kanten in der Regel für das Message Passing verwendet werden. Allerdings kann dies hier zu einem sogenannten Clever-Hans-Phänomen führen.

Das Clever-Hans-Phänomen[21] bezieht sich in diesem Kontext auf eine Situation, in der ein Modell scheinbar korrekte Vorhersagen trifft, jedoch aus den falschen Gründen. Im vorliegenden Fall würde das Modell durch das Message Passing indirekt auf der Grundlage der `directlyConnectedTo`-Verbindungen trainieren, die eigentlich das Ziel der Vorhersage darstellen. Dieses unerwünschte Verhalten war während des ursprünglichen Trainingsprozesses nicht offensichtlich und wurde erst durch umfangreiche Analysen aufgedeckt.

Ein klares Indiz für dieses Phänomen war, dass das Modell keine Vorhersagen abgab, wenn ein Baum ohne die `directlyConnectedTo`-Verbindungen eingegeben wurde. Um dieses Problem zu beheben, werden die `directlyConnectedTo` - Verbindungen vor der Eingabe entfernt und stattdessen ein Platzhalter eingefügt (siehe Quellcode 3.3 Zeile 9-12). Trotz dieser Änderung bleibt das Message Passing in den kleinen Input-Bäumen funktionsfähig, da die GIS-Module weiterhin effizient über das Feld kommunizieren können.

Nachdem der Knotenmerkmalsvektor und die Kantenindexmatrix definiert wurden, werden diese an das GNN-Modell übergeben. Zu diesem Zeitpunkt muss das GNN-Modell zu einem heterogenen Modell konvertiert werden, wie der folgende Codeausschnitt verdeutlicht:

```
1 self.gnn = to_hetero(self.gnn, metadata=data.metadata())
```

In der `forward`-Funktion des Modells (siehe Quellcode 3.3) wird der Knotenmerkmalsvektor für jede Entität konstruiert (Zeile 2-8) und zusammen mit der neu konstruierten Kantenindexmatrix (Zeile 9-12) an das GNN gesendet (Zeile 13). Die Ausgabe der `forward`-Funktion ist ein Tensor, der die generierten Knoten-Repräsentationen enthält (Zeile 14).

```

1 def forward(self, tree) -> Tensor:
2     x_dict = {
3         "substation": self.lin_sub(tree['substation'].x)
4             + self.emb_sub(tree["substation"].node_id),
5         "bay": self.emb_bay(tree["bay"].node_id),
6         "module": self.lin_module(tree['module'].x)
7             + self.emb_module(tree["module"].node_id)
8     }
9     new_tree = tree.clone()
10    del new_tree['directlyConnectedTo'].edge_index
11    new_tree['directlyConnectedTo'].edge_index
12        = torch.tensor([[[],[]], dtype=torch.long)
13    x = self.gnn(x_dict, new_tree.edge_index_dict)
14    return x

```

Quellcode 3.3: Die forward-Funktion des Modells

Im nächsten Kapitel liegt der Fokus auf dem Training des neuronalen Netzwerks.

## 3.4 Training des neuronalen Netzwerks

Das Training eines neuronalen Netzwerks ist ein iterativer Prozess, bei dem die Gewichtungen und Verzerrungen der Neuronen im Netzwerk so angepasst werden, dass die Leistung des Modells bei der Ausführung spezifischer Aufgaben optimiert wird. Im Kontext dieser Arbeit liegt der Fokus auf einem Modell, das darauf ausgerichtet ist, auf der Grundlage eines vorgegebenen Baums die Existenz einer Verbindung zwischen zwei spezifischen GIS-Modulen zu bestimmen.

Um dieses Ziel zu erreichen, wird ein Paradigma des überwachten Lernens verwendet. Dabei werden dem Modell Trainingsbeispiele präsentiert, bei denen sowohl die Eingaben (Eigenschaften und Verbindungen der Module) als auch die gewünschte Ausgabe (das Vorhandensein oder Fehlen einer Verbindung zwischen den Modulen) bekannt sind. Durch wiederholtes Durchlaufen dieses Trainingsprozesses entwickelt das Modell die Fähigkeit, korrekte Vorhersagen über neue, bisher ungesehene Daten zu treffen.

### 3.4.1 Die Klassifikationsfunktion

Zunächst wird die Klassifikationsfunktion (hier genannt Classifier) definiert. Hierbei handelt sich um eine Funktion, die auf dem Hadamard-Produkt<sup>8</sup> zweier Eingabevektoren basiert. Dies ist im Wesentlichen eine Methode, um Interaktionen zwischen den beiden Knoten-Vektorrepräsentationen zu modellieren. Der resultierende Vektor hat

<sup>8</sup>Einfache elementweise Multiplikation

die gleiche Dimension wie die beiden Eingabevektoren und enthält das Produkt der entsprechenden Elemente.

Die Definition der Klassifikator-Klasse ist wie folgt:

```

1 class Classifier(torch.nn.Module):
2     def hadamard(self, x_module: Tensor, edge_label_index):
3         edge_feat_module1 = x_module[edge_label_index[0]]
4         edge_feat_module2 = x_module[edge_label_index[1]]
5         return (edge_feat_module1*edge_feat_module2).sum(dim=-1)

```

Die *hadamard* Methode nimmt zwei Eingabevektoren: *x\_module*, das alle Module im Input-Baum als Embedding darstellt und *edge\_label\_index*, das die Indizes der Module enthält, zwischen denen eine Verbindung besteht. Es berechnet für jedes Modulpaar das Hadamard-Produkt und summiert anschließend die Ergebnisse entlang der letzten Dimension.

Das Ergebnis ist ein Vektor der Länge  $\text{len}(\text{edge\_label\_index})$ , welcher für jeden Eintrag an der Stelle  $i$  einen Wert enthält, der die Vorhersage für das Vorhandensein einer Verbindung zwischen den zwei Modulen (definiert an der Stelle  $\text{edge\_label\_index}[:, :][i]$ ) darstellt. Die Abbildung 3.3 illustriert diesen Prozess.

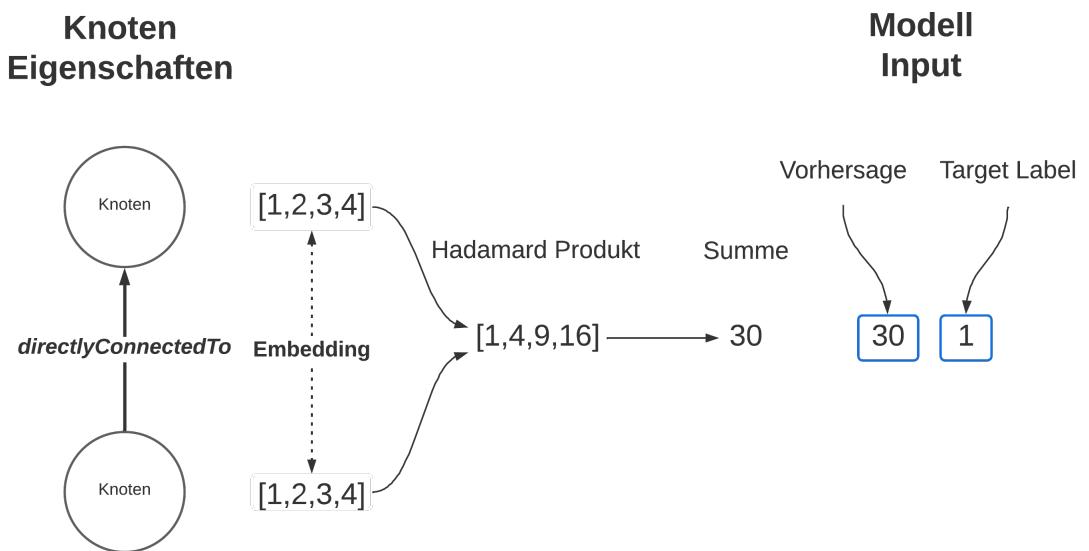


Abbildung 3.3: **Classifier:** Der Vorgang der Verbindungsverhersage basiert auf dem summierten Hadamard-Produkt. Zwei Knoten-Embeddings, die durch eine Relation verknüpft sind, werden zur Erzeugung des Hadamard-Produkts zusammengeführt. Die Summe dieses Produkts liefert die Verhersage der Relation. Durch den Vergleich dieser Verhersage mit dem dazugehörigen Label wird das Modell trainiert. (Quelle: In Anlehnung an [3])

### 3.4.2 Erstellung von positiven und negativen Beispielen

Für das Training des Modells werden sowohl positive als auch negative Beispiele benötigt. Positive Beispiele sind Paare von GIS-Modulen, zwischen denen eine reale Verbindung innerhalb des Baumes besteht, während negative Beispiele Paare von Modulen darstellen, zwischen denen keine Verbindung besteht. Diese Beispiele werden mit der Funktion `generate_positive_negative_examples` generiert.

```
1 def generate_positive_negative_examples(edge_index, num_nodes):
2     pos_edge_index = edge_index
3     neg_edge_index = negative_sampling(edge_index, num_nodes)
4     return pos_edge_index, neg_edge_index
```

Die Funktion `negative_sampling` wird verwendet, um die Kantenindizes für die negativen Beispiele zu erzeugen. Diese Funktion erzeugt zufällige Kantenindizes, die nicht in den Eingabekantenindizes vorhanden sind. Dabei ist die Anzahl der positiven und negativen Kanten ausgeglichen.

### 3.4.3 Lossfunktion und Optimierung des Modells

Nach der Berechnung der Vorhersagen für die positiven und negativen Beispiele wird der Loss<sup>9</sup> für diese Vorhersagen berechnet. Der Loss ist ein Maß dafür, wie gut die Vorhersagen des Modells mit den tatsächlichen Labels übereinstimmen.

In diesem Kontext wird eine Lossfunktion benötigt, welche für binäre Klassifikationsprobleme gedacht ist und die Labels sich in der Menge {0,1} befinden. Diese Beschreibung passt zu der binary cross entropy (binäre Kreuzentropie), welche daher als Lossfunktion verwendet wird. Diese Funktion berechnet den Loss für Klassifikationsprobleme, bei denen jede Beobachtung zu genau einer Klasse gehört.

Für die positiven Beispiele wird der Verlust bestimmt, indem die binäre Kreuzentropie zwischen den Vorhersagen des Modells und den tatsächlichen Labels (die alle auf 1 gesetzt sind) berechnet werden. Für die negativen Beispiele wird der Verlust auf die gleiche Weise berechnet, aber die tatsächlichen Labels sind alle auf 0 gesetzt. Der anschließende Loss, auf dem die Optimierung des Modells basiert, wird durch die Summe der beiden Losses erschlossen.

```
1 pos_labels = torch.ones(pos_preds.shape[0]).to(device)
2 neg_labels = torch.zeros(neg_preds.shape[0]).to(device)
3 loss = F.binary_cross_entropy_with_logits(pos_preds, pos_labels)
4     + F.binary_cross_entropy_with_logits(neg_preds, neg_labels)
```

<sup>9</sup>deutsch Verlust

Nach der Berechnung des Verlustes wird ein Optimierer (in diesem Fall der Adam-Optimierer) verwendet um die Gewichte des Modells anzupassen und den Verlust zu minimieren.

#### 3.4.4 Iterationen zur Modellverbesserung und dessen Validierung

Der gesamte oben beschriebene Prozess wird als „Epoche“ bezeichnet und wird 50-mal durchlaufen, mit dem Ziel, die Leistungsfähigkeit des Modells schrittweise zu verbessern. Bei jeder Iteration wird die Performance des Modells gemessen (siehe Abb. 4.1). Dafür werden zwei Messwerte benutzt: Die Accuracy (Genauigkeit) und der ROC-AUC-Score. Beides sind gängige Metriken zur Messung der Leistung von Klassifikationsmodellen, doch sie messen unterschiedliche Aspekte der Modellleistung.

##### 3.4.4.1 Genauigkeit (Accuracy)

Die Accuracy, als eine grundlegende Metrik, ist das Verhältnis zwischen der Anzahl korrekter Vorhersagen und der Gesamtzahl der Vorhersagen:

$$\text{Accuracy} = \frac{\text{Anzahl der korrekten Vorhersagen}}{\text{Gesamtzahl der Vorhersagen}} \quad (3.2)$$

Für binäre Klassifikationsprobleme, wie sie in dieser Arbeit behandelt werden, ist die Accuracy ein geeigneter Indikator, solange die Klassen ausgewogen sind.

Bei unausgewogenen Klassen (z.B. viel mehr positive Vohersagen als negative) könnte ein Modell, das einfach immer die Mehrheitsklasse vorhersagt, eine hohe Accuracy erzielen, obwohl es eigentlich schlecht arbeitet. Daher wird in solchen Fällen auf andere Metriken zurückgegriffen, wie beispielsweise auf den ROC-AUC-Score.

##### 3.4.4.2 ROC-AUC-Score

Der ROC-AUC-Score ist eine Metrik, die besonders bei unausgewogenen Klassen vorteilhaft ist. Die Receiver Operating Characteristic (ROC)-Kurve (siehe Abb. 4.2) ist ein grafisches Werkzeug zur Darstellung der Leistung eines Klassifikationsmodells über alle Klassifikationsschwellen (Threshold) hinweg. Sie stellt die True Positive Rate (TPR) gegen die False Positive Rate (FPR) dar. Die Area Under the Curve (AUC) fasst die ROC-Kurve in einen einzigen Wert zusammen, indem sie die Fläche unter der ROC-Kurve berechnet. Der AUC-Wert liegt zwischen 0 und 1, wobei ein Wert von 1 eine perfekte Klassifikation darstellt und ein Wert von 0,5 einem zufälligen Raten entspricht.

### 3.4.4.3 Precision und Recall

Zusätzlich zu den bisher diskutierten Leistungsmetriken erfolgt eine ergänzende Beurteilung des Modells mithilfe einer alternativen Genauigkeitsmessung. Wie im Kapitel 3.5 erläutert, nimmt das Modell im praktischen Einsatz alle potenziellen Verbindungen als Input entgegen. Hierbei wird jede *directlyConnectedTo*-Verbindung innerhalb der jeweiligen Bäume einzeln analysiert und eine Vorhersage generiert. Precision<sup>10</sup> gibt an, welcher Anteil der vom Modell als positiv klassifizierten Samples tatsächlich positiv ist. Es beantwortet also die Frage wieviele der Vorhersagen richtig sind. Während Recall<sup>11</sup> den Anteil der tatsächlich positiven Samples angibt, die vom Modell auch als solche erkannt wurden und somit die Frage beantwortet wieviele der tatsächlichen Verbindungen vorhergesagt wurden.

Der bedeutende Unterschied zu herkömmlichen Genauigkeitsmetriken liegt in der Betonung der True Positives, die eine praktisch relevantere Leistungsbewertung ermöglichen. Zusätzlich ergibt sich bei dieser Methode eine abweichende Verteilung der positiven und negativen Verbindungen im Vergleich zu den bisherigen Messungen. Diese Thematik wird im Kapitel 4 näher betrachtet.

Diese Metriken stellen somit eine wertvolle Ergänzung dar und tragen zu einer umfassenderen Beurteilung der Modellleistung bei.

Diese alternative Metrik wird in Abbildung 3.4 veranschaulicht:

Insgesamt liefern sowohl die konventionelle Accuracy als auch der ROC-AUC-Score sowie die alternative Genauigkeitsberechnung wertvolle Informationen zur Beurteilung der Modellleistung. Dabei liefert keine einzelne Metrik ein vollständiges Bild der Modellleistung, weshalb der Vergleich mehrerer Metriken essentiell ist, um ein ausgewogenes Verständnis der Modellleistung zu erlangen.

Die Modellvalidierung wird mittels eines eigenständigen Datensatzes durchgeführt, der während der Trainingsphase für das Modell unbekannt bleibt. Die Modellprognosen werden mit den tatsächlichen Bezeichnungen dieses Validierungsdatensatzes verglichen, um eine unvoreingenommene Einschätzung der Modellleistung zu ermöglichen.

Das primäre Ziel dieses iterativen Optimierungs- und Validierungsprozesses liegt in der Entwicklung eines Modells, das hohe Genauigkeit in der Vorhersage der Präsenz oder Absenz von Verbindungen zwischen GIS-Modulen aufweist. Damit wird sichergestellt, dass das Modell nicht nur auf den Trainingsdaten gute Leistung zeigt, sondern auch in der Lage ist, robuste und genaue Vorhersagen für neue, unbekannte Daten zu liefern. Die Verbesserungen, die durch die wiederholten Iterationen und Validierungen erzielt werden, tragen dazu bei, dass das Modell zuverlässig und widerstandsfähig gegenüber neuen, unerwarteten Daten bleibt.

<sup>10</sup>deutsch Präzision

<sup>11</sup>deutsch Abruf

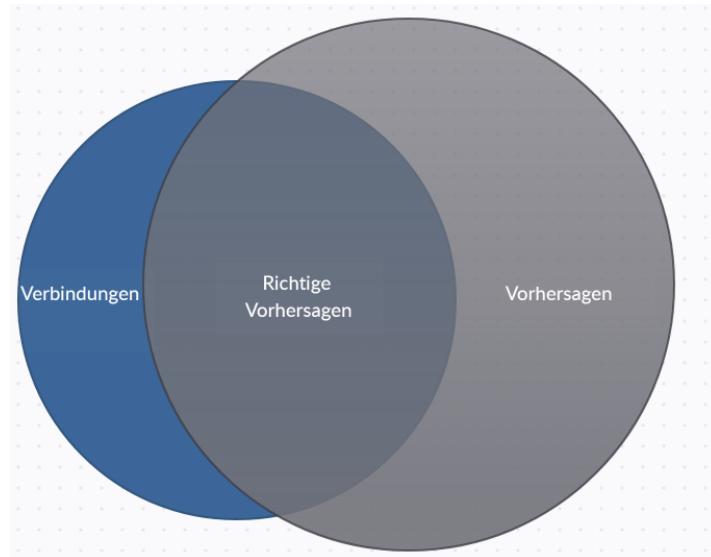


Abbildung 3.4: **Precision und Recall:** Visualisierung der Modellgenauigkeit basierend auf den einzelnen Bäumen in Form eines Venn-Diagramms. Der linke Kreis symbolisiert die Gesamtheit aller tatsächlichen Verbindungen, während der rechte Kreis die Gesamtheit aller vorhergesagten Verbindungen darstellt. Der Schnittbereich der beiden Kreise repräsentiert die korrekt vorhergesagten Verbindungen. Der Recall wird als das Verhältnis dieses Schnittbereichs zur Gesamtheit der vorhergesagten Verbindungen definiert, während die Precision das Verhältnis des Schnittbereichs zur Gesamtheit der tatsächlichen Verbindungen darstellt. (Quelle: eigene Darstellung)

Somit wurde ein weiterer Schritt des Use Case (siehe Aktivitäts-Diagramm 1.1) implementiert. Im nächsten Kapitel wird der letzte Schritt besprochen.

## 3.5 Einfügen in den Graphen

In diesem Kapitel wird der letzte Schritt des Use Case, der unter dieser Arbeit entstanden Softwarespezifikation, erläutert und Implementiert. Dabei wird zu nächst auf Basis des Input-Baums, welcher keine *directlyConnectedTo*-Verbindungen enthält, alle möglichen Verbindungen konstruiert (Quellcode 3.4). Das heißt von jedem GIS-Modul (Zeile 3) gehen Verbindungen zu allen anderen GIS-Modulen (Zeile 5) aus.

```

1 def generate_edges(tree):
2     module_indices = tree['module'].node_id.tolist()
3     edge_0 = [[module]*len(module_indices)-1]
4             for module in module_indices]
5     edge_1 = [module_indices[:i] + module_indices[i+1:]]
6             for i in range(len(module_indices))]
7     edge_index = torch.stack((torch.tensor(edge_0).flatten(),
8                               torch.tensor(edge_1).flatten()))
9
9     return edge_index

```

Quellcode 3.4: Erstellung aller möglichen Kanten

Anschließend kann man durch das im letzten Kapitel trainierte Modell die Embeddings der einzelnen GIS-Module erzeugen. Mit den Kanten und Embeddings kann man nun jede Verbindung prüfen. Zum Schluss bekommt man die vorhergesagten Verbindungen. Diese werden in den Graphen wieder eingefügt, um somit eine PST-Vorlage zu erzeugen auf dem ein Ingenieur arbeiten und anschließend korrigieren kann.

Nachdem nun eine detaillierte Beschreibung der Implementierung und der zugrunde liegenden Methodik vorliegt, folgt im nächsten Kapitel die Definition, Durchführung und Auswertung von Experimenten, die dazu dienen, die Leistungsfähigkeit des Modells weiter zu optimieren.

## 4 Experimente und Auswertung

Dieses Kapitel ist der detaillierten Auswertung des entwickelten Modells gewidmet und beschreibt zudem die zusätzlich durchgeführten Experimente, welche zum weiteren Verständnis und zur Optimierung des Modells beitragen sollten.

Die Abbildung 4.1 gibt einen ersten Eindruck vom Training des Modells. Es zeigt den stetig abnehmenden Loss während der Trainingsphase, was auf eine erfolgreiche Anpassung und Verbesserung des Modells hindeutet. Nach Durchführung von 50 Iterationen, den sogenannten Epochen, wurde ein Modell mit zufriedenstellender Präzision und Verlässlichkeit erzielt. Die erreichte Accuracy des Modells beträgt 79% (siehe Abb. 4.1). Diese Genauigkeitsberechnung berücksichtigt alle Vorhersagen (sowohl positive als auch negative) aus allen Input-Bäumen im Validationsdatensatz.

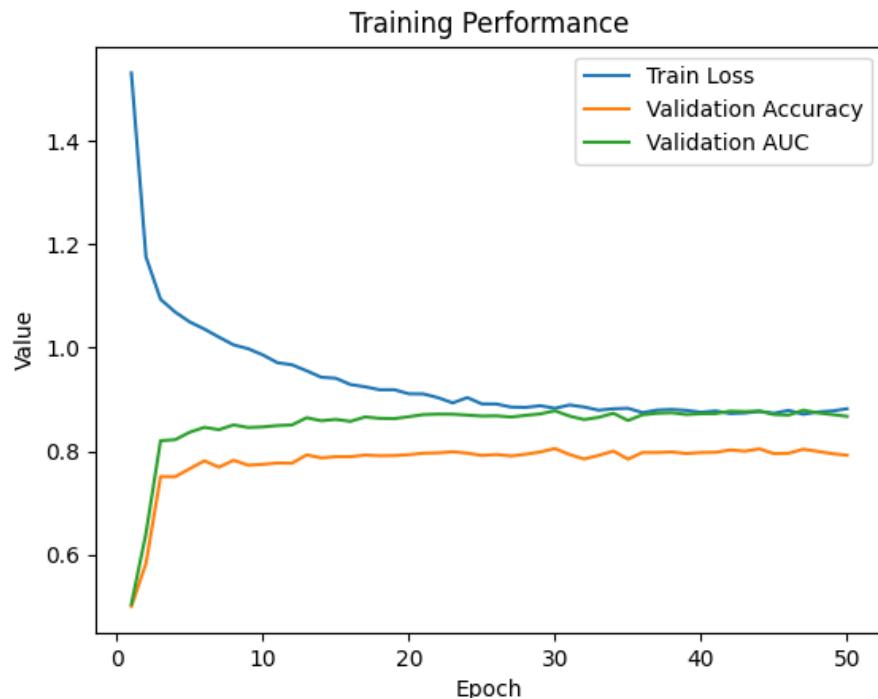


Abbildung 4.1: **Trainingsverlauf:** Diese Abbildung zeigt den Verlauf von Loss, Accuracy und dem ROC-AUC-Score des Modells für jede Trainingsepoke. (Quelle: eigene Darstellung)

Die Abbildung 4.2 präsentiert den ROC-AUC-Score des Modells, der mit einem Wert von 0.88 ein hervorragendes Ergebnis zeigt.

Unter Berücksichtigung der praxisrelevanten Metriken zeichnet das Modell folgendes Bild: Es konnte 85% der tatsächlich bestehenden Verbindungen korrekt vorhersagen (Recall), jedoch entspricht diese Genauigkeit nur 38% der Gesamtheit aller Vorhersagen (Präzision).

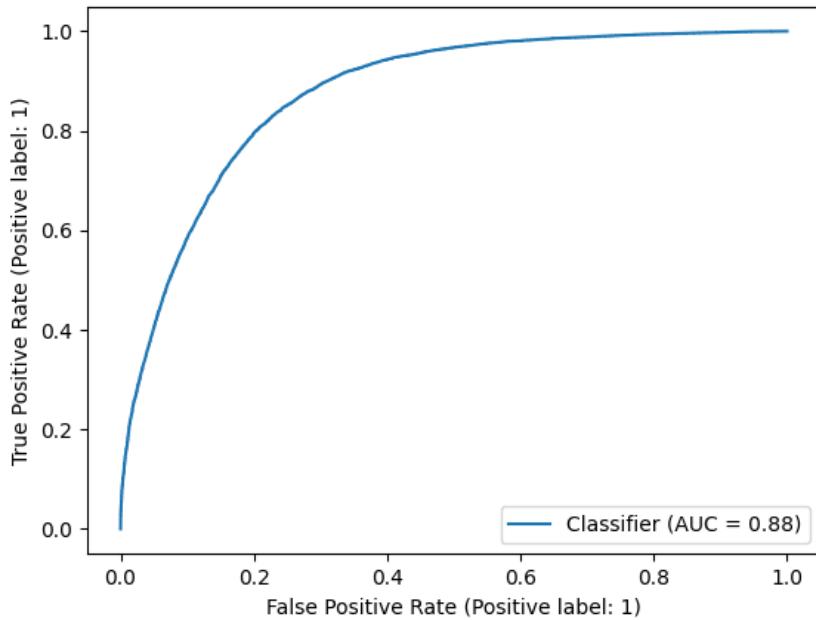


Abbildung 4.2: **ROC-AUC-Score:** Die blaue Linie ist die ROC-Kurve. Die Fläche unter dieser Kurve nennt sich AUC und ist gleich 0.88. (Quelle: eigene Darstellung)

Wie aus der Abbildung 4.1 ersichtlich, zeigt das Modell nach der 30. Epoche keine signifikanten Verbesserungen mehr. Daher wird in den folgenden Experimenten die Anzahl der Trainingsepochen auf 30 reduziert.

## 4.1 Negative Sampling

### 4.1.1 Experiment

Die Berücksichtigung der Realitätssituation im Kontext der Modelltrainingsstrategie ist ein wesentlicher Aspekt. Im praktischen Einsatz werden alle möglichen *directly-ConnectedTo*-Verbindungen eines Baums konstruiert und vom Modell klassifiziert. In der bisherigen Trainingsphase wurde diese Situation nicht berücksichtigt. Es wurde ein einfacheres negatives Sampling-Verfahren angewandt, um ein ausgeglichenes Verhältnis zwischen negativen und positiven Samples zu gewährleisten. Diese Methode spiegelt jedoch nicht die Verteilung in einer realen Anwendung wider: Das Modell ist zwar in der Lage, einen Großteil der tatsächlichen Verbindungen korrekt zu klassifizieren, tendiert jedoch dazu, eine erhebliche Anzahl nicht existierender Verbindungen fälschlicherweise als vorhanden einzustufen. Diese Beobachtung führt zu weiteren Überlegungen und Experimenten, die in den folgenden Abschnitten beschrieben werden.

Um diesem Problem zu begegnen, wird ein alternatives negatives Sampling-Verfahren eingeführt, das der Praxisverteilung ähnlicher ist. Die tatsächlichen Verbindungen dienen weiterhin als positive Samples, während alle anderen möglichen Verbindungen als negative Samples gelten. Das resultierende Ungleichgewicht, das durch die höhere Anzahl an negativen Samples entsteht, führt zu einem Modellbias, der in diesem Fall von Vorteil ist und die realen Bedingungen besser widerspiegelt. Wichtig zu beachten ist hierbei auch die Berechnung des Loss. Positive und negative Vorhersagen werden getrennt betrachtet<sup>1</sup>. Dies führt dazu, dass das Modell nicht durch das Ungleichgewicht an Performance verliert.

### 4.1.2 Auswertung

Die allgemeine Genauigkeit (Accuracy) und der AUC-Score sind, wie in Abbildung 4.3 dargestellt, gesunken. Dies lässt zunächst vermuten, dass die Leistung des Modells abgenommen hat. Doch es ist wichtig zu beachten, dass diese Standardmetriken bei starkem Klassenungleichgewicht nicht immer ein vollständiges Bild der Modellleistung liefern.

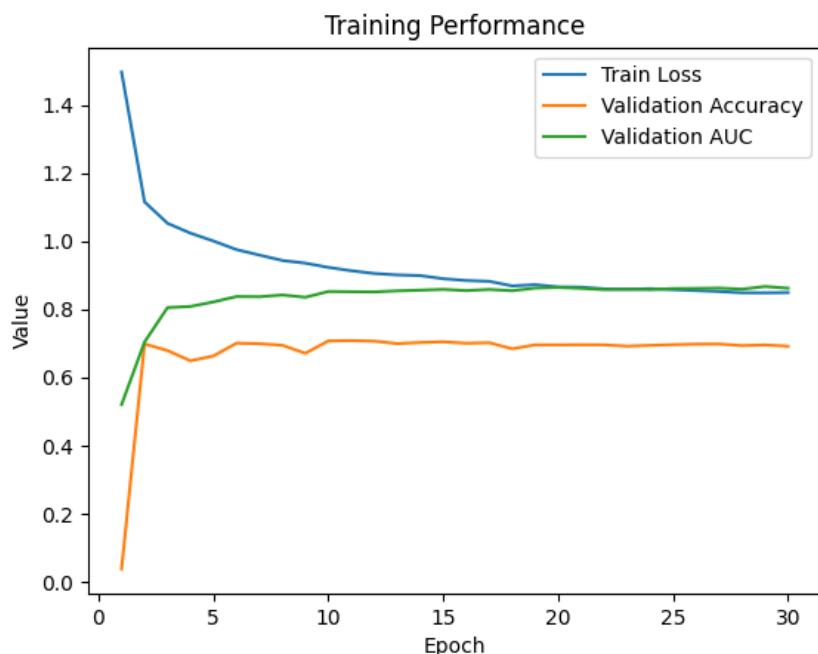


Abbildung 4.3: **Training des alternativen negative Sampling Modells:** Die Accuracy zeigt nach der ersten Epoche keinen weiteren Anstieg, der AUC-Score steigt jedoch stetig an und der Loss im Training sinkt kontinuierlich. (Quelle: eigene Darstellung)

---

<sup>1</sup>siehe Kapitel 3.4.3

Für diesen spezifischen Anwendungsfall bieten die Metriken Precision und Recall, die in einer alternativen Genauigkeitsmessung berücksichtigt werden, einen aussagekräftigeren Einblick in die Leistung des Modells. Diese Bewertungsmethodik ergibt, dass das Modell für 39% der getroffenen Vorhersagen korrekt liegt (Precision) und 87% der tatsächlich existierenden Verbindungen korrekt erkannt hat (Recall). Diese Werte bedeuten, dass trotz der niedrigeren allgemeinen Genauigkeit und des geringeren AUC-Scores eine signifikante Verbesserung der Modellleistung im Vergleich zu den ursprünglichen Resultaten festgestellt wurde.

Wie in der Definition des Experimentes beschrieben, ist die Berechnung des Loss überaus wichtig. Ohne die getrennte Betrachtung erzielt das Modell einen viel kleineren Loss und eine viel höhere Accuracy (siehe Abb. 4.4), jedoch sind Precision und Recall sehr viel schlechter. Und diese Metriken sind für diesen Anwendungsbereich wichtiger.

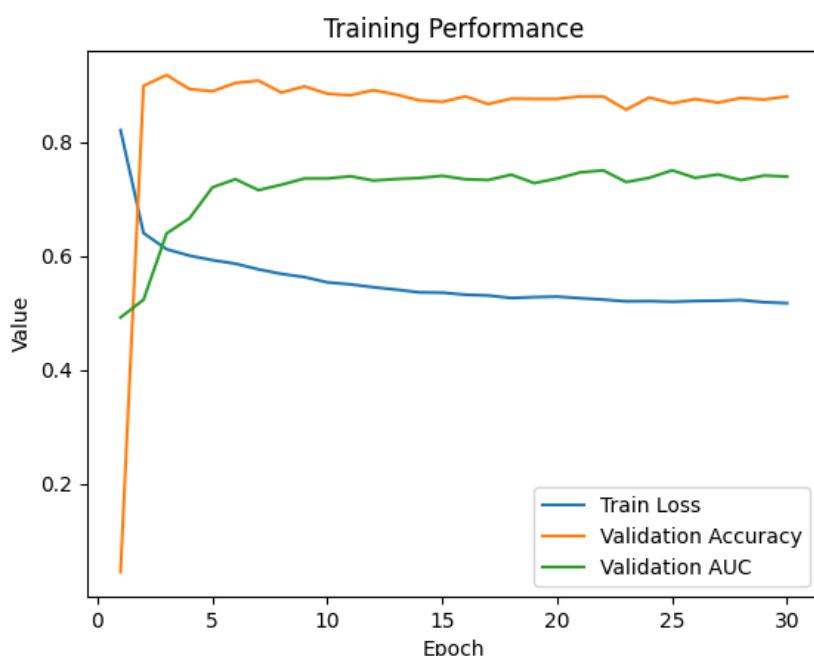


Abbildung 4.4: **Training mit zusammengelegtem Loss:** Die Accuracy ist sehr hoch und der Loss ist sehr niedrig. Anzeichen für ein sehr Performantes Modell. (Quelle: eigene Darstellung)

Zusammengefasst zeigt dieses Experiment, dass eine Trainingsstrategie, die die tatsächlichen Bedingungen besser widerspiegelt, zu einer besseren Modellleistung in der Praxis führen kann, auch wenn dies zu niedrigeren Werten bei Standard-Leistungsmaßen führt. Das in diesem Experiment entstandene Modell wird für das nächste Experiment weiter verwendet.

## 4.2 Anpassung der Lossfunktion

Dieses Experiment konzentriert sich darauf, wie die Modifizierung der Lossfunktion zur Optimierung des Modells beitragen kann, insbesondere bei unausgewogenen Datensätzen.

### 4.2.1 Experiment

Bislang wurde die binäre Kreuzentropie (binary cross entropy) als Lossfunktion genutzt. Obwohl diese Methode bei ausgewogenen Datensätzen wirksam ist, kann sie bei Klassenungleichgewicht zu einer Bevorzugung der dominierenden Klasse führen. Dies kann sich negativ auf die Erkennungsleistung der weniger häufigen Klasse auswirken.

Um dieser Tendenz entgegenzuwirken, wurde in diesem Experiment eine gewichtete Lossfunktion eingeführt. Jeder Klasse wurde ein spezifisches Gewicht zugewiesen, das deren Bedeutung während des Trainingsprozesses reflektiert. Eine höhere Gewichtung erhöht die Kosten für falsche Vorhersagen der entsprechenden Klasse und ermutigt das Modell, sich stärker auf die korrekte Klassifikation dieser Klasse zu konzentrieren.

Für die Gewichtung wurden mehrere Schemata getestet, um die optimale Konfiguration zu finden. Das Modell zeigte die beste Leistung mit einer Gewichtung von 5 für die unterrepräsentierte Klasse. Diese Anpassung führte, wie erwartet zu einer Veränderung im Trainingsverhalten des Modells, wie in Abbildung 4.5 zu sehen ist.

### 4.2.2 Auswertung

Nach Anpassung des Schwellenwerts für die Klassifikation erreicht das Modell eine Precision von 40% und einen Recall von 89%. Diese Werte deuten auf eine verbesserte Modellleistung hin, insbesondere hinsichtlich der korrekten Klassifikation der weniger häufigen Klasse.

Die Anwendung einer gewichteten Lossfunktion hat sich als effektive Methode zur Verbesserung der Modellleistung erwiesen, insbesondere im Hinblick auf den Umgang mit einem unausgewogenen Datensatz. Durch die gezielte Gewichtung der Klassen während des Trainingsprozesses konnte das Modell effektiver dazu angeleitet werden, die korrekte Klassifikation der selteneren Klasse zu verbessern. Dieser Ansatz hat das Potenzial, das Modell robuster und genauer in der praktischen Anwendung zu gestalten.

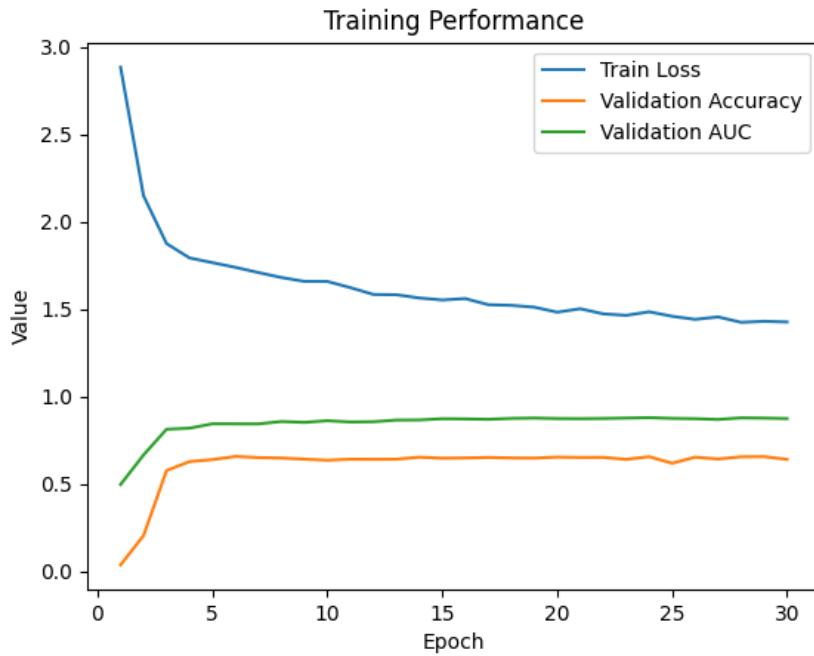


Abbildung 4.5: **Training mit gewichteter Lossfunktion:** Accuracy und AUC-Score ähneln den Ergebnissen vorheriger Modelle, jedoch ist der Loss aufgrund der Gewichtung höher. (Quelle: eigene Darstellung)

## 4.3 Anpassung der Modellarchitektur

Dieses Experiment konzentriert sich auf die Feinabstimmung der Modellarchitektur, um sie stärker auf den spezifischen Anwendungsfall zuzuschneiden.

### 4.3.1 Experiment

Die ursprüngliche Modellarchitektur wurde auf Grundlage gängiger Modelle konzipiert. Empfehlungen aus dem GraphSAGE-Artikel [8], der die Grundlage für die Architektur dieser Arbeit bildet, führten zur Implementierung von drei SAGEConv-Schichten, wobei jede Schicht eine Hop-Anzahl von  $k = 3$  darstellt. Obwohl diese Hop-Anzahl für viele Wissensgraphen optimal sein mag, scheint sie für diesen Anwendungsfall nicht ideal zu sein.

Eine Betrachtung des Input-Baums, auf dem das Modell trainiert wird, zeigt, dass eine Hop-Anzahl von 2 für die GIS-Module ausreicht. In Abbildung 4.6 wird dieser Prozess veranschaulicht. Ein dritter Hop würde für diesen spezifischen Eingabe-Baum keinen zusätzlichen Nutzen bringen, da er schlichtweg zu klein ist. Daher wird in diesem Experiment die Anzahl der SAGEConv-Schichten auf zwei reduziert.

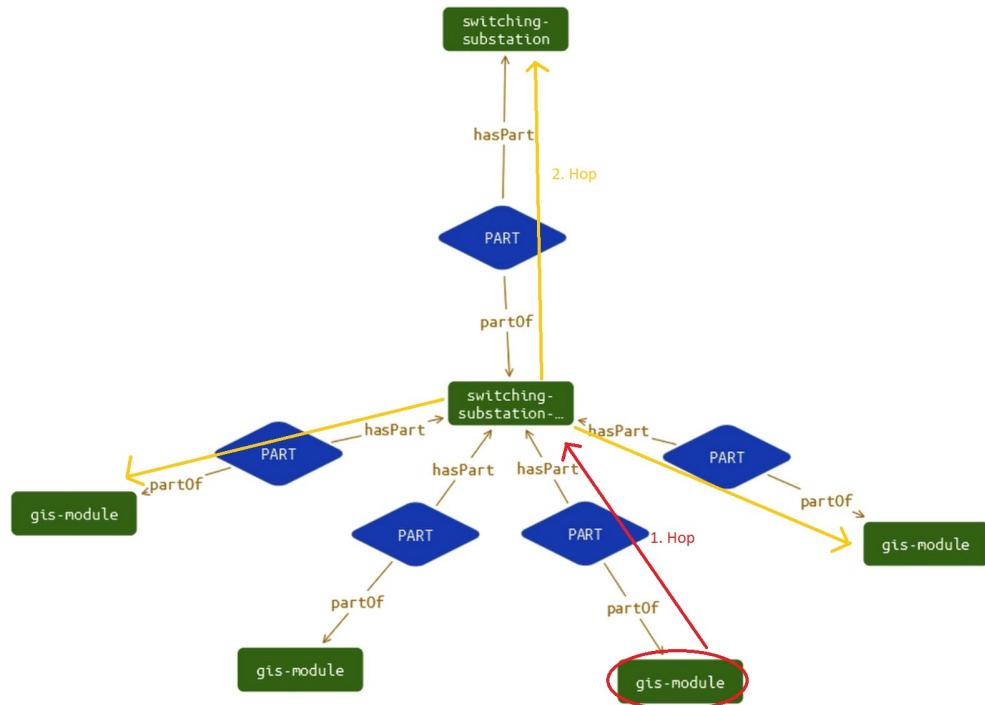


Abbildung 4.6: **Veranschaulichung der Hops im Input-Baum** Der Rote Kreis markiert den Knoten, dessen Embedding berechnet werden soll. Ausgehend von diesem GIS-Modul bleibt dem Modell nur das Feld für den ersten Hop. Von dem Feld aus kommt das Modell mit dem zweiten Hop an jeden anderen Knoten. Ein dritter Hop ist überflüssig. (Quelle: eigene Darstellung)

### 4.3.2 Auswertung

Wie aus Abbildung 4.7 ersichtlich ist, erzielt das Modell eine Genauigkeit (Accuracy) von 65%, was den bisher niedrigsten erreichten Wert darstellt. Im Gegensatz dazu erreicht der AUC-Score mit 0.90 den höchsten gemessenen Wert. Für die praktische Anwendung sind jedoch die Precision und der Recall entscheidend. Diese erreichen beachtliche 44% bzw. 91%.

Trotz einer scheinbar verringerten Genauigkeit hat sich das Modell mit der angepassten Architektur tatsächlich verbessert. Die Reduzierung der SAGEConv-Schichten hatte zwar eine niedrigere Genauigkeit zur Folge, führte jedoch zu Verbesserungen sowohl beim AUC-Score als auch bei den entscheidenden Kennzahlen Precision und Recall. Diese Ergebnisse deuten auf eine verbesserte Leistungsfähigkeit des Modells in seiner praktischen Anwendung hin. Zusammenfassend lässt sich sagen, dass alle durchgeführten Experimente und Anpassungen zu einer Verbesserung der Modellleistung beigetragen haben. Diese Verbesserungen verdeutlichen die Wichtigkeit der Anpassung der Modellparameter an die spezifischen Anforderungen des Anwendungsfalls und bestätigen die Wirksamkeit der verwendeten Methoden und Techniken.

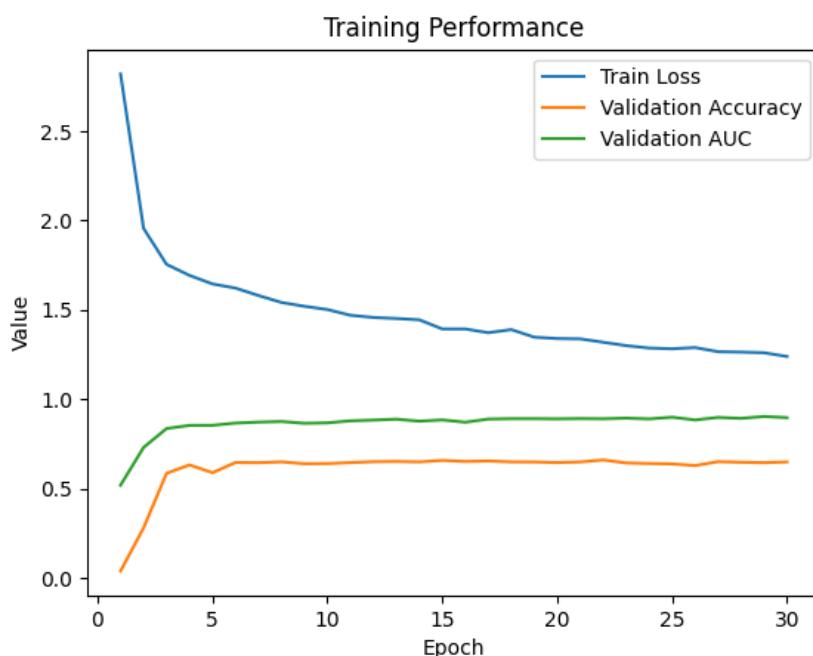


Abbildung 4.7: **Training des angepassten Modells:** Die Genauigkeit erreicht mit 65% den bisher niedrigsten Wert, während der AUC-Score mit 0.9 seinen Höchststand erreicht. Der stetig sinkende Loss deutet darauf hin, dass bei fortgesetztem Training möglicherweise ein noch besseres Modell erzielt werden könnte, jedoch wurde dies durch eine weiterführendes Training widerlegt. (Quelle: eigene Darstellung)

## 5 Fazit

In der vorliegenden Masterarbeit lag der Fokus auf der Entwicklung und Anwendung von Graph Neural Networks (GNNs) und Knowledge-Graph-Embedding-Methoden (KGEs) zur Vorhersage von Verbindungen in komplexen Produkt-System-Topologien. Das angestrebte Ziel bestand darin, den Konfigurationsprozess von Hochspannungsanlagen zu erleichtern.

### 5.1 Diskussion der Ergebnisse

Die Integration von KGE-Methoden und GNNs hat das Potenzial aufgezeigt, die Leistungsfähigkeit bei der Vorhersage von Verbindungen zu steigern. Trotz der Herausforderungen, wie dem Clever-Hans-Phänomen und der Notwendigkeit einer anwendungsspezifischen Modellanpassung, erzielte das Modell in den durchgeföhrten Experimenten zufriedenstellende Resultate.

Die Experimente demonstrierten die Fähigkeit des Modells, Kontextinformationen effizient zu nutzen, um Verbindungen zwischen Komponenten zu prognostizieren. Der Einsatz von Techniken wie gewichtetem Loss und alternativem Negative Sampling trugen zur Leistungssteigerung des Modells bei und passten es den spezifischen Anforderungen der Anwendung an. Mithilfe eines einstellbaren Schwellenwerts lässt sich das Modell und somit dessen Performance weiter an die spezifischen Anforderungen der Konfiguration anpassen.

Die Auswahl geeigneter Leistungskennzahlen zur Beurteilung der Modellleistung ist von zentraler Bedeutung und muss auf den spezifischen Anwendungsfall abgestimmt sein. Sie muss sowohl die Anforderungen des Anwendungsfalls als auch die Charakteristika des Modells berücksichtigen. In diesem Kontext haben sich die Metriken Precision und Recall als aussagekräftig erwiesen, da sie auch bei ungleicher Verteilung der Daten praxisrelevante Werte liefern. Die Accuracy kann aufgrund der großen Diskrepanz zwischen der Anzahl der positiven und negativen Samples irreführend sein.

Eine bemerkenswerte Erkenntnis aus den durchgeföhrten Experimenten war, dass sowohl die Precision als auch der Recall stiegen, obwohl die Accuracy sank. Diese Entwicklung kann auf den in späteren Experimenten eingeföhrten Bias zurückgeführt werden, der dazu führte, dass das Modell mehr positive Kanten identifizierte als zuvor. Dadurch wurden zwar auch viele negative Kanten fälschlicherweise als positive klassifiziert, was zu einem allgemeinen Rückgang der Accuracy führte, jedoch stiegen Precision und Recall an. Dies zeigt, dass trotz einer geringeren Gesamtgenauigkeit das Modell in der Lage war, mehr relevante Ergebnisse zu liefern, und damit die Vorhersageleistung verbessert wurde.

Das finale Modell konnte durchschnittlich 91% der Verbindungen korrekt vorhersagen. Obwohl 56% der Vorhersagen fälschlicherweise Verbindungen annahmen, reichte eine Präzision von 44%, um den Konfigurationsprozess für Ingenieure zu beschleunigen. Das Überprüfen bestehender Verbindungen ist einfacher als das Erstellen neuer. Daher wurden die Anforderungen dieser Arbeit erfüllt und eine solide Grundlage für zukünftige Forschungsarbeiten geschaffen. Dennoch gibt es Verbesserungspotenzial, das im folgenden Abschnitt besprochen wird.

## 5.2 Ausblick

Die Forschungsfelder der GNNs und KGEs sind noch verhältnismäßig jung, daher besteht ein deutlicher Bedarf an weiteren Untersuchungen. Es könnte insbesondere von Nutzen sein, eine spezifische Modellarchitektur für kleinere Abschnitte von Wissensgraphen zu konzipieren. Eine mögliche Strategie wäre die Kombination zweier Modelle: eines, das auf dem gesamten Wissensgraph trainiert wird, und eines, das spezifisch auf Input-Bäumen lernt. Solche Ansätze könnten zur weiteren Verbesserung der Modellleistung beitragen und die Anwendbarkeit auf eine größere Palette von Anwendungsfällen erhöhen.

Eine weitere potenzielle Verbesserung wäre die Implementierung von Regeln für Verbindungen, wie beispielsweise eine Beschränkung der Anzahl von *directlyConnectedTo*-Verbindungen, die ein GIS-Modul haben darf. Viele GIS-Module dürfen nur eine oder zwei Verbindungen haben, sehr wenige mehr als zwei. Die Einbindung solcher und weiterer Konfigurationsregeln in den Trainingsalgorithmus könnte die Modellleistung erheblich steigern.

Darüber hinaus kann der Anwendungsbereich sowie die damit einhergehenden Anforderungen an das Modell erweitert werden. Dabei wäre es möglich, nicht nur vorherzusagen, welche GIS-Module miteinander verbunden sind, sondern auch die spezifischen Details dieser Verbindungen präzise zu erfassen, um den Ingenieuren eine umfassendere Unterstützung zu bieten. Die erweiterte Ontologie wird in Abb. 5.1 dargestellt. Ein auf dieser erweiterten Ontologie basierender KG ist in Abb. 5.2 abgebildet. Die Vorhersage erstreckt sich nun nicht mehr ausschließlich auf die *directlyConnectedTo*-Verbindungen, sondern bezieht auch Ports, Modul Interfaces und deren Verbindungen mit ein. Dieses erweiterte Problem kann effizient mit dem vorhandenen Modell gelöst werden, indem den Relationen Attribute hinzugefügt werden, die simultan vorhergesagt werden. Damit wäre es nicht notwendig, ein neues Modell zu entwickeln, das auf Class Prediction basiert. Vielmehr könnte, wie bisher, eine Link Prediction durchgeführt werden.

Zusätzlich ist es wichtig, dass zukünftige Studien weiterhin die Effektivität und Anwendbarkeit des Modells in unterschiedlichen Kontexten bewerten. Die Anwendung des Modells in spezialisierten Domänen wurde bis jetzt nur begrenzt erforscht und bewertet.

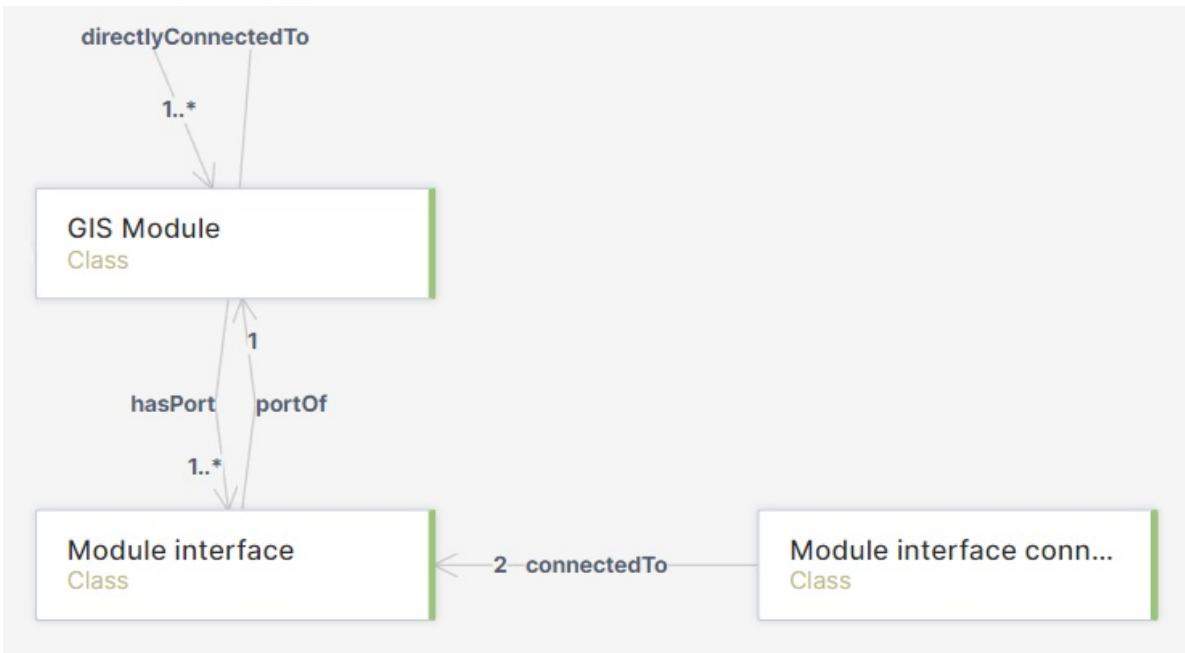


Abbildung 5.1: **Erweiterte Ontologie des Use Case:** Die erweiterte Ontologie beinhaltet das Modul Interface und die Modul Interface Connection, welche die *directlyConnectedTo*-Verbindungen detaillierter beschreiben. (Quelle: In Anlehnung an Siemens Energy, 2023)

Insgesamt unterstreicht diese Arbeit das Potenzial von Graph Neural Networks und Knowledge-Graph-Embedding-Methoden zur Vorhersage von Verbindungen in komplexen Produkt-System-Topologien. Sie sind daher wertvolle Instrumente für eine Vielzahl von Unternehmen und Anwendungen. Mit fortschreitender Forschung und Entwicklung könnten diese Methoden eine noch größere Unterstützung bieten und Ressourceneinsparungen ermöglichen, von denen mehr Unternehmen, Ingenieure und Anwendungen profitieren könnten.



Abbildung 5.2: Ausschnitt aus dem KG des erweiterten Use Case: Dieses Beispiel zeigt eine Modul Interface Connection in einem realen KG. (Quelle: In Anlehnung an Siemens Energy, 2023)

## Referenzen

- [1] **Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak und Zachary Ives.** „DBpedia: A Nucleus for a Web of Open Data“. In: *The Semantic Web*. Hrsg. von Karl Aberer, Key-Sun Choi, Natasha Noy, De-an Allemang, Kyung-II Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber und Philippe Cudré-Mauroux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 722–735. ISBN: 978-3-540-76298-0.
- [2] **Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston und Oksana Yakhnenko.** „Translating Embeddings for Modeling Multi-Relational Data“. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, S. 2787–2795.
- [3] **Tomaz Bratanic.** *A Deep Dive into Neo4j Link Prediction Pipeline and FastRP Embedding Algorithm*. 2021. URL: <https://towardsdatascience.com/a-deep-dive-into-neo4j-link-prediction-pipeline-and-fastrp-embedding-algorithm-bf244aeed50d>.
- [4] **Dimitrios Christou, Marilena Mitrouli und Jennifer Seberry**. In: *Special Matrices* 6.1 (2018), S. 155–165. DOI: doi:10.1515/spma-2018-0012. URL: <https://doi.org/10.1515/spma-2018-0012>.
- [5] **Djork-Arné Clevert, Thomas Unterthiner und Sepp Hochreiter**. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].
- [6] **Lisa Ehrlinger und Wolfram Wöß**. „Towards a Definition of Knowledge Graphs“. In: *International Conference on Semantic Systems*. 2016.
- [7] **James Fletcher**. *Knowledge Graph Convolutional Networks: Machine Learning over Reasoned Knowledge*. 2018. URL: <https://blog.vaticle.com/knowledge-graph-convolutional-networks-machine-learning-over-reasoned-knowledge-9eb5ce5e0f68> (besucht am 10. 04. 2023).
- [8] **William L. Hamilton, Rex Ying und Jure Leskovec**. „Inductive Representation Learning on Large Graphs“. In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: <http://arxiv.org/abs/1706.02216>.
- [9] **Fangfang Han, Bin Liu, Junchao Zhu und Baofeng Zhang**. „Algorithm Design for Edge Detection of High-Speed Moving Target Image under Noisy Environment“. In: *Sensors* 19.2 (2019). ISSN: 1424-8220. DOI: 10.3390/s19020343. URL: <https://www.mdpi.com/1424-8220/19/2/343>.
- [10] **IBM**. *Knowledge Graph*. n.d. (Besucht am 10. 04. 2023).
- [11] **Google Inc**. *Introducing the Knowledge Graph: things, not strings*. 2012. URL: <https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html>.

- [12] **Diederik P. Kingma und Jimmy Ba.** *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].
- [13] **Jan Eric Lenssen und Matthias Fey.** *Link Prediction on Heterogeneous Graphs with PyG*. 2022. URL: [https://medium.com/@pytorch\\_geometric/link-prediction-on-heterogeneous-graphs-with-pyg-6d5c29677c70](https://medium.com/@pytorch_geometric/link-prediction-on-heterogeneous-graphs-with-pyg-6d5c29677c70).
- [14] **Jure Leskovec, William L. Hamilton, Rex Ying und Rok Sosic.** *Representation Learning on Networks*. <https://snap.stanford.edu/proj/embeddings-www/>. Tutorial held at The Web Conference (WWW), Lyon, France, April 24, 2018. 2018.
- [15] **Vijini Mallawaarachchi.** *Inductive vs. Transductive Learning*. 2020. URL: <https://towardsdatascience.com/inductive-vs-transductive-learning-e608e786f7d> (besucht am 10. 04. 2023).
- [16] „Matrix Factorization Techniques for Recommender Systems“. In: *IEEE Computer* (2009). DOI: 10.1109/MC.2009.263.
- [17] **Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean.** *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].
- [18] **Shabnam Mokhtarani.** *Embeddings in Machine Learning: Everything You Need to Know*. 2021. URL: <https://www.featureform.com/post/the-definitive-guide-to-embeddings> (besucht am 10. 04. 2023).
- [19] **Sarang Narkhede.** *Understanding AUC - ROC Curve*. 2018. URL: <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>.
- [20] **Maximilian Nickel, Kevin Murphy, Volker Tresp und Evgeniy Gabrilovich.** „A Review of Relational Machine Learning for Knowledge Graphs“. In: *Proceedings of the IEEE* 104.1 (Jan. 2016), S. 11–33. DOI: 10.1109/jproc.2015.2483592. URL: <https://doi.org/10.1109%2Fjproc.2015.2483592>.
- [21] **Oskar Pfungst.** *Clever Hans (The Horse of Mr. Von Osten)*. Thoemmes Press, 1998.
- [22] **Phuoc. Pham.** *Understanding of Message Passing in Pytorch Geometric*. 2021. URL: <http://phuocphn.info/research/2021/12/01/understanding-of-message-passing.html#:~:text=In%5C%20PyTorch%5C%20Geometric%5C%2C%5C%20you%5C%20can%5C%20implement%5C%20Message%5C%20Passing,networks%5C%20by%5C%20automatically%5C%20taking%5C%20care%5C%20of%5C%20message%5C%20propagation..>
- [23] **Ayush Rastogi.** *DEMYSTIFYING DATA-DRIVEN NEURAL NETWORKS FOR MULTIVARIATE PRODUCTION ANALYSIS*. 2019. URL: <https://orgs.mines.edu/daa/blog/2019/08/05/neural-networks-mva/>.
- [24] **Sebastian Ruder.** *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].
- [25] **Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov und Max Welling.** *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML].

- [26] **Sachin Sharma.** *A Comprehensive Case Study of GraphSAGE Algorithm with Hands-on Experience Using PyTorch Geometric*. 2021. URL: <https://towardsdatascience.com/a-comprehensive-case-study-of-graphsage-algorithm-with-hands-on-experience-using-pytorchgeometric-6fc631ab1067> (besucht am 10. 04. 2023).
- [27] **Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever und Ruslan Salakhutdinov.** „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“. In: *Journal of Machine Learning Research* 15.56 (2014), S. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [28] **Fabian M. Suchanek, Gjergji Kasneci und Gerhard Weikum.** „Yago: a core of semantic knowledge“. In: *The Web Conference*. 2007.
- [29] **Peter Sutor, Yiannis Aloimonos, Cornelia Fermuller und Douglas Summers-Stay.** „Metaconcepts: Isolating Context in Word Embeddings“. In: *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2019, S. 544–549. DOI: 10.1109/MIPR.2019.00110.
- [30] **PyG Team.** *torch\_geometric.nn.conv.SAGEConv*. [https://pytorch-geometric.readthedocs.io/en/latest/generated/torch\\_geometric.nn.conv.SAGEConv.html#torch\\_geometric.nn.conv.SAGEConv](https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.SAGEConv.html#torch_geometric.nn.conv.SAGEConv). 2023.
- [31] **Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier und Guillaume Bouchard.** „Complex Embeddings for Simple Link Prediction“. In: *CoRR* abs/1606.06357 (2016). arXiv: 1606.06357. URL: <http://arxiv.org/abs/1606.06357>.
- [32] **Peng Wang, Baowen Xu, Yurong Wu und Xiaoyu Zhou.** „Link Prediction in Social Networks: the State-of-the-Art“. In: *CoRR* abs/1411.5118 (2014). arXiv: 1411.5118. URL: <http://arxiv.org/abs/1411.5118>.
- [33] **Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao und Li Deng.** „Embedding Entities and Relations for Learning and Inference in Knowledge Bases“. In: *CoRR* abs/1412.6575 (2014).
- [34] **Muhan Zhang und Yixin Chen.** *Link Prediction Based on Graph Neural Networks*. 2018. arXiv: 1802.09691 [cs.LG].