technische universität
dortmund

Masterarbeit

Robustheit von Graph Neural Networks zur Vorhersage von Fahrzeug-Trajektorien in urbanen Umgebungen

Shimon Wonsak
01. März 2021

Gutachter:
Prof. Dr. Katharina Morik
M. Sc. Lukas Pfahler

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für künstliche Intelligenz (LS 8)
https://www-ai.cs.tu-dortmund.de/index.html

In Kooperation mit:
Volkswagen AG

Abstract

Mit dem Ziel die Sicherheit im Straßenverkehr zu erhöhen, entwickelt die Automobilindustrie immer *intelligentere* Fahrzeuge. Eine größere Herausforderung ist es proaktiv gefährliche Situationen zu erkennen, um Unfälle zu vermeiden oder ihre Auswirkungen zu reduzieren. Durch die steigende Sensorausstattung von Fahrzeugen und hoch detailliertem Kartenmaterial können Algorithmen entwickelt werden, die die lokale Verkehrssituation für wenige Sekunden antizipieren. Im Bereich des *Machine Learning* haben *Deep Learning* Modelle für graphstrukturierte Daten für die Aufgabe der *node-level* Klassifikation beeindruckende Ergebnisse erzielt. Durch die Modellierung von Verkehrsszenen als Graph mit interagierenden Verkehrsteilnehmern lassen sich *Graph Neural Networks* (GNN) für die Vorhersage von Trajektorien einsetzen. Trotz ihrer stetig wachsenden Anwendungsdomäne ist wenig über ihre Robustheit gegenüber *adversarial attacks* bekannt. Diese Arbeit repräsentiert eine empirische Studie von *adversarial attacks* auf attributierten Graphen, mit besonderem Fokus auf Modelle die *Graph Convolutions* und *Attention*-Mechanismen einsetzen. Die Arbeit stellt drei *Graph Neural Networks* vor, die auf der Architektur für *Graph Attention* sowie *Convolution* aufbauen und dazu in der Lage sind konkurrenzfähige Vorhersagen für Trajektorien im urbanen Gelände zu berechnen. Mit Hilfe eines domainagnostischen Algorithmus für *adversarial attacks* auf dynamischen Graphen im *transduktiven* Lernverfahren wird die Verwundbarkeit von GNNs nachgewiesen. Die Angriffsszenarien umfassen direkte Angriffe auf den Knoten, dessen Trajektorie vorhergesagt werden soll sowie indirekte Angriffe auf adjazente Knoten. Die weitere Differenzierung von *adversarial attacks* zwischen den Attributen des Graphen und seiner Struktur deckt auf, dass modellspezifische Angriffsvektoren existieren, die erfolgreich darin sind die Vorhersage zu täuschen. So konnte gezeigt werden, dass die Fehlklassifikationsrate von *Attention*-basierten GNNs nach wenigen Perturbiereungen der Graphstruktur um mehr als 18% gesteigert wird. Die Ergebnisse leisten einen Beitrag dazu, die verwendeten Architekturtypen sowie ihre Anwendbarkeit in diversen Aufgabengebieten besser zu verstehen und Verfahren zu entwickeln, die sie gegen *adversarial attacks* härten.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Durch den Einsatz autonom fahrender Fahrzeuge kann die Sicherheit im Straßenverkehr erhöht und gleichzeitig die Anzahl an Verkehrsunfällen reduziert werden [44].

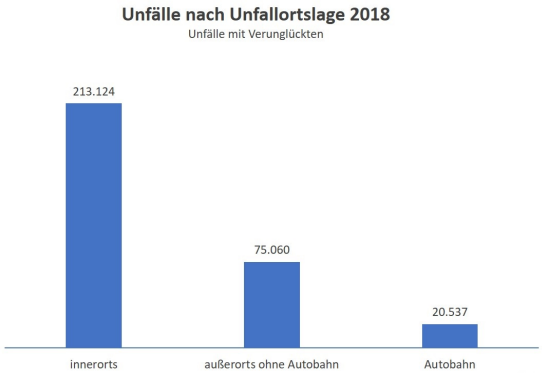


Abbildung 1.1: Von insgesamt 308.721 Unfällen 2018 haben sich 69% innerorts ereignet, wohingegen auf Landstraßen 24% bzw. Autobahnen lediglich 7% entfallen [45].

Statistiken vom Statistischen Bundesamt für Verkehrsunfälle aus dem Jahre 2018 zeigen auf, dass sich ein hoher Anteil aller Verkehrsunfälle innerorts ereignen (vgl. Abbildung 1.1). Dies lässt sich durch das ungleich höhere Verkehrsaufkommen und komplexeren Verkehrssituationen innerhalb von Städten erklären [45]. Aus diesem Grund sollten Sicherheitsaspekte für Anwendungsgebiete verbessert werden, wo sie am meisten nutzen bzw. schaden können. Ein Bestandteil dieses Sicherheitsaspektes ist die Fähigkeit des autonomen Systems das Verhalten anderer Verkehrsteilnehmer vorhersagen zu können. Dies kann einerseits dadurch geschehen, die zukünftige Absicht bzw. das Ziel eines Verkehrsteilnehmers zu bestimmen, ohne dabei Annahmen über den genommenen Weg zu treffen. Das heißt es wird ein Maneuver, wie z.B. *Rechts abbiegen* prognostiziert, unabhängig davon ob das Fahrzeug dabei über den Bürgersteig gefahren ist. Andererseits kann der konkre-

te Bewegungsverlauf eines Verkehrsteilnehmers für einen zukünftigen Zeitraum bestimmt werden. In diesem Fall wird versucht die zukünftige Trajektorie einzelner oder mehrerer naher Verkehrsteilnehmer zu antizipieren, um so proaktiv die eigene Bewegung zu planen und mögliche Unfälle zu vermeiden. Konventionelle Ansätze versuchen unter Einsatz von kinematischen und statistischen Modellen, wie z.B. *Kalman Filter*, *Hidden Markov Models* oder *Regressionsmodellen*, möglichst genaue Vorhersagen über den Systemzustand für einen definierten Zeitraum zu bestimmen. Der Straßenverkehr ist ein komplexes und dynamisches System, welches sich mit klassischen Methoden, je nach Aufwand, nur unzureichend beschreiben lässt. Die Weiterentwicklung von Deep Learning basierten Verfahren, als universelle Funktionsapproximatoren, ermöglicht es solche komplexen Systeme zu erfassen und vergleichbare bzw. bessere Ergebnisse zu erzielen [32][28]. Basis für die genannten Modelle bilden Sensordaten, in Form von LiDAR- und/oder Kameraaufnahmen. Werden diese Informationen durch einen Fehler oder absichtlich verändert, so kann dies zu unerwünschten Ergebnissen führen. Insbesondere bei Deep Learning basierten Verfahren konnte gezeigt werden, dass diese empfindlich für *adversarial attacks* sind [42]. *Adversarial attacks* versuchen die ursprüngliche Eingabe unmerklich zu verändern, sodass das Modell eine andere Klasse vorhersagt, als für die ursprüngliche Eingabe erwartet. Neben absichtlichen Manipulationen kann die Verarbeitungspipeline von den Sensordaten bis zum Modell selbst fehlerhaft sein. Zwischen dem Erfassen der Szene durch die Sensoren bis zur Verarbeitung der Informationen durch das System können beliebige Veränderungen auftreten, dessen Einfluss auf das Vorhersagemodell untersucht werden muss. Die Fähigkeit auf Perturbierungen der Eingabe nicht mit *unerwarteten* Ergebnissen zu reagieren wird als *Robustness* bezeichnet und ist in sicherheitskritischen Systeme eine zwingend erforderliche Eigenschaft [40].

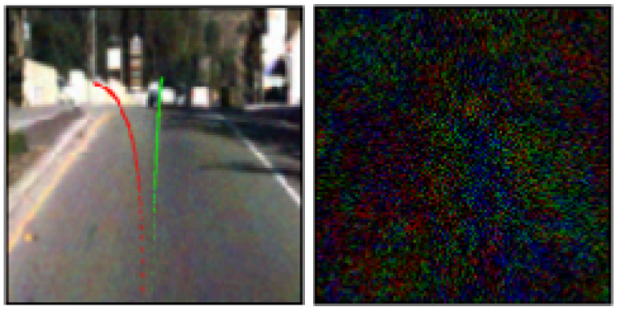


Abbildung 1.2: Links: Die unveränderte Eingabe, **Grün**: die Richtungsempfehlung bei unveränderter Eingabe, **Rot**: Richtungsempfehlung unter Einfluss der adversarial attack. **Rechts**: Die *Adversarial attack* die mittels der *Fast Gradient Sign Method* erstellt wurde. Das Rauschen wird zur Originaleingabe addiert [11].

Für erfolgreiche *adversarial attacks* ist insbesondere die verwendete Modell Architektur von Interesse, da die Angriffs-Methode stark von dieser abhängt. Während bereits diverse

Angriffsszenarien besonders im Bilderkennungsbereich für *Convolutional Neural Networks* und andere Modelle erforscht werden, gibt es für *Graph Neural Networks* (GNN) wenig Material [40]. Abbildung 1.2 zeigt die Gefahr von *adversarial attacks*. Das System trifft auf Basis von Kameradaten Entscheidungen über die Fahrtrichtung. Eine für den Menschen und auch für das System nicht erkannte Veränderung führt zu einer stark abweichende Richtungsempfehlung. Würde der Fahrer oder das System auf diese Vorhersage vertrauen, so könnte dies mitunter lebensgefährliche Folgen haben.

Um den Einsatz von Software in autonom gesteuerten Fahrzeugen sicherer zu gestalten, muss untersucht werden wie sich die verwendeten Modelle unter diversen Angrifsszenarien und Simulation defekter Sensoren verhalten. Die vorliegende Arbeit soll im Bereich der Trajektorievorhersage die *robustness* diverser GNNs unter dem Einfluss *adversarial attacks* empirisch evaluieren und dokumentieren. Die so aufgedeckten Erkenntnisse können dabei helfen robustere Algorithmen im Bereich Deep Learning zu entwickeln, die den Straßenverkehr nachhaltig sicherer gestalten.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit umfasst drei wesentliche Arbeitsschritte. Zunächst soll ein Graph Neural Network entwickelt werden, welches in der Lage ist die Trajektorie von Fahrzeugen in urbanen Gelände vorherzusagen. Die Prognose des Bewegungsverlaufes ist aktueller Bestandteil wissenschaftlicher Forschung. Im Bereich des autonomen Fahrens ist sie ein unverzichtbares Gut, um proaktiv gefährliche Situationen zu vermeiden. Um empirisch evaluieren zu können, wie sich GNN unter diversen Angriffsszenarien verhalten, muss zunächst ein Modell erstellt werden, welches auf Basis von graphstrukturierten Verkehrsinformationen Vorhersagen über die Trajektorie einzelner Fahrzeuge macht. Alahi et. al hat gezeigt, dass durch soziale Informationen die Vorhersage von Trajektorien verbessert werden kann [1]. Dabei können soziale Informationen z.B. die Position von benachbarten Verkehrsteilnehmern sein. Verkehrssituationen könnten dann für ein GNN zugänglich gemacht werden, indem für jedes Fahrzeug ein Nachbarschaftsgraph aufgebaut wird, die anschließend in einem Graphen zusammengefasst werden. Abbildung 1.3 zeigt dies Beispielhaft für eine Kreuzungsszene.

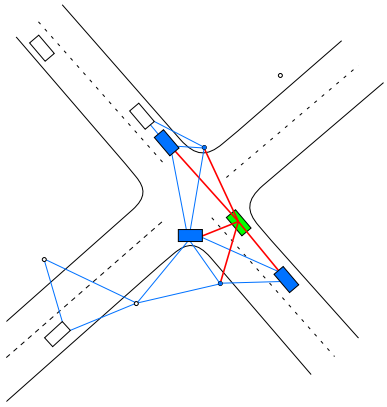


Abbildung 1.3: Rechtecke sind Fahrzeuge und Kreise können Fußgänger oder andere Verkehrsteilnehmer, wie z.B. Fahrradfahrer sein. Zusammen bilden sie die Knoten eines Graphen. Für jeden Verkehrsteilnehmer wird ein Nachbarschaftsgraph erstellt. Für das Fahrzeug von Interesse (**grün**) sind nur die Teilnehmer (**blau**) wichtig, zu denen eine **rote** Kante besteht. **blaue** Kanten repräsentieren die Nachbarschaftsgraphen von allen anderen Verkehrsteilnehmer. Der Radius, ab dem ein Teilnehmer interessant ist, kann beliebig variiert werden.

Rohe Bilddaten müssen aufwändig vorverarbeitet wrden. Daher wird im Rahmen dieser Arbeit auf den Argoverse Datensatz zurückgegriffen [7]. Dieser bietet eine 2 dimensionale Top-Down Sicht (**Bird's Eye View** BEV) auf 5 Sekunden lange Verkehrszenen aus mehreren amerikanischen Städten. Die Szenen enthalten lediglich die Koordinaten aller Verkehrsteilnehmer und bilden unter anderem Fahrspurwechsel-, Abbiege- und Bremsmanöver. Damit repräsentiert der Datensatz typische Fahrmanöver in urbanen Umgebungen. Die niedrigere Dimension von bloßen Bewegungsdaten ermöglicht schnelle Berechnungen

auf Kosten der Güte der Ergebnisse. In der *BEV* können neben Bewegungsdaten räumliche Informationen sowie soziale Interaktionen kodiert werden. Die Verwendung des Argoverse Datensatz wird zusätzlich dadurch motivert, dass neben dem Sequenzen auch **H**igh **D**efinition Maps zur Verfügung gestellt werden, die vekorisierte Informationen zum Straßensystem enthalten. Diese können einerseits genutzt werden, um zusätzliche Features für das zu evaluierende Modell zu extrahieren und andererseits, um die Ergebnis visuell darzustellen, womit die Ergbnisse optisch bewertet werden können.

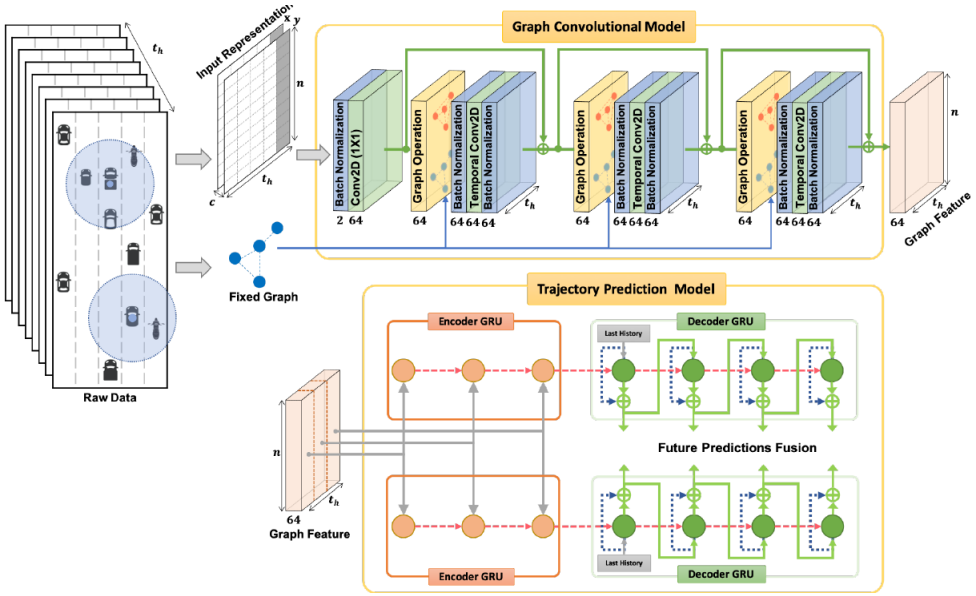


Abbildung 1.4: Grip++ verarbeitet benachbarte Fahrzeuge als Graphen mittels zeitlicher und räumlicher Faltung. Die extrahierten Features dienen als Eingabe für ein Seq2Seq Modell, um die Position aller Verkehrsteilnehmer vorherzusagen [29].

Als Inspiration für ein solches Modell soll *Grip++* dienen [29]. Grip++ ist ein mehrschichtiges Graph Neural Network, welches *Graph Features* extrahiert, um anschließend auf diesen Vorhsagen anzustellen. Wird die Verkehrsituation so in eine Graphstruktur überführt, dass nahe Verkehrsteilnehmer miteinander verbunden sind, so können soziale *higher level* Features vom GNN gewonnen werden. Das Modell soll dahingehend angepasst werden, dass in der Zeit dynamische Graphen verarbeiten werden können, die aus dem Argoverse Datensatz berechnet wurden. Die durch räumliche und zeitliche Faltung gewonnenen Features werden final in ein *Seq2Seq* Modell überführt, welches für den Agenten, das Ziel für die Trajektorievorhersage, die zukünftigen Positionen als 2-Dimensionalen Koordinaten ausgibt.
Um die Aussagekraft der Ergebnisse zu steigern und um einen Vergleich anstellen zu können, sollen mehrere Variationen des Modells erstellt werden. Neben der Baseline Version sollen die Convolutional Layer durch Attention bzw. Multi-Head Attention Layer ausgetauscht werden.

Anschließend wird die *Robustheit* jedes einzelnen Modells unter diversen Angriffsszenarien evaluiert. Dabei wird die Bedingung der *impercetibility*, d.h. die nicht- Wahrnehmbarkeit von perturbierten Eingaben, von *adversarial attacks* fallengelassen, sodass maximal schädliche Angriffe durchgeführt werden können. Da GNN zwei Angriffsvektoren bieten (Features und Graphstruktur), sollen Perturbierungen für beide Varianten für jedes Modell durchgeführt werden.
Abschließend sollen die Ergebnisse und Effektivität der *adversarial attacks* auf die unterschiedlichen Methoden untersucht werden. Dabei steht die Frage im Vordergrund, ob manche Modelle empfindlicher auf bestimmte Angriffsszenarien reagieren und wie sich dies interpretieren lässt.

Um den Leser mit den wesentlichen Konzepten vertraut zu machen, beschäftigt sich Kapitel 2 mit den essentiellen Bestandteilen aller verwendeten Bausteine und Ideen. Da Trajektorienvorhersage mittels Graph Neural Networks und *adversarial attacks* auf diesen aktueller Forschungsgegenstand ist, bietet Kapitel 3 einen Überblick über verwandte Themen, sodass diese Arbeit in den aktuellen Kontext eingeordnet werden kann. Kapitel 4 bespricht das Problem der Trajektorievorhersage, den verwendeten Datensatz und die Erstellung der Modelle und Angriffsszenarien im Detail. Abschließend erfolgt in Kapitel 5 eine umfassende Untersuchung und Bewertung der durchgeführten Angriffe auf die Modelle, die in Kapitel 6 in einem Fazit zusammenfassend beurteilt werden und einen Ausblick darauf geben, welche weitere Möglichkeiten diese Arbeit bietet.

Kapitel 2

Grundlagen

Das Ziel der Arbeit besteht aus zwei Aufgaben. Zunächst soll ein Machine Learning Verfahren entwickelt werden, mit dessen Hilfe es möglich ist, auf Grundlage von Beobachtungen, die zukünftige Bewegung eines Fahrzeugs vorherzusagen. Diese Aufgabe umfasst mehrere Teilaufgaben einer Pipeline, die in Kapitel 4 näher erläutert werden. In den nachfolgenden Abschnitten werden die nötigen Konzepte des verwendeten *Deep Learning* Modells beschrieben.
Anschließend werden sogenannte *adversarial attacks* auf das entwickelte Modell angewendet, um zu evaluieren auf welche Weise das Modell auf gezielt manipulierte Änderungen der Eingabe reagiert. Die nötige Taxonomie und verwendeten Konzepte von *adversarial attacks* für Neural Networks werden im Kapitel 2.3 erläutert.

2.1 Graph Neural Networks

Deep Learning hebt sich von anderen Verfahren im Bereich des Machine Learnings durch die Tatsache ab, dass Modelle dieser Kategorie für das Problem relevante Features selbst extrahieren und verarbeiten können. Das *Universal Approximation Theorem* besagt, dass Neural Networks mit nur einem *hiden layer* jede beliebige Funktion approximieren können [46]. Jedoch können diese Modelle beliebig breit werden, um ein gegebenes Problem zu lösen. Mit Convolutional Neural Networks (CNN) wurde gezeigt, dass systematische Anpassung der Architektur besser dazu in der Lage sind relevante Features im Bereich Compute Vision aus Bildern zu extrahieren [26]. Graph Neural Networks (GNN) umfassen eine weitere Familie von Architekturen im Bereich Deep Learning, die dazu in der Lage sind graphbasierte Daten zu verarbeiten. Um tiefere Einblicke in die Verarbeitung von Graphdaten mittels Graph Neural Networks zu bekommen, müssen zunächst Graphen definiert werden.

Ein ungerichteter Graph G lässt sich repräsentieren als $G = (V, E)$, wobei V die Menge aller Knoten und E die Menge aller Kanten ist. Dann ist $v_i \in V$ ein Knoten und $e_{ij} = (v_i, v_j) \in E$ die Kante, die die Knoten v_i und v_j miteinander verbindet. Die Nachbarschaft eines Knotens v ist definiert als $N(v) = \{u \in V | (u, v) \in E\}$. Die Adjazenz-Matrix A ist eine $n \times n$ Matrix, die an den Stellen $A_{ij} = 1$ ist, wenn $e_{ij} \in E$ gilt, und $A_{ij} = 0$, falls $e_{ij} \notin E$. Weiterhin kann jeder Knoten über eine Menge von Attributen $x_i \in \mathbb{R}^F$ verfügen, wobei $X \in \mathbb{R}^{N \times F}$ die Menge aller Features der N Knoten mit der Dimension F sind [49]. Ein Beispiel für einen solchen Graphen ist in Abbildung 2.1 dargestellt.

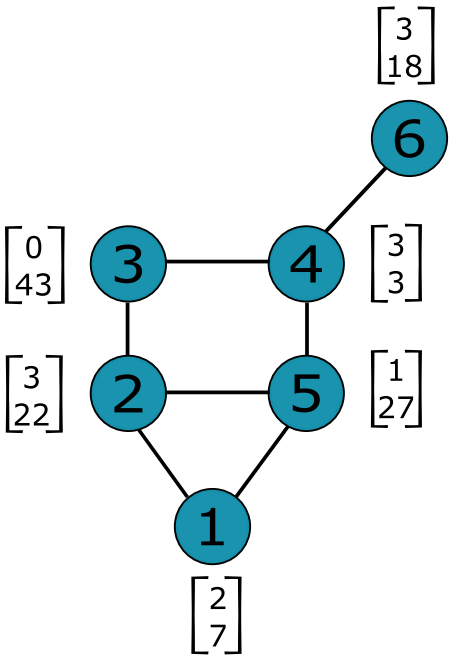


Abbildung 2.1: Ein ungerichteter Graph mit $|V| = 6$ Knoten. Jeder Knoten hat Features x_i. Alle Attribute umfassen die Dimension $X \in \mathbb{R}^{6 \times 2}$

Diese Darstellung eines Graphen ist für den Menschen gut interpretierbar, jedoch können neuronale Netze diese Informationen nicht verarbeiten, daher wird der Graph in die Matrixnotation überführt:

$$X^{6\times 2} = \begin{bmatrix} 2 & 7 \\ 3 & 22 \\ 0 & 43 \\ 3 & 3 \\ 1 & 27 \\ 3 & 18 \end{bmatrix} \qquad A^{6\times 6} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

X und A enthalten alle Informationen, die in der Definition gefordert sind. Jede Zeile von X repräsentiert die Attribute eines Knotens, wobei jede Spalte für eine konkrete Merkmalsausprägung steht. Jede Zeilen- bzw. Spaltennummer von A korrespondiert zu ei-

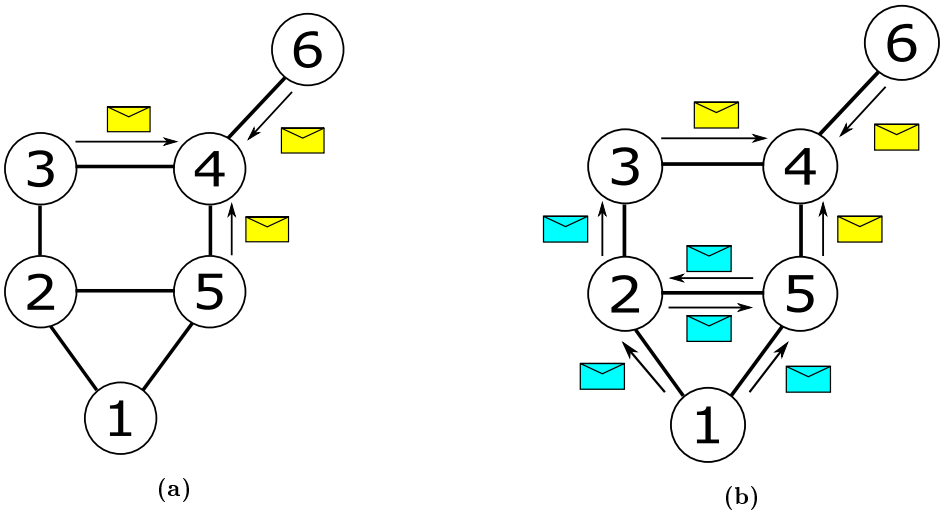


Abbildung 2.2: Konzeptionelle Sicht von Message Passing für einen Knoten. **(a)** Nach dem ersten Message Passing Schritt kennt Knoten 4 die aggregierten Attribute seiner direkten Nachbarn. **(b)** Nach dem zweiten Message Passing Schritt kennt Knoten 4 die aggregierten Attribute von 2 und 1, die im ersten Schritt in den Knoten 3 und 5 zusammengefasst wurden.

ner Knotennummer, sodass bspw. der Eintrag A_{34} für die Kante zwischen Knoten 3 und 4 steht. Das Ziel von GNN ist es für jeden Knoten eines Zustandsvektor $\vec{h}_i$ zu lernen, der die Attribute seiner Nachbarn aggregiert. Um diesen Zustandsvektor zu erhalten dürfen nur die Features von direkten Nachbarn in einem Knoten aggregiert werden. Die Grundlage für dieses Verfahren bildet das sogenannte *Message Passing* bzw. die *Propagation Rule*, das angibt welche Attribute miteinander verrechnet werden [52]. In seiner einfachsten Form entspricht dies:

$$H = AX \tag{2.1}$$

Mit Hilfe des Beispiels kann man sich verdeutlichen, dass jeder Eintrag der Matrix $H \in \mathbb{R}^{N \times F}$ die Summe aller Attribute der Nachbarknoten ist. Wird diese Operation wiederholt, so enthält ein Knoten bereits Informationen von den Nachbarn seines Nachbarn. Die Anzahl der Iterationen legt fest, wieviele Nachbarn erreicht werden sollen. Ein Beispiel für zwei Message Passing Schritte ist in der Abbildung 2.2 illustriert.

Erweitert man die Formel 2.1 um eine Gewichtsmatrix $\theta^{F \times F'}$:

$$H = AX\theta \tag{2.2}$$

so kann die Propation Rule in ein Neural Network integriert werden. Die Integration von θ erlaubt es Knoten *higher level* Features zu lernen und gleichzeitig ihre Dimension zu

verändern, denn die Dimension von $H \in \mathbb{R}^{N \times F}$ hängt von θ ab. Die konkrete Ausprägung der Propagation Rule ist das Herzstück eines GNN das graphstrukturierte Daten verarbeiten kann. Zwei Architekturvarianten, *Graph Convolution Networks* und *Graph Attention Networks*, die in dieser Arbeit verwendet wurden, werden in den nachfolgenden Kapiteln vorgestellt.
Mit Hilfe von GNN können Graphen auf mehrere Arten analysiert werden. Die Repräsentation der Daten als Menge von Knoten mit zugehörigen Attributen bietet die Möglichkeit sich einzelne Knoten herauszunehmen und damit sogenannte **Node-level** Klassifikations-/Regressionsaufgaben durchzuführen. Führt man den Message Passing Schritt oft genug durch, so ergibt sich ein stabiles Equilibrium in allen Knoten Attributen. Dies kann beispielsweise in einem **Semi-Supervised** Setting, wo es nicht zu jedem Knoten ein Label gibt, genutzt werden, um das Label von nicht gelabelten Knoten vorherzusagen. Ein häufig zitiertes Beispiel ist das *Zachary's Karate Club* Netzwerk (siehe 2.3).

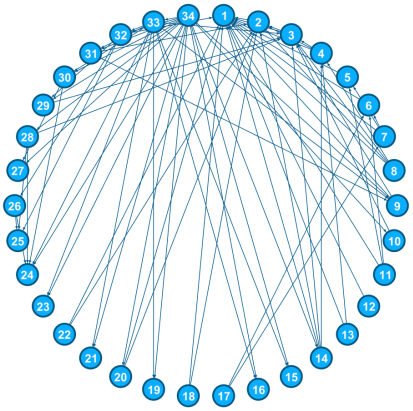


Abbildung 2.3: Zarachy's Karate Club: Knoten sind Mitglieder. Eine Kante bedeutet, dass zwei Mitglieder eine Freundschaft führen. [51]

Zarachy's Karate Club ist ein Beispiel für ein soziales Netzwerk, in dem Knoten Mitglieder von verschiedenen Karate Vereinen repräsentieren und Kanten die Freundschaftsbeziehung unter Mitgliedern anzeigen. Werden nur einigen dieser Knoten Label zugewiesen, handelt es sich um einen Semi-Supervised Anwendungsfall. Während des Trainings werden Knoten Embeddings sich an den Zustand der Nachbarn anpassen. Unter der Annahme, dass gleiche Klassenzugehörigkeiten sich im Graphen räumlich (über die Kantenbeziehung) nah beieinander befinden, weil sich beispielsweise zwei Freunde ähnliche Werte teilen, können hohe Klassifikationsgüten erreicht werden.

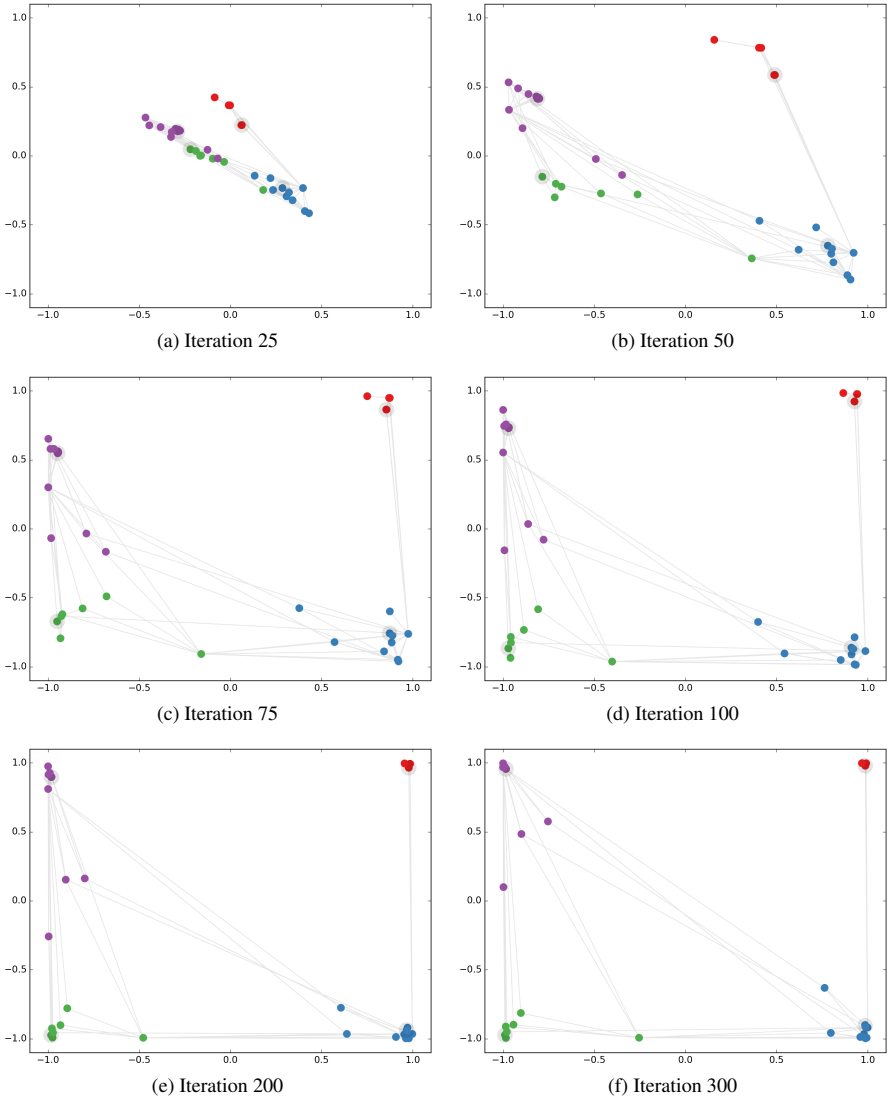


Abbildung 2.4: Zarachys Karate Club: Es gibt 4 Knoten, wobei jede Klasse nur durch einen gelabelten Knoten (grau hinterlegt) repräsentiert ist. Jeder Knoten besitzt 2 zufällig initialisierte Merkmale, um die Vorhersage darstellen zu können. Die Wahre Klasse jedes Knoten ist durch die entsprechende Farbe markiert. Das GNN wurde mit einem Graph Convolutional Network trainiert. Bei steigender Iterationszahl Bewegen sich ungelabelte Knoten auf ihre Wahre Klasse zu. [25]

Die Ergebnisse aus Grafik 2.4 zeigen, dass sich gute Vorhersagen über die Klassenzugehörigkeit über räumliche Beziehung treffen lassen, selbst wenn nur wenige Knoten gelabelt sind. Neben der Möglichkeit GNN für Semi-Supervised Node-Level Klassifikation einzusetzen, können sie ebenfalls für **Graph-Level** Aufgaben eingesetzt werden. Statt einzelnen Knoten kann die gesamte Feature Matrix zur Weiterverarbeitung verwendet werden, z.B. dann wenn der gesamte Graph eine Klasse, die es zu bestimmen gilt, repräsentiert [49]. Insbesondere zeigt das Zacharys Karate Club Beispiel eine weitere Besonderheit von GNNs auf, so ist es möglich derartige Modelle im sogenannten **Transductive** Lernverfahren zu

trainieren. In diesem Fall besteht sowohl das Trainingsset, als auch das Testset aus denselben Daten. Die Verwendung von Transductive Modellen ist insbesondere bei der Analyse von statischen Graphen von Relevanz, wo beispielsweise inhärente soziale Beziehungen untersucht werden sollen.
Diese Arbeit verwendet den **Inductive** Ansatz, d.h. das Modell sieht zum Trainings-/Testzeitpunkt unterschiedliche Graphen. Das Ziel ist es ein Konzept zu lernen und den Generalisierungsfehler minimal zu halten. Konkret heißt dies, dass mehrere Sequenzen von aufgezeichneten Verkehrsituationen als Graphen kodiert werden. Jeder Verkehrsteilnehmer entspricht einem Knoten, welche unter anderem mit ihrer Position in der Szene attribusiert sind. Das Modell soll mit der aggregierten Nachbarschaft die Trajektorie vom Autonomous Vehicle vorhersagen. Die Vorhersage entspricht reellwertigen Koordinaten, sodass es sich um eine supervised Nodel-level Regressionsaufgabe handelt.

2.1.1 Graph Convolutional Network

Graph Convolutional Networks (GCN) von T. Kipf & M. Welling sind eine Adaption von spektralen Filtern, um higher-level Features in Graphen zu erzeugen [25]. Spektrale Filter haben ihren Ursprung in der Signalverarbeitung und erlauben es Signale über deren Frequenzspektrum zu manipulieren. Die ursprüngliche Idee für sogenannte *spectral graph filter* wurde durch Convolutional Neural Networks inspiriert. Die Faltung eines Bildes mittels eines Kernels ist das Pendant zum spektralen Filter. Ein digitales Bild kann durch diskretisierte Farbwerte dargestellt werden. In dieser Darstellung kann das Bild mit einem sogenannten Kernel gefaltet werden. Abbildung 2.5 zeigt ein Beispiel für eine solche Faltung. Der Kernel, orange hinterlegte Matrix, wird über jeden Eintrag in der Bildmatrix gelegt, wo jeder Wert des Kernels mit dem gedeckten Wert in der Bildmatrix multipliziert wird. Die aufsummierten Werte werden in einer neue Matrix, an der Stelle wo sich der Kernel in der Bildmatrix befindet, geschrieben.

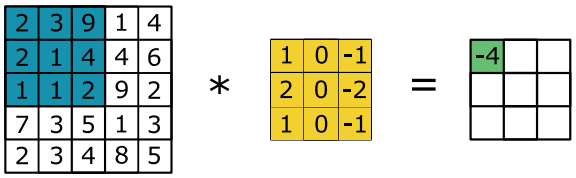
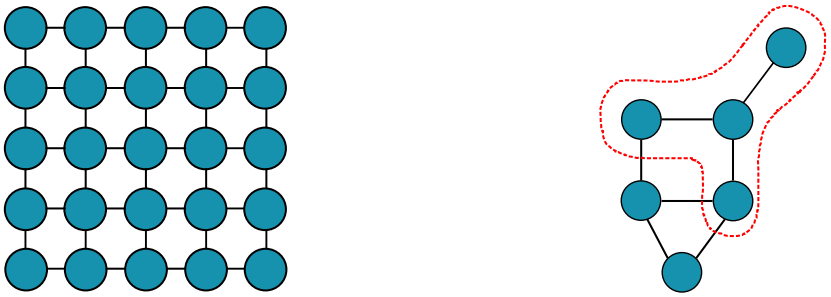


Abbildung 2.5: Links die diskretisierte Matrix der Farbewerte *Image*. Jeder Eintrag repräsnetiert einen Pixel. Der Einfachheithalber wird nur ein Farbkanal dargesellt. Die 3×3 Matrix θ in der Mitte ist der sogenannte Sobel-Filter, mit dem Kanten in einem Bild detektiert werden können. Das Ergebnis ist eine Feature Map F, die Features aus der Eingabe enthält.

Die Formel 2.3 formalisiert diese Operation für C Farbkanäle.

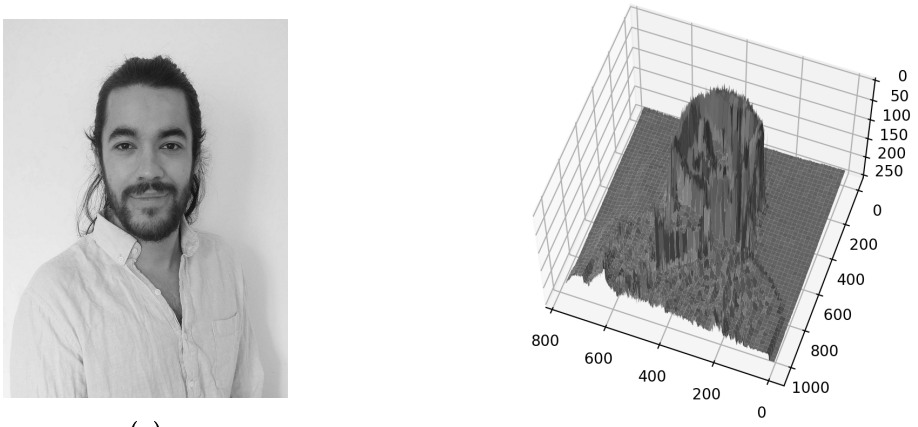
$$F_{X,Y} = \sum_{c=1}^{C} \sum_{i=1}^{K} \sum_{j=1}^{K} \theta_{i,j} * Image_{X+i,Y+j} \tag{2.3}$$

Die 3×3 Matrix θ, die den Kernel repräsentiert ist der sogenannte *Sobel-Filter* [39]. Mit Hilfe des Sobel-Filters können Kanten in einem Bild erkannt werden. Filter die durch die Faltungsoperation Features extrahieren können, haben den Vorteil, dass sie **lokal** sind, d.h. sie benötigen nicht die gesamte Information des Bildes, um Informationen zu extrahieren. Convolutional Neural Networks setzen auf diesen Sachverhalt, um eine Menge von Kernels selbsständig zu lernen, um sogenannte *Feature Maps* zu gewinnen, die higher-level Features der Eingabe entsprechen.
Eine direkte Übertragung der Faltung von Bildern auf Graphen scheitert an der Tatsache, dass Graphen keine fixe Struktur haben. Würde man jeden Pixel als Knoten betrachten und ihn mit all seinen direkten Nachbarn verbinden, so würde sich ein strukturierter Graph ergeben auf dem weiterhin die Faltungsoperation angewendet werden könnte (vgl. Abbildung 2.6a).



(a) (b)

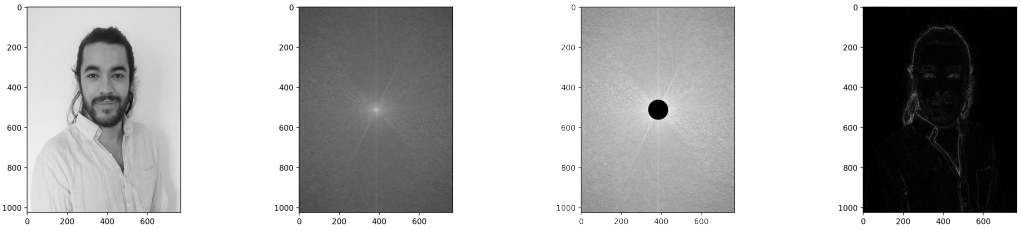
Abbildung 2.6: (a) Die Pixelmatrix aus Abbildung 2.5 überführt in einen Graphen. Jeder Pixel ist ein Knoten. Alle direkten Nachbarn der Matrix werden durch Kanten verbunden. **(b)** Ein Graph mit unterschiedlichen Nachbarschaftsstrukturen. Eine Faltung ist auf dieser Struktur nicht möglich, da nicht klar ist welcher Knoten zu welchem Eintrag im Kernel korrespondiert.



(a) (b)

Abbildung 2.7: (a) Ein Bild nur mit Grauwerten zwischen $0-255$ dargestellt. Je heller der Pixel, desto höher der Grauwert. **(b)** Dasselbe Bild dargestellt als Höhenkarte. Die Z-Koordinate kodiert die Grauwerte.

Für Graphen gilt dies jedoch nicht. Der Graph aus Abbildung 2.6b lässt sich nicht ohne weiteres in eine Gitterstruktur überführen. Eine Faltung ist auf dieser Struktur nicht definiert und es gäbe mehrere Probleme zu lösen. So ist beispielsweise nicht klar in welcher Reihenfolge welcher Knoten mit welchem Eintrag im Kernel multipliziert werden soll und wie Nachbarschaften gefaltet werden sollen, die den Kernel nicht vollständig ausfüllen, wie im dargestellten Beispiel.
Eine weitere Möglichkeit, denselben Effekt zu erzielen, bieten die zuvor angesprochenen spektralen Filter. Werden Bilder aufgenommen, so wird ein zeitveränderliche Signal, z.B. Photonen, gemessen und diskretisiert. Das Result am Ende ist ein 2-dimensionales Bild, welches ein räumliches Signal (**spatial domain**) zu einem festen Zeitpunkt repräsentiert (vgl. 2.7b).



(a) **(b)** **(c)** **(d)**

Abbildung 2.8: Filterung eines Signals im Frequenzspektrum: **(a)** Ein Grauwert Bild. **(b)** Das Frequenzspektrum des Bildes. Je heller der Eintrag, desto größer ist die Amplitude des Frequenzanteils. **(c)** Der schwarze Kreis in der Mitte symbolisiert einen Hochpassfilter. Alle kleinen Frequenzen werden *ausgeschnitten*. **(d)** Das gefilterte Bild, nach dem das Frequenzspektrum mit Hilfe der inversen Fouriertransformation zurück in die räumliche Darstellung transformiert wurde.

Abbildung 2.7b zeigt eine Höhenkarte des Grauwertbildes links daneben. Die X, Y Achse entspricht den Dimensionen des Ursprungbildes. Auf der Z-Achse wurden die Grauwerte im Bereich von $0 - 255$ abgetragen, wobei höhere Werte hellere Farbtöne kodieren. Auffällig an der Grafik ist, dass sie Muster aufweist, die durch Kombinationen von 2D *sinus* und *cosinus* Funktionen dargestellt werden können, d.h. Bilder können mit Hilfe der Fouriertransformation in ihr Frequenzspektrum (**frequency domain**) zerlegt werden. Die diskrete Fouriertransformation im 2D lautet wie folgt:

$$F(u,v) = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} f(x,y) \cdot e^{-i2\pi\left(\frac{ux}{X} + \frac{vy}{Y}\right)} \tag{2.4}$$

$$= \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} f(x,y) \left[cos\left(\frac{2\pi ux}{X} + \frac{2\pi vy}{Y}\right) - i \cdot sin\left(\frac{2\pi ux}{X} + \frac{2\pi vy}{Y}\right)\right] \tag{2.5}$$

Damit kann jedes Signal als Linearkombination der orthogonalen Basis-Funktion von *sinus* und *cosinus* geschrieben werden. Abbildung 2.8b zeigt das Frequenzspektrum für das Beispielbild. Jeder Punkt entspricht der Frequenz und Richtung der *sinus* bzw. *cosinus* Welle in die entsprechende Koordinatenrichtung. Die Helligkeit kodiert die Amplitude der Frequenz. Das Frequenzspektrum kann genutzt werden, um das Originalbild zu filtern. Würden man, wie in Abbildung 2.8c dargestellt, alle niedrigen Frequenzen herausfiltern, einen sogenannten *Hochpassfilter* anwenden, dann würde das gefilterte *Invers Fouriertransformierte* Frequenzspektrum alle Kanten hervorheben.

Um einen Filter über dem Frequenzspektrum zu trainieren ist das gesamte Bild nötig, da dieses zunächst überführt werden muss, das heißt spektrale Filter sind **globale** Filter. Für neuronale Netze bedeutet dies, dass eine Gewichtsmatrix dieselben Dimensionen wie die Eingabe haben muss.
Das Bindeglied zwischen der Faltung und spektralen Filter ist das *Convolution Theorem*:

$$x * \theta = F^{-1}(F(x) \cdot F(\theta)) \tag{2.6}$$

$$x \cdot \theta = F^{-1}(F(x) * F(\theta)) \tag{2.7}$$

Es sagt aus, dass eine Faltung mit einem Kernel θ in der *spatial domain* das gleiche ist, wie eine Multiplikation mit einem Filter θ in der *frequency domain* und umgekehrt [8]. Da sich die Faltung in der spatial domain für Graphen nicht ohne weiteres übertragen werden kann, kann ein äquivalenter Filter in der frequency domain definiert werden. Graphen sind nicht das Resultat einer Linearkomibination von *sinus* und *cosinus* Funktionen, stattdessen repräsentiert die **Laplace Matrix** des Graphen dessen Frequenzspektrum. Die Laplace Matrix ist die Differenz zwischen der *Degree*- und Adjazenz-Matrix:

$$L = D \cdot I - A \tag{2.8}$$

wobei $I, A, D \in \mathbb{R}^{N \times N}$ und I ist die Einheitsmatrix für den Graphen mit N Knoten. Die Degree Matrix eine ist Matrix, wo auf der diagonalen für jeden Knoten die Anzahl seiner Nachbarn zusammengefasst ist:

$$d_{ii} = \sum_{j=1}^{N} A_{ij} \tag{2.9}$$

Für den Beispielgraphen aus Kapitel 2.1 sieht die Laplace Matrix wie folgt aus:

$$L = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} * I - \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

L ist eine symmetrische und positiv semidefinite Matrix und hat damit orthogonale Eigenvektoren $U = [u_0, ..., u_{n-1}] \in \mathbb{R}^{N \times N}$, die stellvertretend für die Basis-Funktionen der Fouriertransformation stehen und nicht-negative Eigenwerte $\Lambda = diag([\lambda_0, ..., \lambda_{n-1}]) \in \mathbb{R}^{N \times N}$, die die Frequenz des Graphen sind [5] und es gilt $L = U\Lambda U$. Anlehnend an das Convolution Theorem haben Bruna et. al die **spectral graph convolution** definiert:

$$x *_G y = U((U^T x \odot U^T y)) \tag{2.10}$$

wobei $\odot$ das Hadamard Produkt ist, die elementweise Matrixmultiplikation. Damit ist eine Faltung in der spatial domain dasselbe wie eine Multiplikation, des mittels Eigenvektoren

transformierten Signals und Filters. $\odot$ entspricht der elementweisen Multiplikation. Ein Signal kann nun mittels eines Filters g im Frequenzbereich Λ wie folgt gefiltert werden:

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x \tag{2.11}$$

Ein neuronales Netz kann einen Filter lernen, indem die Filter-Funktion $g_\theta(\Lambda)$ durch trainierbare Koeffizienten ersetzt wird, dabei lässt sich die Transformation ins Spektrum direkt durch die Koeffizienten ersetzen.

$$g_\theta(\Lambda) = diag(U^T g) = diag(\theta) \tag{2.12}$$

Die Matrix ist nur auf der Diagonalen besetzt, jedoch hat sie den Nachteil, dass ihre Größe von der Größe des Graphen abhängt, denn $\theta \in \mathbb{R}^N$. Ein weiterer Nachteil dieser Methode ist, dass das Konzept vom *Message Passing* keine Anwendung findet, denn die *spectral graph convolution* bezieht den ganze Graphen zur Filterung des Signales ein und arbeitet somit nicht mit einer lokalen Nachbarschaftsbeziehung. Defferrard et. al lösen diese Probleme, indem sie die Faltung in der frequency domain durch Folgen von orthogonalen Funktionen ersetzen [10]. Solche Folgen können mit dem **Tschebyshow-Polynom erster Art** erzeugt werden:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{2.13}$$

wobei $T_0 = 1$ und $T_1 = x$ ist. Statt den Filter in der frequency domain zu definieren, kann das *Tschebyshow Polynom* des Grades K genommen werden:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) \tag{2.14}$$

wobei $\tilde{L} = \frac{2L}{\lambda_{max}} - I_N$ die skalierte Laplace-Matrix ist, deren Eigenwerte zwischen $[-1, 1]$ sind. Die Definition des Filters auf diese Weise hat mehrere Vorteile. Zum einem hängt die Dimension der trainierbaren Parameter $\theta \in \mathbb{R}^K$ von dem Grad des Polynoms ab. Zum anderen sorgt die Verwendung der Laplace Matrix als Argument für das Tschebyshow-Polynom dafür, dass der Filter für jeden Knoten nur auf der Nachbarschaft, die in K Schritten erreichbar ist, agiert. Das bedeutet der Filter ist K **lokalisiert** [10]. Die Definition von Defferrard et. al benötigt jedoch weiterhin eine rechenintensive Eigenwertzerlegung, daher schlagen N. Kipf & M. Welling eine Approximation vor, die ohne eine Eigenwertzerlegung auskommt. Die Autoren gehen von Formel 2.14 aus und argumentieren, unter der Annahme, dass jeder Knoten nur ein skalares Attribut $x \in \mathbb{R}^N$ hat, dass weiterhin eine große

Klasse von Filtern erreicht werden kann, wenn man das Tschebyshow-Polynom auf Grad $K = 1$ beschränkt. Damit lässt sich die *spectral graph convolution* vereinfachen zu:

$$g_\theta * x \approx \theta'_0 x + \theta'_1 (L - I)x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \tag{2.15}$$

$L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ entspricht der normalisierten Laplace Matrix. Weiterhin nehmen sie an, dass sich der größte Eigenwert der Laplace-Matrix im Verlauf des Trainings bei $\lambda_{max} \approx 2$ einstellen wird. Unter dieser Annahme lässt sich die Eigenwertzerlegung der Laplace-Matrix sparen und die Skalierung entspricht nun dem Term für das Tschebyshow-Polynom ersten Grades. Da dies zu numerischen Instabilitäten führt wurde die Laplace Matrix mit Hilfe des *re-normalization tricks* normalisiert:

$$\tilde{L} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \tag{2.16}$$

wobei $\tilde{A} = A + I$ die Adjazenz-Matrix plus den sogenannten **self-loops** ist. Das heißt jeder Knoten hat eine Kante zu sich selbst. Die Degree-Matrix $\tilde{D}$ wurde entsprechend auf dieser Adjazenz-Matrix definiert. Um die Anzahl an Matrixmultiplikationen weiter zu reduzieren wird die Anzahl der 2 freien Parameter θ weiter eingeschränkt, sodass sich folgende Formel ergibt:

$$g_\theta * x \approx \theta \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right) x \tag{2.17}$$

Verfügt der Graph über N Knoten mit Attributen der Dimension $X \in \mathbb{R}^{N \times F}$, dann lässt sich die Formel generalisieren und es ergibt sich:

$$H^{l+1} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l \Theta^l \right) \tag{2.18}$$

wobei $\Theta^l \in \mathbb{R}^{F \times F'}$ der Filter des l-ten Layers ist, der die Knoten Features von Dimension F nach F' transformiert, sodass sich die neuen Knoten Attribute zu $H^l \in \mathbb{R}^{N \times F'}$ ergeben. σ ist eine nicht-lineare Aktivierungsfunktion. Auf diese Weise können mehrere Graph Convolution Layer geschichtet werden. GCN sind rechengünstige Implementierungen für spektrale Graph convolution und es wurde empirisch nachgewiesen, dass sie ihren Vorgängermodellen ebenbürtig sind [25]. Trotz des komplexen Ursprungs in der spektralen Signalverarbeitung, ähnelt die finale Struktur GCN stark der intuitiv herleitbaren Formel für die *Propagation Rule*.

2.1.2 Graph Attention Network

Das in Kapitel 2.1.1 vorgestellte Graph Convolutional Network hat den Nachteil, dass das Gewicht, mit dem die Features der Nachbarn aggregiert werden, abhängig von der Struktur des Graphen ist. Somit hat das Netzwerk nicht die Möglichkeit sich auf Nachbarn zu konzentrieren, deren Informationen besondere Relevanz für den Entscheidungsprozess enthalten.
P. Veli˘ckovi´c et al. beschreibt *Graph Attention Networks* (GAT) mit denen das oben genannte Problem umgangen werden kann [47]. Die Verwendung eines solchen Modells lässt sich für diese Arbeit dadurch motivieren, dass im realen urbanen Straßenverkehr nicht jedes Fahrzeug gleich viel Einfluss auf das Fahrverhalten anderer Verkehrsteilnehmer hat.
Graph Attention Networks sind eine Architekturvariante für Neuronale Netzwerke, die graphbasierte Daten verarbeiten können. Das von T. Kipf & M. Welling vorgestellte GCN aggregiert im Message Passing Schritt alle direkt erreichbaren Nachbarn eines Knotens mit Hilfe der normalisierten Laplace Matrix:

$$I_n + D^{-0.5}AD^{-0.5}$$

Dadurch ist die Convolution für einen Knoten auf der direkten Nachbarschaft lokalisiert, jedoch ist der Einfluss jedes Nachbarn durch die Struktur des Graphen vorbestimmt. Ziel von GAT Layern ist es für jeden Knoten i und jeden Nachbarn $j \in N(i)$ ein Gewicht α_{ij} zu lernen, dass die Wichtigkeit von j's Features für i widerspiegelt. Dieses Gewicht α wird *Attention* genannt und ist unabhängig von der Struktur des Graphen. Darüber hinaus sind GAT Layer dazu in der Lage die Dimension der Features zu verändern.
Als Eingabe für den Layer werden die Knoten Feautres $h = \{\vec{h}_1, \vec{h}_2, ..., \vec{h}_N\} \in \mathbb{R}^F$, wobei N die Anzahl der Knoten ist, und eine Adjazenz Matrix $A^{N\times N}$, die die Verbindungen zwischen Knoten kodiert, benötigt. Je nach gewünschter Dimension erhält man die Ausgabe $h' = \{\vec{h'}_1, \vec{h'}_2, ..., \vec{h'}_N\} \in \mathbb{R}^{F'}$, die wie folgt berechnet werden kann. Zunächst werden die Features aller Knoten einer linearen Transformation mit einer Gewichtsmatrix $W \in \mathbb{R}^{F'\times F}$ unterzogen.

$$\vec{z_i} = W\vec{h_i} \tag{2.19}$$

Diese Transformation erlaubt es dem Modell sogenannte *higher-level* Features zu erlernen, die in einer besseren Modellperformance resultieren können. Anschließend werden, unter Zuhilfenahme der Adjazenzmatrix A, die transformierten Features von Knoten i mit denen seiner Nachbarn j konkateniert und mit einer weiteren Gewichtsmatrix $\vec{a} \in \mathbb{R}^{2F'\times 1}$ multipliziert. $\vec{a}$ lässt sich als ein einlagiges feed-forward Perzeptron interpretieren, dass

die Wichtigkeit der Nachbarschaftsbeziehung, nach einer nicht linearen Transformation, in einem Koeffizienten e ausdrückt.

$$e_{ij} = LeakyReLU(\vec{a}^T(\vec{z_i}|\vec{z_j})) \tag{2.20}$$

wobei | die Konkatenation zweier Featurevektoren ist und *LeakyReLU* die *leaky Rectified Liniar Unit* ist mit

$$LeakyReLU(x) = \begin{cases} z & if z > 0 \\ \alpha z & if z \leq 0 \end{cases} \tag{2.21}$$

mit $\alpha \neq 0$. e_{ij} spiegelt das Kantengewicht zwischen Knoten i und seinem Nachbarn j wider. Jedoch ist dieser Koeffizient nicht normalisiert und kann beliebige Werte annehmen. Um die negativen Folgen von Skalierungseffekten zu reduzieren, müssen die Werte normalisiert werden, sodass die Summe über alle Kanten eines Knotens, also seinen Nachbarn $k \in N(i)$, gleich 1 ist. Mit Hilfe der normalisierten Exponentialfunktion

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k \in N(i)} exp(e_{ik})} \tag{2.22}$$

können die Kantengewichte normalisiert werden. Die resultierenden α_{ij}-Werte werden Attention Koeffizienten genannt und werden abschließend die ursprünglichen Gewichte ersetzen. Die neuen Features $\vec{h_i'}$ von Knoten i können nun als gewichtete Summe von α_{ij} und den transformierten Nachbarfeatures $\vec{z_j}$ gewonnen werden.

$$\vec{h_i'} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij} \vec{z_j}\right) \tag{2.23}$$

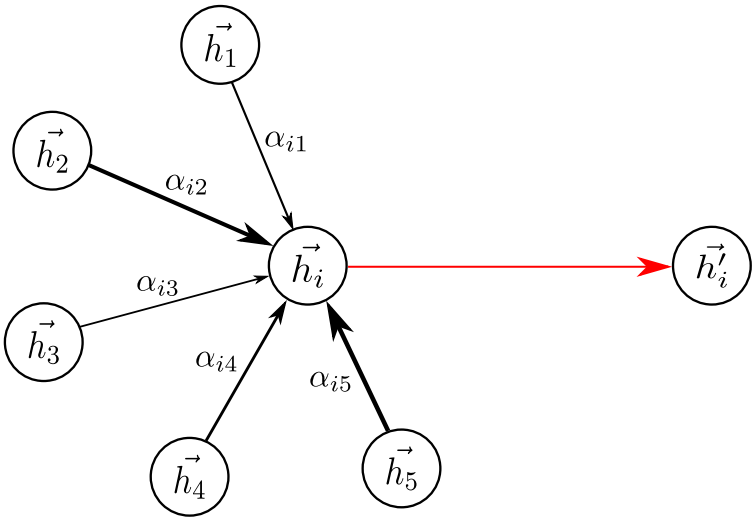


Abbildung 2.9: Graph Attention: Knoten i aggregiert die Features seiner Nachbarn $\vec{h_j}$ mit den Attention Koeffizienten $\alpha_i j$ auf. Dickere Kanten symbolisieren ein höheres Gewicht.

Die Verwendung von Graph Attention Networks löst einige Probleme im Bereich von Graph Neural Networks, die insbesondere für die Verarbeitung von großen Datenmengen eine wichtige Rolle spielen. So lässt sich die Attention Matrix eines GAT Layers effizient berechnen, da sie im Gegensatz zu früheren Arbeiten im Bereich der spektralen Graph Convolution [5], [10] keine Eigenwertzerlegung benötigt. In einem GAT Layer muss für jeden Knoten $v \in V$ eine Transformation von F nach F' durchführt werden und für jede Kante $e \in E$ eine Transformation von F' nach 1. Wenn $|V|$ und $|E|$ jeweils die Größen der Knonten bzw. Kantenmenge sind, so lässt sich die Komplexität für den Rechenaufwand mit $O(|V|FF' + |E|F')$ ausdrücken. Ein weiterer Vorteil ist die Tatsache, dass Graph Attention Networks unabhängig von der Graphstruktur Attention Koeffizienten berechnen können. Wie Graph Convolutional Layer können GAT daher im Inductive Learning Bereich eingesetzt werden, wo der Testdatensatz neue, noch nicht gesehene Graphen, enthält. Um den Lernprozess unter Verwendung von Graph Attention Layern zu stabilisieren können K GAT Layer in sogenannten *multi head* Attention zusammengefasst werden. Jeder GAT Layer hat seinen eigenen Satz an lernenden Gewichtsmatrizen, sodass eine diverse Menge an α Attention Koeffizienten berechnet werden kann. Anschließend wird über die Attention aller GAT Layer gemittelt

$$\vec{h'_i} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N(i)}\alpha_{ij}\vec{z_j}\right) \tag{2.24}$$

2.2 Sequence to Sequence

Um Ausgaben variabler Länge zu erzeugen sind *vanilla* neuronale Netze nicht geeignet. Sie benötigen eine Eingabe fixer Länge, um eine Ausgabe im vordefinierten Format zu erzeugen. Damit sind sie nicht in der Lage Daten zu bearbeiten, deren Dimension nicht einheitlich ist. Ein Beispiel bildet die Sprache, wo jeder Satz unterschiedliche viele Wörter behinhaltet. Die Bedeutung des Satzes wird erst durch die korrekte Aneinanderreihung von Wörtern klar, die sich aufeinander beziehen. Ein Lösungsansatz bildet das Konzept von *Sequence to Sequence*-Modellen. Diese wurden erstmals von Ilya et. al eingeführt und ermöglichen es Vorhersagen variabler Länge zu erzeugen, die von beliebig vielen Eingaben abhängen können.
Vanilla neuronale Netze sind, bei entsprechender Tiefe, dazu in der Lage belieibige Funktionen zu lernen. Wie in Grafik 2.10 dargestellt wird die Eingabe x_1, x_2, x_3 in der ersten Schicht mit diversen Gewichten W^1_{ij} multipliziert und die Resultate anschließend addiert, um die *hidden representations* h_1, h_2 zu erhalten. Diese wird wiederum mit Gewichten W^2_{jk} multipliziert und addiert, um das finale Ergebnis y_1 zu berechnen.

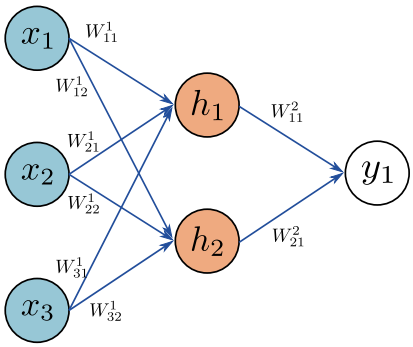


Abbildung 2.10: Unverändertes feedforward neuronales Netz: Die Eingabe fester Größe wird ohne Rückkoppelungen zur Ausgabe y_1 transformiert. Inspiriert durch [30].

Besteht die Aufgabe daraus eine zuvor nicht festgelegte Menge von Y Ausgaben zu bestimmen, die selbst auf einer unbekannten Eingabegröße X beruht, so müsste für jede Ein-/Ausgabegröße ein eigenes Netzwerk trainiert werden. Daraus resultieren zwei Nachteile. Erstens würden die Netzwerke, aufgrund unterschiedlicher Gewichte W, keine Informationen miteinander teilen, was dazu führt, dass das gesamte Problem in kleinere Probleme unterteilt wird, ohne den Gesamtkontext zu lernen. Zweitens wird ein eventueller Kontext innerhalb der Ein-/Ausgabe ignoriert. Beispielsweise entfaltet sich die Bedeutung eines Satzes mit der Zeit, mit jedem weiteren gesprochenen Wort. Die Architektur 2.10 enthält für jede Eingabe x_i eigene Gewichte und bildet eine innere Repräsentation als gewichtete Summe. Die Bedeutung der Sequenz wird damit auf dem Weg zur Ausgabe vermischt. Als Lösung bieten sich *Recurrent Neural Networks* (RNN) an, die sich durch die Tatsa-

che auszeichnen, dass die Informationen wieder zurück ins Netz gespeist werden können. Abbildung 2.11 stellt ein rekurrentes Netz schematisch dar.

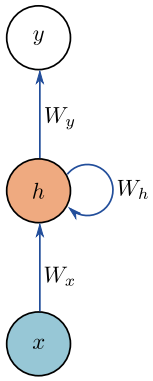


Abbildung 2.11: Rekurrentes neuronales Netz: Verbindungen zu Neuronen, deren Verarbeitungsschritt zeitlich vorgelagert sind, können zusammenhängende Datenströme in einen gemeinsamen Kontext verarbeiten. Inspiriert durch [30].

Der *hidden state* verfügt über eigene Parameter W_h. Nachdem die Eingabe x in den *hidden state* h überführt wurde, wird dieser an die Ausgbe, über W_y weitergeleitet, gleichzeitig wird der Zustand h gespeichert. Den Parameter W_h kommt die Aufgabe aus dem Zustand h *higher level* Features zu gewinnen, indem diese mit der Eingabe x verknüpft wird. Erhält das Netzwerk eine weitere Eingabe, so kann diese nach Überführung in den *hidden state* mit dem alten Zustand verknüpft werden, sodass Informationen aus vorherigen Eingaben in die neue Ausgabe mit einfließen können. Auf diese Weise kann mit Hilfe von geteilten Gewichten W_h die Eingabe mit vorherigen Informationen verbunden werden und das Netzwerk ist in der Lage zu *erinnern*. Abbildung 2.12 repräsentiert dasselbe Netzwerk wie in Grafik 2.11 als Vorwärtsgerichtetes Modell.

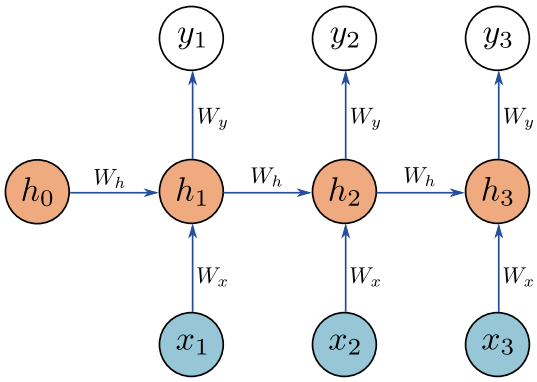


Abbildung 2.12: Rekurrentes neuronales Netz: Entfaltete Version aus Abbildung 2.11. Jeder Verarbeitungsschritt in der Zeit entspricht einem Layer. Inspiriert durch [30].

Ausgehend von einem initialen Zustand h_0 werden sequentiell Eingaben x_t in das Modell getätigt. In jedem Schritt t wird x_t mit dem vorherigen *hidden state* h_{t-1} verbunden

und dient als Eingabe für den nächsten Schritt. Das Netzwerk kann variable Lange Eingabesequenzen entgegennehmen und erzeugt Ausgaben der gleichen Länge.
Während des Trainings von rekurrenten Netzen kann es zum sogenannten *vanishing/exploding* Gradienten kommen. Rekurrente Netzen teilen sich die Gewichte W, weshalb die Fehlerrückführung in der Zeit auf denselben Parametern operiert. Während der Rückfürhung des Fehlerterms, um die Gewichte W anzupassen, kann es bei tiefen Netzen dazu kommen, dass der Fehler exponentiell schnell anwächst oder schrumpft [35].
Gated recurrent units (GRU) sind eine architektonische Abwandlung der standard RNN's, die das Problem des *vanishing/exploding* Gradienten lösen und darüber hinaus relevante Zusammenhänge während des iterativen Bearbeitungsprozesses speichern können. Sind beispielsweise Wörter zu Beginn eines Satzes wichtig, die sich auf nachfolgende Wörter beziehen, so kann diese Informationen in einem GRU gespeichert werden. Schematisch ist die *Gated recurrent Unit* in Abbildung 2.13 dargestellt.

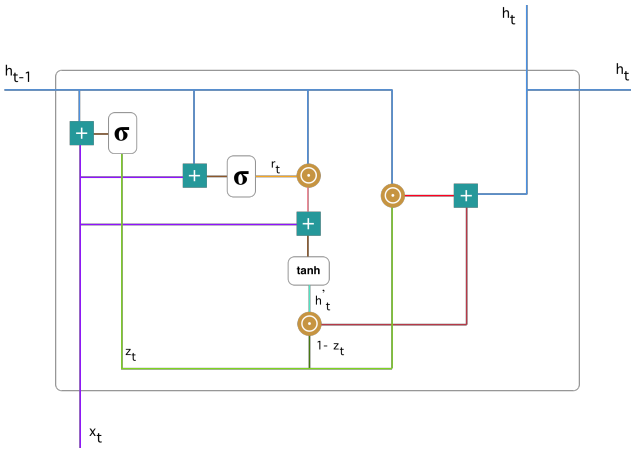


Abbildung 2.13: Gated Recurrent Unit: Der innere Zustand h_t wird durch das *Reset* und *Update Gate* aktualisiert. Die Mechanismen ermöglichen es alte Informationen aus h_{t-1} zu vergessen und mit neueren Informationen aus der Eingabe x_t zu überschreiben [38].

Der Zustand h_t des GRU's, der gleichzeitig die Ausgabe in einem Schritt darstellt, ist abhängig von vier Operationen. Zusammengenommen erlauben diese Operationen dem Modell Informationen in h_t wahlweise zu vergessen und durch neue Versionen aus der aktuellen Eingabe x_t zu ergänzen. Das *Update Gate* hilft dem Modell dabei zu entscheiden welche Informationen vom vorherigen Zeitschritt h_{t-1} behalten werden sollen.

Update Gate:

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1}\right) \tag{2.25}$$

Die Eingabe x_t und der *hidden state* des vorherigen Zeitschritts werden mit ihren eigenen Gewichten W_z, U_z multipliziert und anschließend addiert. Die *Sigmoid*-Funtion $\sigma(x)$ bildet dieses Ergebnis auf einen Bereich zwischen $[0, 1]$ ab und bildet somit z_t.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.26}$$

Analog dazu entscheid das *Reset Gate* wieviel Informationen aus dem vorherigen Zeitschritt h_{t-1} behalten werden sollen.

Reset Gate:

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1}\right) \tag{2.27}$$

Durch die Tatsache, dass sich Werte von r_t ebenfalls im Intervall $[0, 1]$ befinden, können diese wie folgt dazu eingesetzt werden, um Informationen aus h_{t-1} zu löschen.

$$h'_t = tanh\left(W_h x_t + U_h\left(r_t \odot h_{t-1}\right)\right) \tag{2.28}$$

$\odot$ bezeichnet die elementweise Multiplikation zweier Matrizen. Sind Einträge in r_t nahe an eins, so werden die entsprechenden Informationen aus dem vorherigen Zeitschritt h_{t-1} behalten, andernfalls werden sie auf 0 gesetzt. Dieser Wert wird mit einem *Gate* spezifischen Gewicht U_h multipliziert und mit der Eingabe x_t addiert, die zuvor mit der Matrix W_h transformiert wurde. Auf dieses Ergebnis wird die nicht lineare Aktivierungsfunktion *tanh* angwendet, um h'_t zu erhalten.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.29}$$

Um die finale Ausgabe zu erhalten, werden die zu Beginn berechneten *update* Werte z_t dazu benutzt, um zu entscheiden welche Werte aus dem vorherigen Zeitschritt h_{t-1}, und dem aktuellen h_t, behalten werden sollen.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{2.30}$$

Die Ausgabe, und der neue *hidden state*, h_t wählt aus h_{t-1} nur solche Werte aus, an denen z_t nahe eins ist. Umgekehrtes gilt für den aktuellen Zustand h'_t. Dieser enthält sowohl die transformierten Informationen der Eingabe x_t, als auch den aktualisierten Anteil von h_{t-1}. Sind Informationen aus früheren Zeitschritten relevanter als aktuellere, dann wird $z_t = 1$ die Informationen aus h_{t-1} fortführen und, aufgrund von $1 - z_t$, die aus h'_t ignorieren. Die Verwendung eines *rekurrenten* Netzes mit *Gated Recurrent Units*, wie in Abbildung 2.11 vorgestellt, beinhaltet das Problem, dass die Sequenzlänge der Eingabe gleichzeitig die Länge der Ausgabe bestimmt. Ilya et. al stellen als Lösung *Sequence to Sequence* (Seq2Seq) Modelle vor, die ihren Ursprung im *natural language processing* haben [41]. Ihr Aufbau ist schematisch in Grafik 2.14 dargestellt.

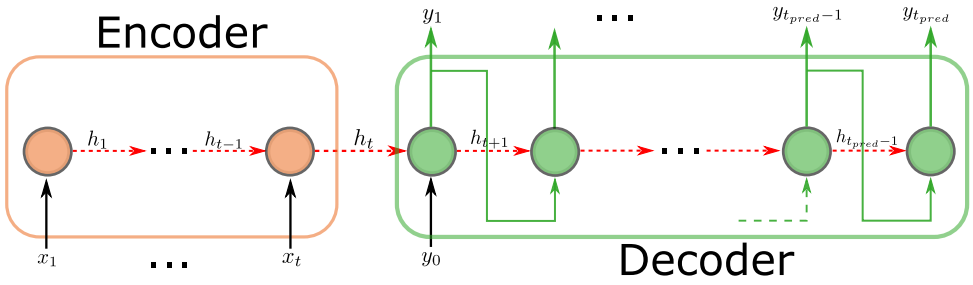


Abbildung 2.14: Sequence to Sequence Modell: Der *Encoder* Part erzeugt eine *hidden representation* der Eingabe. Der aggregierte Zustand der Eingabe kann im *Decoder* Modell dazu genutzt werden variable Lange Ausgabeströme zu erzeugen.

Das Modell besteht aus zwei voneinander unabhängigen Modulen. Der **Encoder** enthält eine varibale Menge an rekurrent vernetzen Neuronen, z.B. GRU's. Jeder orangene Kreis markiert eine solche Einheit. Für eine Eingabesequenz der Länge t wird eine *hidden representation* Schritt für Schritt aufgebaut. Dazu wird in jedem Zeitschritt die Einheit mit der Eingabe $x_i, i = 1, ..., t$ und dem *hidden state* aus dem Vorschritt h_{i-1} versorgt. Für den ersten Zeitschritt kann diese Eingabe beliebig gewählt werden. Oft entspricht diese einem Nullvektor. Nachdem die Eingabesequenz abgearbeitet wurde, enthält h_t die aggregierten Informationen der gesamten Eingabe. Bei dieser Konstruktion ist es möglich beliebig Lange Eingaben in eine *hidden representation* zu überführen, z.B. indem eine Eingabe solange bearbeitet wird bis ein Stopzeichen erreicht wird. Die schematisch Zeichnung repräsentiert ein Vorwärtsgerichtetes Netzwerk, dennoch teilen sich alle GRU's die Gewichte. Die komprimierten Informationen h_t dienen als Eingabe für das **Decoder** Modell, welches eine Ausgabe variabler Länge erzeugen kann. Analog zum *Encoder* wird ein GRU wiederholt rekurrent durchlaufen, um in jedem Zeitschritt ein Ausgabe $y_j, j = 1, ..., t_{pred}$ zu erzeugen. Statt eines *hidden state* benötigt der *Decoder* eine initiale Eingabe y_0, die im Fall dieser Arbeit die letzte beobachtete Position eines Fahrzeugs sein kann. Alle nachfolgend berechneten Ausgaben dienen als Eingabe für den nächsten Schritt. Daneben wird der *hidden state* h_{j-1} fortwährend aktualisiert. Die Ausgabe wird mit der Ausgabe eines Stopzeichens oder durch eine vordefinierte Ausgabelänge abgeschlossen. Auf diese Weise lassen sich mit zwei GRU's beliebige Eingabe-/Ausgabesequenzen verarbeiten und erzeugen.

2.3 Adversarial Attacks

C. Szegedy et. al wiesen erstmal die Existenz von sogenannten **blind spots** in Deep Neural Networks nach [42]. Bei einem Klassifikationsmodell würde man erwarten, dass sich das Ergebnis für eine Eingabe $\tilde{x}$ nicht von x unterscheidet, falls beide Eingaben nah beieinander liegen, es also gilt $|\tilde{x} - x| \leq \epsilon$, für ein kleines ϵ. Formal lässt sich dieser Zusammenhang für eine Modell f und dem Label j von x beschreiben als:

$$\arg\max_j f(x) = \arg\max_j f(\tilde{x}) \tag{2.31}$$

Allgemein gilt diese Gleichung nicht und der Nachweis ihrer Gültigkeit für ein gegebenes Modell erfordert komplexe Zertifizierungsstrategien, die formal beweisen können, dass das Modell auch für Varianten von x in einer ϵ-Umgebung die selbe Klasse vorhersagen. Die Erkenntnis von *blind spots* bedeutet, dass es ähnliche Eingaben gibt, die ein anderes Ergebnis hervorrufen. Die Tatsache, dass wenige Perturbierungen von x zu einer Fehlklassifikation führen können, sorgte dafür das $\tilde{x}$ **adversarial examples** genannt werden [42]. Die Autoren fanden darüber hinaus heraus, dass *blind spots* nicht modellspezifisch sind oder durch mangelhaftes Training hervorgerufen werden, denn sie konnten zeigen, dass unabhängig voneinander, mit unterschiedlichen Datensätzen, trainierte Modelle sich *adversarial examples* teilen. Dieses Phänomen wird **transferbility** genannt. *Adversarial examples* lassen sich in gezielt durchgeführte Angriffe, *adversarial attack*, und fehlerhafte Eingaben, *corruption*, die z.B. durch einen Sensorfehler entsteht, unterteilen. Als Antwort auf die Erkenntnis von *blind spots* wurden viele wissenschaftliche Arbeiten veröffentlicht, die einerseits versuchen *adversarial examples* mit den unterschiedlichsten Eigenschaften zu erstellen und andererseits Modelle gegen derartige *Angriffe* zu schützen [20]. Eine Möglichkeit die Eingabe zu Perturbieren bietet die sogenannte **F**ast **G**radient **S**ign **M**ethod (FGSM):

$$\tilde{x} = x + \epsilon * sign(\nabla_x L(\theta, x, y)) \tag{2.32}$$

$L(\theta, x, y)$ ist die Kostenfunktion mit den Modellparametern θ und der wahren Klasse y. Statt, wie im Training üblich, die Kostenfunktion nach θ abzuleiten, um die Koeffizienten anzupassen, wird der Gradient für die Eingabe x bestimmt. Bei einem trainierten Modell kann auf diese Weise der Einfluss der Eingabe auf die Vorhersage bestimmt werden. Um ein *adversarial example* zu erstellen wird ein kleiner Schritt ϵ, in die Richtung gegangen, die den Fehler am stärksten erhöht. Um den Modellfehler zu erhöhen verfolgt das Verfahren den Gradienten in Richtung des steilsten Anstiegs für die Eingabe. Bereits für kleine ϵ können Perturbierungen mit weitreichenden Folgen erreicht werden (vgl. Abbildung 2.15) [15].

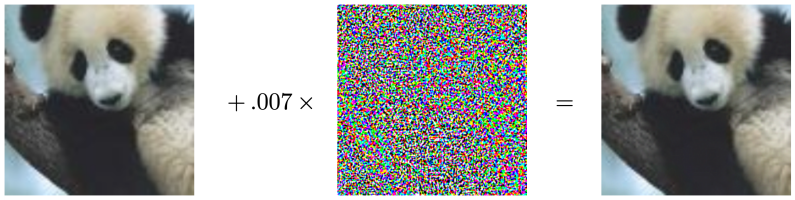


Abbildung 2.15: Die unveränderte Eingabe links wurde mit einem trainierten Bildklassifikator als Panda klassifiziert mit einer Konfidenz von 57.7%. Auf diese wird anteilig ($\epsilon = .007$) das *adversarial* Rauschen addiert, um das *adversarial example* auf der Rechten Seite zu erhalten. Das Bild rechts wurde mit einer Konfidenz von 99.3% als Menschenaffe klassifiziert. Ein Mensch erkennt weiterhin einen Panda [15].

FGSM ist nur eine von vielen Methoden und benötigt das gesamte Modell um eine Perturbierung berechnen zu können. Angriffsmethoden, die auf alle Informationen des Modells zurückgreifen werden **White Box** Angriffe genannt. Dem gegenüber stehen Methoden, die beispielsweise über Heuristiken oder randomisierte Veränderungen *adversarial examples* berechnen. Diese Formen benötigen keine tiefergehenden Informationen über das Modell, wie bspw. seine Modellparameter θ, und werden **black bock** Angriffe genannt [40].
Um Perturbierungen untereinander vergleichen zu können, muss das *adversarial example* mit dem Original vergleichbar sein. Dies ist mit einer Metrik möglich, die wie folgt definiert ist:

- $\|x\| \geq 0$ Nicht-Negativität
- $\|x - y\| = 0 \Rightarrow x = y$ Identität
- $\|x - y\| = \|y - x\|$ Symmetrie
- $\|x - y\| + \|y - z\| \geq \|x - z\|$ Dreiecksungleichung

Häufig verwendete Distanz Metriken sind L_p **Normen** für $1 \leq p < \infty$, die wie folgt definiert sind:

$$\|x\|_p = \left(\sum_{d=1}^{D} |x_d|^p \right)^{\frac{1}{p}} \tag{2.33}$$

Wobei D der Dimension des Eingabevektors x entspricht. Die L_2 Norm (Euklidische Distanz) wird im Computer Vision Bereich benutzt, da sich dazu eignet die Veränderung für den menschlichen Betrachter zu quantisieren [6]. Somit können sogenannte **imperceptible**, also nicht wahrnehmbare, *adversarial examples* erstellt werden, wenn das ϵ unter einem bestimmten Schwellwert liegt. Sie lässt sich ebenso zur Messung von perturbierten Fahrzeugtrajektorien verwenden, da sich die Bewegung im 2-dimensionalen euklidischen Raum befindet. Für graphstrukturierte Daten eignet sich die L_0 Norm,

$$\|x\|_0 = |\{x_d | x_d \neq 0, d = 1, ..., D\}| \tag{2.34}$$

die misst, wieviele Einträge ungleich 0 sind. Die L_0-Norm erfüllt nicht die Dreiecksungleichung und ist damit nicht Teil der zuvor definierten Metriken. Damit lässt sich vergleichen, wieviele neue Kanten in einer Adjazenz-Matrix hinzugekommen oder entfernt wurden.
Mit Hilfe dieser Bausteine lassen sich *imperceptible adversarial examples* formal definieren. Eine Eingabe x gilt als fehlerhaft klassifiziert, wenn ein Orakel H, dass die wahre Klasse j kennt, etwas anderes vorhersagt als das Modell f.

$$\arg\max_j f(x) \neq \arg\max_j H(x) \tag{2.35}$$

Wird diese Definition um eine Distanzmetrik und ein ϵ, dass die maximale Entfernung von der Originaleingabe angibt, erweitert, dann ist eine *adversarial attack* eine perturbierte Eingabe $\tilde{x}$ für die folgendes gilt:

$$\begin{aligned} \arg\max_j H(\tilde{x}) = \arg\max_j H(x) \\ \wedge \|x - \tilde{x}\|_p \leq \epsilon \\ \wedge \arg\max_j f(\tilde{x}) \neq \arg\max_j H(x) \end{aligned}$$

Umgangssprachlich bedeutet dies, dass ein Orakel die Eingabe x nicht von dem *adversarial example* $\tilde{x}$ unterscheiden kann. Weiterhin hält sich die Perturbierte Eingabe in einer vordefinierten ϵ-Umgebung vom Original auf und das Modell sagt für $\tilde{x}$ eine andere Klasse vorher [20]. Diese allgemeine Definition fordert lediglich, dass das *adversarial example* die Vorhersage auf eine andere Klasse $i \neq j$ ändert. Dieser Fall wird **untargeted** Angriff genannt. Der andere Fall zielt darauf ab, die Eingabe so zu manipulieren, dass eine gewünschte Vorhersage erreicht wird. Die formale Definition für diesen Fall wird hier ausgelassen und lässt sich in der Literatur unter dem Begriff **targeted** Angriff wiederfinden [40].
Mit der Existenz von *adversarial examples* geht die Frage einher, wie Robust Neuronale Netze auf derartige Eingaben reagieren. Bezogen auf die Definition 2.3 ist ein Modell robust gegen lokale Perturbierungen, wenn gilt:

$$\forall x : \|x - \tilde{x}\|_p \leq \epsilon \Rightarrow \arg\max_j f(x) = \arg\max_j f(\tilde{x}) \tag{2.36}$$

Das heißt, dass sich für alle Eingaben und ihren perturbierten Gegenstücken, innerhalb eines definierten Radius, die Ausgabe des Modells nicht ändert. Diese Definition hat zwei Nachteile. Zu einem ist sie auf der wahren Verteilung von x definiert, die unendlich groß ist und nicht gemessen werden kann. Zum anderen muss für jede Eingabe das minimal gültige ϵ bestimmt werden, denn dieser Wert kann sich von Objekt zu Objekt unterscheiden, falls für ein x gilt, dass dieses mitten in einer Klasse liegt und ein anderes direkt an der Grenze zu zwei Klassen.

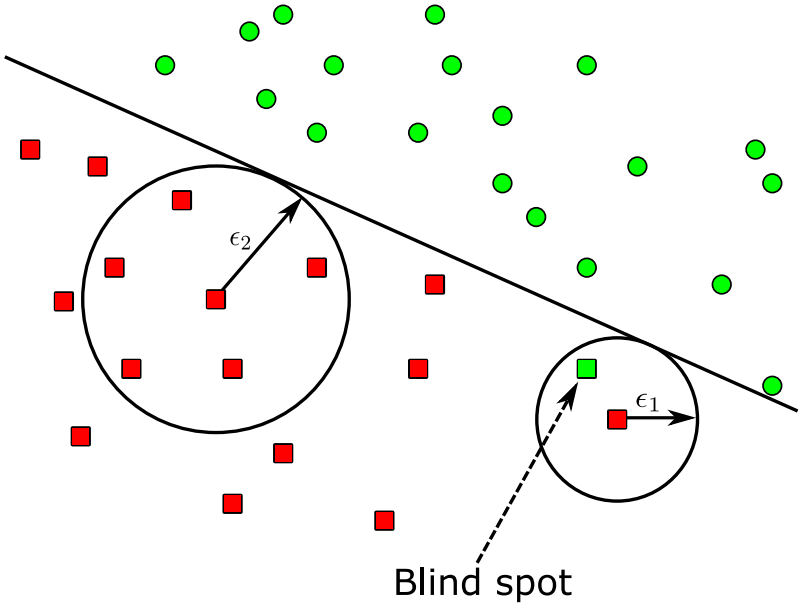


Abbildung 2.16: Zwei linear seperierte Klassen: Um das lokale Robustheitskriterium zu erfüllen, muss für jedes Objekt sein minimal gültige ϵ-Umgebung bestimmt werden, in der sich die Vorhersage nicht ändert. Der Abstand zur Klassifikationsgrenze ist kein gültiger Abstand, da sich innerhalb der Umgebung *blind spots* befinden können, für die *adversarial examples* existieren.

Daher wird üblicherweise die globale Robustheit für eine Eingabemenge DB definiert:

$$\forall x, \tilde{x} \in DB : \|x - \tilde{x}\|_p \leq \epsilon \Rightarrow |L(f, x, j) - L(f, \tilde{x}, j)| < \delta \tag{2.37}$$

Die globale (ϵ, δ)-Robustheit fordert, dass sich die Wahrscheinlichkeit für die Wahre Klasse von x, für x und dem *adversarial example*, die sich in einem Abstand von weniger als ϵ zueinander befinden, in maximal δ unterscheidet [24].
Für graphstrukturierte Daten gelten dieselben Regeln, jedoch muss berücksichtigt werden, dass GNN neben den Eingabefeatures x, die für Knoten Attribute stehen, noch eine Adjazenzmatrix A erhalten, die angibt wie Knoten untereinander verbunden sind. Die Eingabe kann somit als Graph $G = (A, x)$ zusammengefasst werden. Da sich der Informationsfluss in einem Graphen während des *Message Passing* Schrittes signifikant ändern kann, wenn neue Kanten hinzukommen oder alte entfernt wurden, muss die Adjazenzmatrix als weiterer Angriffsvektor in die Analyse von *adversarial attacks* mit einbezogen werden. Angriffe

die auf die Attribute der Knoten abzielen werden **feature attacks** genannt. Perturbierungen an der Graphstruktur werden als **structure attack** bezeichnet [40]. Sind mehr als N Perturbierungen für einen Graphen vorgesehen, so können auch Kombinationen beider Angriffe durchgeführt werden.

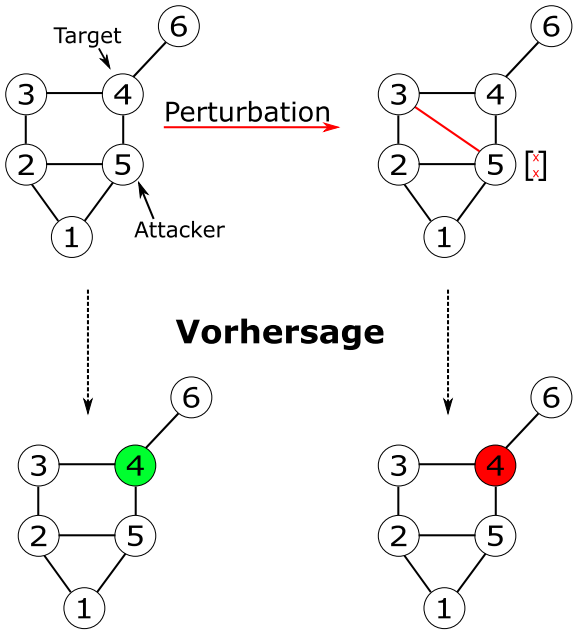


Abbildung 2.17: Target Knoten Nr. 4 hat die drei **attacker** Nachbarn 3, 5 und 6. Vor dem Angriff wird Knoten 4 richtig klassifiziert. Nachdem sowohl die Features, als auch die Struktur, von *attacker* 5 perturbiert wurden, wird Knoten 4 falsch klassifiziert

Wird das GNN zur *Node-level* Klassifikation eingesetzt, so kann darüber hinaus festgelegt werden welcher Knoten angegriffen werden soll. Wie in Abbildung 2.17 dargestellt, gibt es im *Node-level* Setting zwei Gruppen von Knoten. Der Knoten der klassifiziert werden soll wird **target** genannt und hat nichts mit den zuvor beschriebenen *targeted* Angriffen zu tun. Alle Nachbarn des *targets* sind die **attacker**, manchmal als *influencer* bezeichnet, denn sie sind aufgrund der *propagation rule* in der Lage die Klasse für den *target* Knoten indirekt zu manipulieren [53].
Für GNN *adversarial examples*, bei denen die Eigenschaft für *impercetibility* wegfallen kann, lässt sich die ursprüngliche Definition in zwei Komponenten aufteilen. Zu einem soll es möglich sein ein GNN mittels *Feature*-Angriffen zu täuschen, sodass gilt:

$$\max_{\hat{x}_i \in \Phi(\hat{x}_{n-1})} \sum_{n=1}^{N} L(f(\hat{x}_i, A), y) \tag{2.38}$$

Wobei $\Phi(x)$ der Prozess ist, der die *adversarial examples* erzeugt, mit $\hat{x}_0 = x$. In N aufeinanderfolgenden Perturbierungsschritten sollen die Fehlerkosten des Modells maximiert

werden. Im Zweiklassenfall lässt sich diese Definition in Abhängigkeit von Fehlklassifikationen, wie in Defintion 2.3 schreiben, indem ab überschreiten des Fehlers $L(f(\hat{x}_i, G), y)$ über einen definierten Threshold d, die andere Klasse gewählt wird.

Äquivalent kann für *structure* Angriffe definiert werden:

$$\max_{\hat{A}_i \in \phi(\hat{A}_{n-1})} \sum_{n=1}^{N} L(f(x, \hat{A}_i), y) \tag{2.39}$$

Wobei $\phi(A)$ der Prozess ist, der zum Graphen sowohl Kanten hinzufügen, als auch entfernen kann, mit $\hat{A}_0 = A$. Für $N > 1$ können beide Kriterien zusammengefasst werden, sodass abkürzend gefordert werden kann, dass die gemeinsame Anwendung von *structure* und *feature* Angriffen, den Fehler maximieren. Wobei in jedem Schritt nur der Angriff, der beiden möglichen, durchgeführt wird, der den Fehler maximiert [40].

$$\max_{\hat{G}_i \in \Psi(\hat{G}_{i-1})} \sum_{n=1}^{N} L(f(\hat{G}_i), y) \tag{2.40}$$

Wobei $\hat{G}_i = (\hat{A}_i, \hat{x}_i)$ und $\hat{G}_0 = (A_0, x_0)$

Kapitel 3

Verwandte Arbeiten

Sowohl das Fortführen von Zeitreihen, als auch die Erforschung der Verwundbarkeit von neuronalen Netzen sind aktueller Gegenstand der Forschung. Die Analyse von Zeitreihen ist nicht auf die Vorhersage von Bewegungen beschränkt. Sie ist überall dort auffindbar, wo das zu untersuchende Problem sich quantisieren lässt und einen zeitlichen Zusammenhang hat, wie z.B. die Verarbeitung und Vorhersage von Sprache, Kursanalysen an der Börse oder der Antizipation der aktuellen Phase im Konjunkturzyklus. Für *adversarial attacks* gilt vergleichbares. Szegedy et. al machten 2014 auf *blind spots* in *Convolutional Neural Networks* im *Computer Vision* Bereich aufmerksam [42]. Seit dem wurden und werden Angriffe auf unterschiedliche Domänen und Architekturen adaptiert [20]. Dieses Kapitel liefert einen Überblick über diese beiden Domänen, in denen die Arbeit eingebettet ist.

3.1 Vorhersage der Trajektorie

Klassische Ansätze versuchen das Problem der Trajektorievorhersage mittels statistischer Verfahren, wie dem Kalman Filter [13], Regressionsanalyse [7] und diversen Physik basierten Modellen [28] zu lösen. Durch die Integration von sozialen Features, wie z.B. der dynamische Bezug von Personen und Fahrzeugen untereinander, als auch statischen Features, wie z.B. der Fahrbahngeometrie, steigt die Komplexität klassischer Lösungsansätze stark an. Mittels *Deep Learning* Ansätzen kann die Prognose der Bewegung als eine multimodale Wahrscheinlichkeitsverteilung analysiert werden, ohne die Komplexität der Situation im Detail modellieren zu müssen [12], [27], [31], [19].

State-of-the-Art Verfahren versuchen das Problem der Trajektorie Vorhersage mittels Conditional Variational Autoencodern (CVAE) auf Basis der rohen Sensordaten zu lösen [27], [37]. Die Netzwerke lernen die Verteilung der Trainingsdaten und können so, bedingt auf einer Eingabe (*condition*), neue Daten generieren. Diese Art von Modellen benötigt große Mengen an Trainingsdaten, um die Verteilung adäquat widerzuspiegeln. Alahi et al. zeigt, dass die Bewegungen von Fußgängern besser vorhergesagt werden können, wenn ihr

sozialer Kontext mit einbezogen wird [1]. Jeder Fußgänger wird als *LSTM* rerpräsentiert. Treten mehrere Fußgänger in eine unmittelbare Nachbarschaft zueinander ein, so wird der soziale Kontext mit Hilfe von *Social Pooling* aggregiert. Der resultierende *Hidden State* des LSTM-Modells wird anschließend zur Vorhersage benutzt. *Seq2Seq* Modelle, die ihren Ursprung im *Natural Language Processing* haben, sind auf Grund ihrer Architektur dafür geeignet beliebige Ausgabeströme zu generieren [41]. Damit sind sie in der Lage Trajektorien für beliebig lange Zeiträume vorherzusagen, selbst wenn der beobachtete Zeitraum variabel ist. Dabei wird in einem *Encoder* Modell eine *hidden-representation* generiert die die Beobachtung aggregiert, auf der das *Decoder* Modell die Vorhersage anstellt. [16] greift die Idee von *Social-Pooling* auf und bildet soziale Interaktionen zwischen Fußgängern mittels eines LSTM *Seq2Seq* Netzwerkes nach, welches die *hidden representation* einer Pooling-Operation unterzieht. Deo et al. überführt diesen Ansatz zur Trajektorievorhersage von Fahrzeugen, indem die Social-Pooling Layer um einige Convolutional Layer erweitert werden [12]. Es hat sich gezeigt, dass neben der Beschaffenheit der Verkehrswege auch die sozialen Interkationen zwischen einzelnen Verkehrsteilnehmer eine wichtige Rolle bei der Vorhersage spielen. Li et al. geht einen anderen Weg als bei Social-Pooling und stellt Nachbarschaftsbeziehung zwischen Verkehrsteilnehmer über Kanten in einem Graphen dar [29]. Inspiriert durch GNN aggregieren Sie den räumliche Beziehung benachbarter Verkehrsteilnehmer und erweitern diesen Ansatz um eine zeitliche Komponente, indem sie, wie in [50], eine Convolution über die Zeit durchführen. Dieser Ansatz erreicht annähernd State-of-the-Art Ergebnisse, verwendet jedoch in der Dimension stark reduzierte Eingabedaten, was diese Vorgehen für Embedded Systems interessant macht. Aktuelle Ansätze erweitern die Feature Dimension, die oft aus der Position innerhalb der Szene besteht, um einen Szenenrelevanten Kontext. Chang et al. stellt einen Datensatz vor, welcher neben Positionsdaten sogenannte *High Definition Maps* bereitstellt, die vektorisierte Straßenelemente beinhalten [7]. VectorNet nutzt diese Informationen und bettet die vektorisierte Karte in Graphen ein, die den lokalen Kontext einer Verkehrsszene kodieren und erreicht *State-of-the-Art* Vorhersage im Bereich der Trajektorievorhersage mittels Graph Neural Networks [14].

3.2 Angriffe auf Graph Neural Networks

Szegedy et al. zeigte, dass *Deep Neural Networks*, im Bereich der Objekterkennung, sich bereits bei leichten Veränderungen der Eingabe täuschen lassen [42]. Die Aufdeckung von *blind spots* führte zur Erforschung diverse Methoden, um neuronale Netze zu täuschen und vor *adversarial attacks* zu schützen [15], [20]. Huang et al. liefern eine ausführliche Einführung über die Grundlagen von *adversarial attacks* und *robustness* für Neural Networks [20]. Dabei sind nicht nur spezifische Anwendungsdomänen oder Modelle anfällig für *adversarial attacks*. Tramer et al. zeigen, dass Angriffe und bereits manipulierte Eingaben auf andere Modelle übertragbar sind [43]. Diese Erkenntnisse haben sich auf die Erforschung von GNN's fortgepflanzt. Die Zahl der Veröffentlichungen in Bezug auf *adversarial attacks* auf Graph Neural Networks ist überschaubar. Eine ausführliche Auflistung der aktuellen Forschungsthemen inklusive einer Einführung und Taxonomie in das Thema liefert [40]. Aufgrund der komplexeren Eingabe von *Graph Neural Networks* und ihrer breiten Anwendungsdomäne sind *adversarial attacks* für diese schwieriger zu konzipieren. Die Eingabe besteht aus Knoten Attributen und der Graphstruktur, um GNN zu täuschen sind Angriffe auf beide Eingaben möglich.

Dai et al. untersuchen in ihrer explorativen Arbeit die Effektivität mehrere Angriffsszenarien und konnten zeigen, dass gängige GNN's Architekturen anfällig für diverse *adversarial attacks* sind und manche der gelernten Angriffsmethodiken auf andere Modelle transferierbar sind [9]. Die Methoden gehen davon aus, dass das Modell verfügbar ist, sogenannte *White Box Attack*. Untersucht wurden graientenbasierte, heuristische, zufallsbasierte und *Reinforcement*-Verfahren. Bei allen konnte die Wirksamkeit von *adversarial attacks* auf die Reduzierung der Klassifikationsrate nachgewiesen werden.

Wu et al. bauen auf diesen Erkenntnissen auf und untersucht den Einfluss von *adversarial attacks* für die Aufgabe der *node-classification* unter Einbezug beider Eingaben [48]. Ausgehend vom *white box* Fall generieren sie perturbierte Eingaben mit Hilfe von gradientenbasierten Verfahren, unter anderem *FGSM*.

Zügner et al. führten erstmals ein Framework ein, welches ohne Kentnisse über das zugrundeliegende Modell, sogenannte *Black-Box* Szenarien, *imperceptible adversarial examples* erzeugen kann [53]. Das Modell, **Nettack**, lernt dabei ein *Surrogate-Model* an, welches das original Modell approximiert, und generiert *adversarial examples* auf Grundlage von Graph- und Featurestatistiken.

Viele weitere Attacken, die *Reinforcement Learning*, *genetische*, *gradientbasierte* und andere Techniken verfolgen, haben das Ziel die Fehlklassifikationsrate des Opfers so hoch zu treiben, wie nur möglich, ungeachtet der Tatsache, wie aufwändig es ist den Angriff zu detektieren [40]. Die küzrlich veröffentlichte Arbeit [23] fasst bekannte Verfahren zusammen und kategorisiert sie nach Ausgangslage, Wissen über das zugrunde liegende Modell,

Aufgaben- und Angriffstyp. Keiner der aufgeführten Methoden hat sich bisher mit *adversarial attacks* auf Graphen für die Vorhersage von Zeitreihen auseinandergesetzt.

Kapitel 4

Modell

Die *adversarial attacks* bauen auf selbst erstellten Graph Neural Networks auf. Um die Netzwerke auf die Vorhersage von Fahrzeugtrajektorien in urbanen Umgebungen zu trainieren, benötigen diese einen entsprechenden Datensatz. Dieses Kapitel beschreibt die Struktur des Vorhersage- und Angriffsmodell und motiviert die Verwendung und Vorverarbeitung des *Argoverse*-Datensatz. Der Diskussion über den verwendeten und aufbereiteten Datensatz in Abschnitt 4.1 folgt die Problemformulierung in Abschnitt 4.2. Die Problemformulierung beschreibt, bezugnehmend auf das Grundlagen Kapitel in 2, die mathematischen Konzepte des Vorhersage- und Angriffsmodell. Abschnitt 4.3 beschreibt detailliert alle erstellten Modelle, stellt sie gegenüber und geht auf mögliche Interpretationsansätze von vorhergesagten Trajektorien ein. Das Kapitel schließt mit der Diskussion über alle erstellten Angriffsszenarien und ihre Anwendbarkeit auf Graphstrukturierten Daten in Abschnitt 4.4.

4.1 Datensatz & Vorbereitung

Alahi et al. hat gezeigt, dass durch soziale Informationen die Vorhersage von Trajektorien verbessert werden kann [1]. Hierzu gehören z.B. die Position von benachbarten Verkehrsteilnehmern sein. Wie bereits herausgestellt, wird die Verwendung von Graph Neural Networks dadurch motiviert, dass mit dem *Message Passing* Schritt der soziale Kontext der Nachbarschaft ermittelt werden kann. Die in der Literatur verwendeten Datensätze eignen sich nicht unmittelbar für das überwachte Training eines GNN, da die Daten nicht in einer graphstrukturierten Form vorliegen. Diese müssen zuvor durch einige Vorverarbeitungsschritte in eine geeignete Darstellung überführt werden. Bei der Wahl des Datensatzes zu beachten, dass dieser der Anforderung an eine urbane Umgebung genügen muss. Das heißt, dass er möglichst viele Trainingsdatenpunkte enthalten sollte, die die Diversität an verschiedenen Fahrmanövern in einer Stadt widerspiegelt. In diesem Kapitel soll die Wahl des verwendeten Datensatzes und dessen Aufbereitung für das Modell diskutiert werden.

Timestemp	Object-ID	Object-Type	X	Y

Tabelle 4.1: Schema eines Samples des Argoverse Datensatzes. Jedes Samples enthält für jedes Objekt der Szene 50 Einträge/Zeitschritte.

Zu den am häufigsten zitierten Datensätzen gehören **NGSIM**, **ApolloScape** und **Argoverse**. **N**ext **G**eneration **SIM**umlation (NGSIM) hat es sich zum Ziel gemacht das Fahrverhalten zu analysieren und hat zu diesem Zweck zwei Datensätze veröffentlicht, die über mehrere Minuten die genaue Position von Fahrzeugen auf Autobahnabschnitten enthalten. Die Datensätze heißen NGSIM I-80 und U-101 [34] und sind der Kenncode und entstammen verschiedener Autobahnabschnitte. Beide Datensätze enthalten auf einer Länge von 500 bzw. 640 Metern die genaue Position von jedem Fahrzeug. Der gemessene Zeitraum erstreckt sich auf 45 Minuten, mit einer Auflösung von 1/10 Sekunden. Sie enthalten jedoch keine komplexen Fahrmanöver, wie sie im urbanen Gelände üblich sind, wie z.B. das Abbiegen an Ampeln oder das Warten an Zebrastreifen. Dem gegenüber steht der öffentlich zugängliche Datensatz von ApolloScape, der insgesamt 103 Minuten, 53 Minuten an Trainingssequenzen und 50 Minuten an Testsequenzen, an Kamera und LiDAR Material aus Beijing, China, zur Verfügung stellt. Jedes Sample liegt zudem als tabellarisierte Zeitreihe vor, in der jedes Features als in einer separaten Spalte kodiert ist und hat eine Länge von einer Minute, bei einer Auflösung von 1/2 Sekunde. Sie enthalten neben der Position im 3D auch Meta-Informationen zum Fahrzeug, wie z.B. der Ausrichtung, dem Fahrzeugtyp oder der Höhe [21]. Damit repräsentiert der Datensatz von ApolloScape eine urbane Umgebungen für die Fahrzeugtrajektorien vorhergesagt werden können. Jedoch ist davon auszugehen, dass 53 Minuten/Sequenzen aus einer Stadt nicht genügen, um die wahre Verteilung aller möglichen Fahrmanöver zu approximieren. Argoverse stellt umfangreicheres Datenmaterial, speziell für die Aufgabe der Vorhersage von Trajektorien, zur Verfügung. Das Unternehmen hat über 240.000 Sequenzen mit einer Länge von je 5 Sekunden, mit einer Auflösung von 1/10 Sekunden, in den Städten Miami und Pittsburgh mittels Kameras und LiDAR ausgestatteten Fahrzeugen, gesammelt [7]. Karten der abgefahrenen Strecken befinden sich im Anhang A. Die gesammelten Daten wurden in ein tabellarisches Zeitreihenformat konvertiert und enthalten zu jedem aufgenommenen Zeitpunkt, für jeden Verkehrsteilnehmer, die absolute Position, bezogen auf den Kartenursprung, sowie weitere Metadaten. Eine genaue Auflistung aller Attribute befindet sich in Tabelle 4.1.

Object-Type enthält die Typen **AV**, **Agent** und **Others**. AV steht für Autonomous Vehicle, das Fahrzeug, welches die Daten aufgezeichnet hat. Der Agent ist der Teilnehmer, dessen Positionen für die nächsten Zeitschritte vorhergesagt werden soll. Others sind alle anderen Verkehrsteilnehmer. Es ist nicht bekannt, ob es sich um Passanten, Fahrradfahrer oder andere Fahrzeuge handelt. **X, Y** sind die absoluten Koordinanten bezogen auf den Kartenursprung und haben die Einheit Meter. Das Weglassen der Tiefenkoordinate führt

zur sogenannten **Bird's Eye View** (BEV). Die Autoren begründen die Verwendung der BEV damit, dass sie zu weniger komplexen Modellen bei ähnlicher Qualität führen soll [7]. Der Datensatz ist in ca. 204.000 Trainings- und 44.000 Testsequenzen aufgeteilt, wobei jede Sequenz für zwei Sekunden die volle Trajektorie aller aufgenommenen Verkehrsteilnehmer enthält und die letzten drei Sekunden lediglich die Positionen des **Agenten**. Abbildung 4.1 zeigt die BEV für die ersten zwei Sekunden einer Sequenz. Die Trajektorien *OTHERS* Verkehrsteilnehmer wurden fixiert.

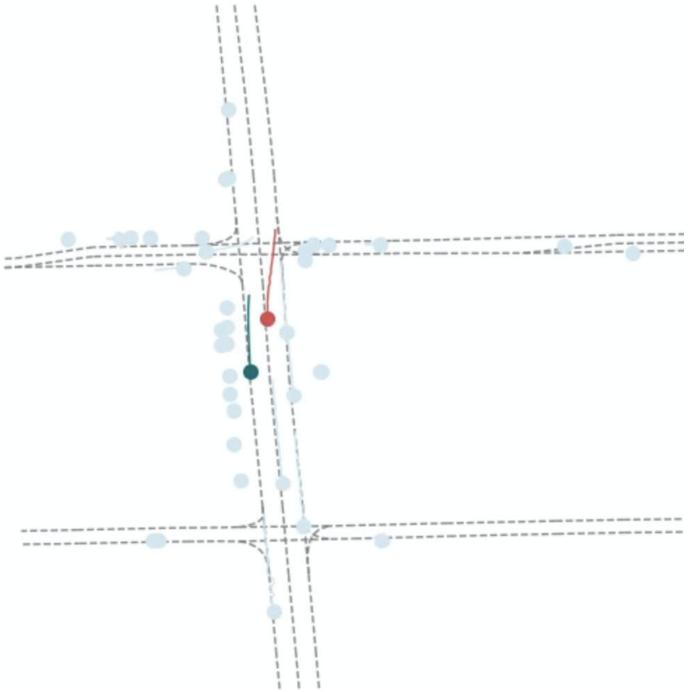


Abbildung 4.1: Bird's Eye View auf einer Trainingssequenz für die ersten zwei Sekunden. Der Bewegungsverlauf des AV ist in grün dargestellt. Die Trajektorie des Agenten ist rot.

Der Argoverse-Datensatz enthält zusätzlich sogenannte **H**igh **D**efinition Maps, die für die abgefahrenen Städte alle Straßensegmente als Vektoren enthält. Jede Straße ist mit ihren Mittellinien und Straßenberandungen in mehrere Abschnitte mit einzigartiger ID abgespeichert. Argoverse bietet eine Schnittstelle an, um die vektorisierten Segmente als Polygone zu rendern, wodurch vorhergesagte Trajektorien im Kontext der gesamten Szene visuell bewertet werden können. Daneben wird eine Schnittstelle angeboten, mit deren Hilfe es möglich ist, die nächstgelegene Mittellinie zu einem Fahrzeug zu extrahieren. Dies kann genutzt werden, um straßenabhängige Features vorzubereiten, die die Performance des Modells verbessern. Verfügt das gesamte aufgenommene Straßennetz über Mittellinien, so ist es wahrscheinlich, dass sich der *Agent* durchgehend in der Nähe einer Fahrbahnmarkierung aufhält. Auf diese Weise lässt sich neben dem sozialen Kontext, also umgebende Verkehrsteilnehmer, das Konzept eines regelbasierten Straßensystems einführen.

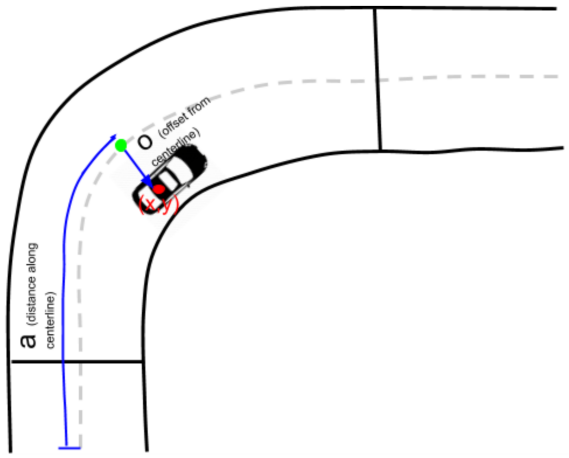
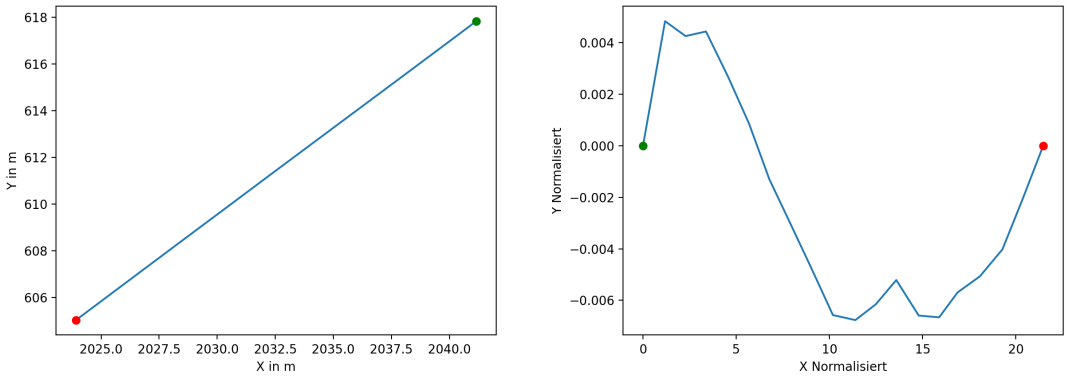


Abbildung 4.2: Die absolute Position eines Fahrzeugs lässt sich in zwei straßenabhängige Features konvertieren. Jedes Straßensegment verfügt über eine Mittellinie. Zu jedem Zeitschritt wird die bereits zurückgelegte Strecke an einer Mittellinie, sowie der aktuelle Abstand zu dieser, aufgezeichnet.

Abbildung 4.2 enthält eine visuelle Beschreibung der Map Features. Zu jedem Zeitschritt lässt sich die aktuelle Position des Verkehrsteilnehmers in eine relative straßenabhängige Koordinate konvertieren. Dazu wird beginnend ab dem ersten Zeitschritt die zurückgelegte Strecke zur nächsten Mittellinie aufsummiert (**Distance Along Centerline**), sowie den Abstand zu dieser (**Offset From Centerline**).

Neben den *Map Features* lassen sich aus der Zeitreihenaufzeichnung soziale und normalisierte Positionsmerkmale extrahieren. Bevor diese Attribute berechnet werden, werden alle Teilnehmer aus der Szene entfernt, deren Anwesenheit in den ersten zwei Sekunden unter 65% liegt. Da der AV und Agent der Mittelpunkt der Sequenz sind, liegt ihre Anwesenheit für jedes Sample bei 100%. Für alle anderen Teilnehmer gilt dies nicht unbedingt, da sie je nach Aufnahmebedingung nur kurz in die Szene eintreten und wieder aus dem Aufzeichnungsradius verschwinden können. Um Lücken in der Zeitreihe zu schließen, werden fehlende Position mit der *Nearest Neighbor* Interpolation berechnet [33]. Definiert jede Zeile einer Matrix einen Zeitschritt und ihre Spalte die zugehörigen Attribute, so wird eine leere Zeile mit einer Kopie der zeitlich nächsten, nicht leeren, Zeile befüllt.
Auf Basis der reduzierten Teilnehmermenge wird für jeden Zeitschritt und jeden Teilnehmer der Abstand zum nächsten Vorder- (*Distance Front*) bzw. Hintermann (*Distance Back*)berechnet. Dies ist notwendig, da die Abstandsmatrix für jeden der V Teilnehmer zu einer Matrix mit $\frac{V(V-1)}{2}$ relevanten Einträgen führen würde, die sich bei Samples mit unterschiedlicher Teilnehmerzahl V nicht trivial zu einem Batch zusammenfassen lässt.
Für die in X und Y Koordinaten kodierte Trajektorie hat sich gezeigt, dass das Modell einen stabileren Lernprozess einnimmt, wenn diese orts- und richtungsunabhängig sind.



(a) **(b)**

Abbildung 4.3: (a) Die unnormalisierte Trajektorie. Der grüne Punkt markiert den Startpunkt, der rote Punkt das Ende der aufgezeichneten Trajektorie. Das Fahrzeug fährt in einer geraden Linie von Nord-Ost nach Süd-West. **(b)**: Die normalisierte Trajektorie. Die Bewegung ist nun Richtungsunabhängig. Die Trajektorie wurde auf den Ursprung translatiert und so rotiert, dass die Bewegung auf der X-Achse endet.

Grafik 4.3a zeigt den Bewegungsverlauf für einen Verkehrsteilnehmer mit absoluten Koordinaten in Metern. Da sich die Position auf den Kartenursprung bezieht, lässt sich sowohl der genaue Ort innerhalb der Karte, als auch die geographische Richtung bestimmen. Indem die Trajektorie auf den Koordinantenursprung bei $(0, 0)$ projiziert wird und so rotiert wird, dass die Bewegung wieder auf der X-Achse endet, kann der Verlauf relativiert werden. Die normalisierte Bewegung aus Abbildung 4.3b kann mit Hilfe der gespeicherten Translationsmatrix und Rotationswinkel zurücktransformiert werden. Durch die Rotation parallel zur X-Achse nimmt die Varianz auf der Y-Achs stark ab. Die abgebildete Trajektorie ist dieselbe *gerade* Bewegung, wir vor der transformation, aus einem näheren Blickwinkel.
Da sowohl die Map Features, als auch die normalisierte Bewegung, den Fahrtverlauf widerspiegeln, kann das Modell darauf trainiert werden sowohl die Bewegung relativ zu Mittellinie vorherzusagen, als auch die normalisierte Trajektorie. In beiden Fällen muss die Zielvariable mit denselben Methoden konvertiert werden, damit sie mit der Vorhersage verglichen werden kann.
Zusammengenommen verfügt jeder Teilnehmer über sechs Merkmale verteilt über 20 Zeitschritte, die in Tabelle 4.2 zusammengefasst sind. Verfügt ein Sample über V Teilnehmer, die über einen Zeitraum von $T = 20$, mit $\{t = 1, ..., T\}$, Zeitschritten beobachtet wurden, dann lässt sich die Eingabe für das Modell als Feature Matrix der Dimension $X \in \mathbb{R}^{V \times T \times C}$ mit $C \in \mathbb{R}^6$ schreiben.

Um die Daten von einem Graph Neural Network verarbeiten zu können, muss die räumliche Struktur der Teilnehmer durch eine Adjazenzmatrix gespeichert werden. Im einfachsten Falle ist dies eine Matrix, die in allen Zeilen und Spalten eine 1 enthält. Dies

$X^{20\times 6}$

Dist. Centerline	Offset Centerline	Dist. Front	Dist. Back	X-Norm	Y-Norm

Tabelle 4.2: Aufbau der Feature Matrix für einen Verkehrsteilnehmer. Eine Zeile entspricht einem Zeitschritt.

würde dazu führen, dass jeder mit jedem verbunden ist und der Vorteile des GNN soziale Features durch die Graphstruktur zu extrahieren nicht genutzt wird. Daher werden aus dem Datensatz in der Zeit dynamische Graphen A_t erstellt, die reale Situationen simulieren. Dabei wird angenommen, dass jeder Verkehrsteilnehmer einen Sichtradius von 20 Metern hat und jeder andere Teilnehmer in diesem Radius für sie von Relevanz ist. Werden relevante Verkehrsteilnehmer in einem Zeitschritt t gefunden, so wird der Teilnehmer i mit seinem Nachbarn $j \in \mathcal{N}_i$ in der entsprechenden Zeile/Spalte in A_t mit einer 1 kodiert. Dabei ist die Nachbarschaft eines Knotens j definiert als $N(j) = \{u, v \in V | A_{u,v} = 1\}$. Diese Konstruktion führt zu einem ungerichteten Graphen, da die Eigenschaft des Sichtradius symmetrisch ist. Sieht Teilnehmer j Teilnehmer i, dann gilt dies auch andersherum, dabei sind Sichtbeschränkungen durch Hindernisse ausgenommen. Abbildung 4.4 zeigt die Visualisierung einer Vorhersage mit eingezeichneten Observationsradius von 20 Metern. Für jeden Teilnehmer, der in einem Schritt innerhalb des Radius liegt, wird ein Eintrag in der entsprechenden Adjazenzmatrix hinterlegt.

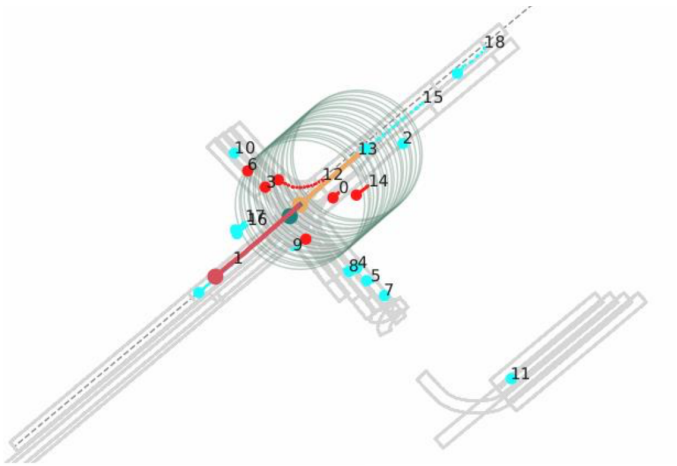


Abbildung 4.4: Visualisierung einer Vorhersage: Observationsradius (grün) des Agenten für 20 Zeitschritte. Der Radius beträgt 20 Meter. Verkehrsteilnehmer, die zum gemessenen Zeitpunkt innerhalb des Sichtradius waren, sind mit einem roten Punkt markiert. Die beobachtete Trajektorie des Agenten ist orange, die Vorhersage in grün und die wahre Bewegung rot.

Durch den Einsatz von mehreren hintereinander geschalteten Graph Convolutional Layern kann ein Teilnehmer den Kontext über seine direkten Nachbarn hinaus ermitteln.

$$A = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_T \end{bmatrix} \tag{4.1}$$

Für ein Sample mit V Verkehrsteilnehmer über T Zeitschritte hat A die Dimension $A \in \mathbb{R}^{VT \times VT}$, wobei für jeden Subgraph $A_t \in \mathbb{R}^{V \times V}$ gilt.

Da jedes Sample über eine unterschiedliche Anzahl an Verkehrsteilnehmern V verfügt, können mehrere Featurematrizen $X \in \mathbb{R}^{V \times T \times C}$ zusammengefasst werden, indem die maximale Anzahl an V in einem Batch bestimmt wird. Werden die Featurematrizen der Beispiele, die weniger als V Verkehrsteilnehmer aufweisen mit *Null*-Einträgen aufgefüllt (**gepadded**), sodass sie auf dieselbe Dimension kommen wie das größte Sample, so können sie zu einem Batch der Größe $X \in \mathbb{R}^{N \times V \times T \times C}$, mit N als Batchgröße, zusammengefasst werden. Für die Adjazenzmatrizen gilt ebenfalls, dass jeder Subgraph $A_{nt}, n = 1, ..., N$ ebenfalls um die ergänzten Verkehrsteilnehmer gepadded werden muss. Die resultierenden Adjazenzmatrizen können dann wie zuvor gestacked werden:

$$A_B = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_N \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{1T} \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} A_{N1} & & \\ & \ddots & \\ & & A_{NT} \end{bmatrix} \end{bmatrix} \tag{4.2}$$

Dadurch steigt der Rechenaufwand, da die gepaddeten Einträge nicht von der Matrixmultiplikation AX ausgeschlossen werden können, allerdings beeinflussen die neuen Werte nicht das Ergebnis einer Graph Convolution, da kein ursprünglicher Knoten eine Kante zu den ergänzten Einträgen hat.

4.2 Problemformulierung

Die Aufgabe des Trajektorie Vorhersage-Modells besteht darin, die Trajektorie des Agenten für $T_{Pred} = 30$ Zeitschritte, mit $\{t_{Pred} = T_{+1}, ..., T_{Pred}\}$, vorherzusagen. In diesem Fall muss die Zielvariable $Y_i = \left(x_i^{t_{Pred}}, y_i^{t_{Pred}}\right)$ ausschließlich die *Ground Truth* des Agenten i enthalten. Diese kann, wie bereits erwähnt, durch Normalisierung der gewünschten Zielgröße (Map Features oder normalisierte Trajektorie) erhalten werden. Um diese Aufgabe durchzuführen benötigt das Modell als Eingabe die Graphstruktur A und die Knotenattribute X. Wie in Kapitel 4.1 beschrieben, können diese zu Batches der Größe N mit V Verkehrsteilnehmern zusammengefasst werden. Dabei kann die Adjazenzmatrix in A_t, mit $t \in T = \{1, ..., 20\}$, Subgraphen unterteilt und jeder Verkehrsteilnehmer enthält für jeden Zeitschritt t eigene Features C. Die Dimensionen der Adjazenzmatrizen und der Features bleibt, wie in Kapitel 4.1 beschrieben, bestehen, sodass gilt: $A \in \mathbb{N}^{VT \times VT}$ und $X \in \mathbb{R}^{N \times V \times T \times C}$.
Um *higher-level* Features auf Basis eines größeren Kontextes zu gewinnen, besteht die Eingabe aus allen vorbereiteten Features, die zusammen die Historie der beobachteten Szene für $T = 20$ Zeitschritte bilden:

$$X = \left[x^0, ..., x^{t-1}, x^t\right] \tag{4.3}$$

wobei,

$$x^t = \begin{bmatrix} d_0^t & o_0^t & f_0^t & b_0^t & x_0^t & y_0^t \\ & & & \vdots & & \\ d_V^t & o_V^t & f_V^t & b_V^t & x_V^t & y_V^t \end{bmatrix} \tag{4.4}$$

die Distanz d entlang der Mittellinie, Offset o von dieser, Abstand zum vorderen Vehicle f, Abstand zum hinteren Vehicle b, normalisierte x- und y- Koordinate für alle Verkehrsteilnehmer V zum Zeitpunkt t der Szene sind. Neben den Features benötigt das Modell die Graphstruktur, die vorgibt welche Knoten im *Message Passing*-Schritt miteinander aggregiert werden sollen. Die zum *batching* in Kapitel 4.1 vorbereitete Matrix A verbindet alle Verkehrsteilnehmer, die sich zueinander in einem Sichtradius von 20 Metern aufhalten. Da sie die räumliche Beziehung unter allen Teilnehmern repräsentiert, wird sie im folgenden $A_{spatial}$ genannt.

Inspiriert durch [50], [29] wurde das Modell um die Fähigkeit erweitert Merkmale über die zeitliche Dimension zu aggregieren. Wie in Abbildung 4.5 dargestellt, wird die Szene aus einer Menge von dynamischen Graphen $|A_{spatial}| = |T| = 20$ dargestellt, die alle Verkehrsteilnehmer in einen räumlichen Bezug zueinander stellt.

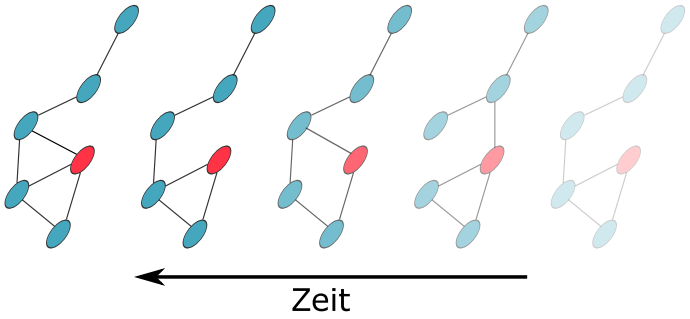


Abbildung 4.5: Der dynamische Graph in der Zeit dargestellt. Objekte können den Sichtradius eines anderen Teilnehmers betreten oder verlassen. Die Anzahl an Teilnehmer bleibt für eine Szene immer gleich. Zeitlich vergangene Graphen sind verblasster dargestellt.

Dabei kann es vorkommen, dass Objekte den Sichtradius betreten oder verlassen. Indem Kandidat i mit sich selbst in der Zeit t über eine Kante verbunden wird, können, wie in Grafik 4.6 skizziert, **Temporal Graph Convolutions** durchgeführt werden.

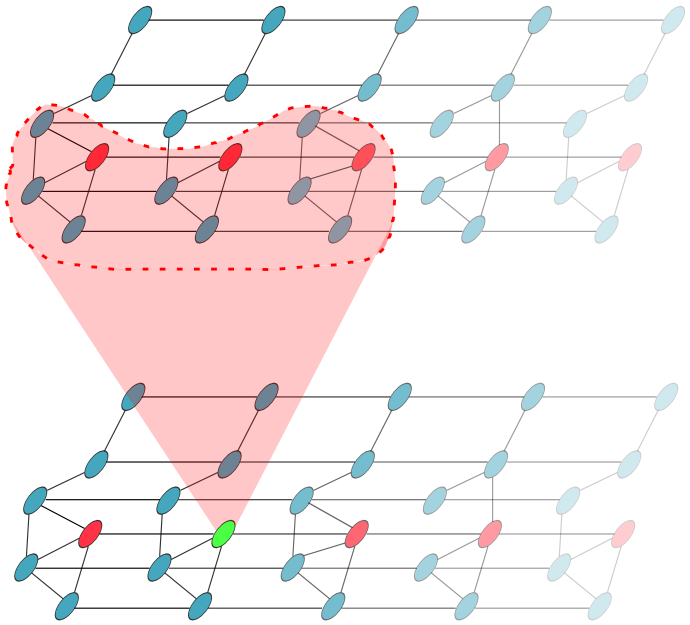


Abbildung 4.6: Eine Graph Convolution wird über ihre Adjazenzmatrix bestimmt. Temporale Adjazenzmatrizen verbinden Objekte in der Zeit mit sich selbst (horizontalen Kanten). Bei der temporalen Convolution werden die zeitlichen Nachbarn (rote Knoten) im grünen Knoten aggregiert. Die Graphstruktur bleibt von dieser Operation unberührt. Sie wirkt nur auf den Knotenattributen.

Eine *Temporal Graph Convolution* nimmt die zeitlichen Nachbarn eines Knotens und führt eine Graph Convolution, wie in Kapitel 2.1.1 beschrieben durch. Das heißt, die ursprünglichen definierten Beziehungen aus $A_{spatial}$ werden ignoriert und es werden neue Kanten zwischen denselben Teilnehmer über alle Zeitschritte hinweg definiert. Zu diesem Zweck muss die Adjazenzmatrix so angepasst werden, dass sie Verkehrsteilnehmer in der Zeit, statt im Raum verbindet. In Kapitel 4.1 wurde der dynamische Graph als diagonali-

sierte Adjazenzmatrix definiert, wobei Einträge von Graphen aus unterschiedlichen Zeiten sich nicht überlappen. Werden Brücken zwischen diesen Graphen geschaffen, so können zeitliche Abhängigkeiten in einer **temporalen Adjazenzmatrix** $A_{temporal}$ kodiert werden. Dazu muss für jeden Teilnehmer eine Kante zum selben Teilnehmer in den Nachbargraphen erstellt werden. Alle anderen Verbindungen zwischen Teilnehmern werden fallengelassen. Formal kann dies wie folgt ausgedrückt werden:

$$A_{temporal} = \begin{bmatrix} 1 & \dots & a_{0,V} & 1 & \dots & \\ \vdots & \ddots & & & & \\ a_{V,0} & & 1 & \dots & a_{tV+i,tV+i} & 1 \dots \\ 1 & & \vdots & & & \\ \vdots & & a_{tV+i,tV+i} & & & \\ & & 1 & & & \\ & & \vdots & & & \end{bmatrix} \tag{4.5}$$

Da alle Graphen, mit V Teilnehmern, die Dimension $V \times V$ haben, muss nach V-Schritten für jeden Verkehrsteilnehmer eine Brücke eingebaut werden. Für drei Verkehrsteilnehmer hat die temporale Adjazenzmatrix:

$$A^3_{temporal} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & \\ 0 & 1 & 0 & 0 & 1 & 0 & \\ 0 & 0 & 1 & 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & 1 & 0 & 0 & \\ 0 & 1 & 0 & 0 & 1 & 0 & \\ 0 & 0 & 1 & 0 & 0 & 1 & \\ & & \vdots & & & & \ddots \end{bmatrix} \tag{4.6}$$

sowohl Einträge auf der Hauptdiagonalen, denn jeder Teilnehmer nimmt seinen eigenen Zeitschritt wahr, sowie je eine Verbindung zum zeitlich versetzten *Ich*, welches V Schritte entfernt liegt. Für 3 Kandidaten einer Szene hat die Adjazenzmatrix für einen Zeitschritt t: $A_t = 3 \times 3$ Einträge. Diese Konstruktion einer *temporalen* Adjazenzmatrix wird durch den Batchingprozess, der die Methode des Paddings verwendet, nicht beeinflusst. Werden fehlende Teilnehmer durch leere Einträge zwischen den Zeitschritten aufgefüllt, um auf die maximale Anzahl an Verkehrsteilnehmer eines Batches zu kommen, so befindet sich der Eintrag zum vergangenen und zukünftigen Kandidaten wieder V Schritte entfernt.

Das Ziel der *adversarial attacks* ist es, das trainierte Modell in seiner Vorhersage so gut wie möglich zu behindern. Um alltägliche Fahrsituationen zu simulieren wird bei den

Angriffen der Anspruch an die *imperceptibility* fallengelassen, sodass die *adversarial examples* im wesentlichen *korrupte* Sensoren widerspiegeln. Die Menge der strukturellen Angriffe besteht aus perturbierten Adjazenzmatrizen $\left\{\hat{A}_0, ..., \hat{A}_j\right\}$, die aus einer Eingabe-Adjazenzmatrix A durch die Funktion $\phi(A)$ gewonnen wird. Die Menge der Angriffe auf die Knotenattribute lässt sich analog formulieren, sodass $\left\{\hat{X}_0, ..., \hat{X}_h\right\}$ aus der Eingabe X mittels Φ erzeugt wird. Anlehnend an Kapitel 2.3 lässt sich das Ziel der *adversarial attacks* wie folgt definieren:

$$\underset{\hat{X}_p \in \Phi(\hat{X}_{p-1}), \hat{A}_p \in \phi(\hat{A}_{p-1})}{\arg\max} \sum_{p=1}^{P} L(f(\hat{X}_p, \hat{A}_p, y)$$

Wobei $P \in \mathbb{N}^{\geq 1}$, die Anzahl der möglichen Perturbierungen ist. Diese abgeschwächte Form der Definition 2.3 ist notwendig, da die Maximierung des Fehlerterms für alle möglichen Kombinationen von *adversarial attacks* aus Zeit- und Speicherplatzgründen nicht berechnet werden kann. Insbesondere steigt der Mehraufwand exponentiell mit der Anzahl der möglichen Perturbierungen P. Die abgeschwächte Variante ist ein iterativer Ansatz, die in jedem Perturbierungsschritt p von allen möglichen *adversarial examples* $\hat{X}_p, \hat{A}_p$ diejenige auswählt, die den Fehlerterm maximiert. Auf Basis dieser Auswahl werden die nächsten Perturbierungen berechnete bis in der Summe P Manipulationen an den Merkmalen und der Graphstruktur durchgeführt wurden.
Bei der Vorhersage der Trajektorie handelt es sich um ein Regressionsproblem. Zur Messung der Angriffsqualität kann die Fehlerfunktion des Modells verwendet werden. Eine Alternative bietet die Möglichkeit aus der Trajektorievorhersage ein künstliches Klassifikationsproblem zu machen, indem es nur die zwei Klassen $\{richtig, falsch\}$ gibt. In der Literatur wird dazu die **Miss Classification Rate** definiert, die mit jeder Vorhersage steigt, deren finale Position einen festgelegten Threshold d überschreitet. Der **F**inal **D**isplacement **E**rror (FDE) gibt die Abweichung der Vorhesage von der wahren Trajektorie für den letzten Vorhersagezeitschritt an:

$$FDE_i = \sqrt{(\tilde{x}_i^t - x_i^t)^2 + (\tilde{y}_i^t - y_i^t)^2} \tag{4.7}$$

wobei $\tilde{x}_i^t, \tilde{y}_i^t$ die absoluten Koordinaten des Agenten i im letzten Zeitschritt $t = 30$ sind. Ein Verkehrsteilnehmer gilt dann als fehlklassifiziert, wenn sein FDE um mehr als zwei Meter von der wahren Trajektorie abweicht:

$$MC_i = \begin{cases} 1 & if \quad FDE_i > 2 \\ 0 & otherwise \end{cases} \tag{4.8}$$

Für K Testsamples enthält $MC = \{MC_1, ..., MC_K\} \in \{0, 1\}$ die Menge aller fehlklassifizierten Trajektorien. Daraus lässt sich die **Miss Classification Rate** berechnen:

$$MCR = \frac{1}{K} \sum_{k=1}^{K} MC_k \tag{4.9}$$

Mit Hilfe der *Miss Classifacation Rate* lässt sich der Einfluss von *adversarial attacks* auf unterschiedlichen Modelle bewerten und untereinander vergleichen.

4.3 Trajektorievorhersage

Für das Trajektorie Vorhersage Modell dient *Grip++* als Vorlage [29]. Die Idee der räumlichen und zeitlichen Graph Convolution, sowie die Vorhersage mittels des *Seq2Seq*-Modells wurde übernommen. Das Netz wurde als *supervised node-level* Regressionsmodell konzipiert, sodass der Bewegungsverlauf nur für ein Objekt ermittelt wird. Der schematische Aufbau ist in Abbildung 4.7 dargestellt und gleicht, mit einigen Ausnahmen, dem von *Grip++*.

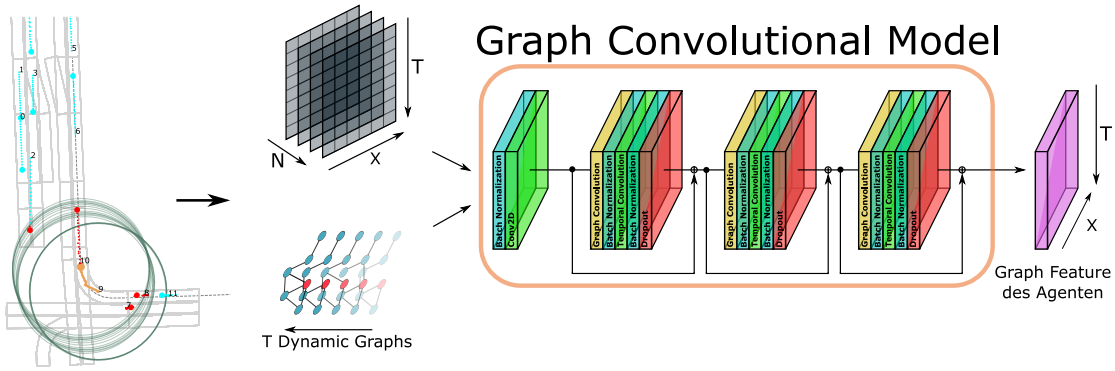


Abbildung 4.7: Die Architektur des adaptierten Modells [29].

Beginnend mit den vorbereiteten Sequenzen von *Argoverse* werden statt einem statischen Graphen, wie in der Vorlage, die über die Zeit dynamischen Graphen $A_{spatial}$, sowie die extrahierten Knotenmerkmalen X in das Model gespeist. Um das Training des Modells zu stabilisieren und zu beschleunigen befindet sich zwischen jedem Convolutional Layer ein **Batch Normalization** (BN) Layer [22]. Dadurch, dass die Eingabe zwischen jeder Schicht standardisiert wird, d.h.

$$\hat{X} = \frac{X - \mu_X}{\sqrt{\sigma_X^2}}\gamma + \beta \tag{4.10}$$

mit Mittelwert μ_X und Varianz σ_X^2, kann ein Mini-Batch in die Form einer Standardnormalverteilung gebracht werden. Durch die zwei zusätzlich zu lernenden Parameter γ, β ist es dem Modell möglich, die Verteilung der Eingabedaten anzupassen, sodass ein stabiler Lernprozess eingenommen werden kann. Die konkrete Diskussion über die Wirkungsweise von *Batch Normalization* ist aktueller Gegenstand der Forschung [3]. Darüber hinaus befindet sich zwischen jedem Layer die nicht lineare Aktivierungsfunktion **Rectified Linear Unit**, die die Eingabe auf $f(x) = \max(0, x)$ abbildet und die Gefahr von Vanishing Gradients reduziert [36]. Um die Gefahr von *Overfitting*, d.h. dem Auswendiglernen der Trainingssamples, zu minimieren befindet sich hinter jedem Graph Convoluional Layer und dem dem Decoder-Part im *Seq2Seq*-Modell ein **Dropout** Layer, der zufällig während der

Trainingsphase einen festgelegten Prozentsatz an Neuronen ausschaltet, bzw. auf 0 setzt. Während der Testphase werden die *Dropout* Layer deaktiviert. Dies sorgt dafür, dass das Modell lernt sich auf *wichtige* Features zu konzentrieren, die das Konzept der Lernaufgabe vermitteln [18]. Nach dem ersten BN-Layer durchlaufen die Daten zunächst eine Faltungsoperation, die im 2-dimensionalen, mit einem (1×1) Kernel, die 6-dimensionale Eingabe auf die Dimension 64 streckt. Dafür müssen die Features in der Form $X^{N \times C \times T \times V}$ vorliegen.

$$Out_{t,v} = \sum_{c=1}^{C} \theta_c * X_{c,t,v} \tag{4.11}$$

mit den Zeitschritt t des Verkehrsteilnehmers v. Dadurch werden für jedes Sample im Batch mit Hilfe von $\theta \in \mathbb{R}^{64 \times 6}$ lernenden Parametern 64 *Feature Maps* aus der 6-dimensionalen Eingabe erzeugt. Ab diesem Punkt durchlaufen die *higher level* Features drei aufeinanderfolgende Graph Convolutional Layer, in der sich die *temporal* und *spatial* Graph Convoltuion abwechseln. Kein Graph-Layer teilt sich die Gewichte mit anderen Layern oder verändert die Dimension der Attribute. Jede Convolution verfügt über anpassbare Parameter der Größe $\theta_{graph} = (64 \times 64)$, die die Eingabedimension auf die gleiche Ausgabedimension abbilden. Um das Training weiter zu stabilisieren befinden sich zwischen den drei zusammengefassten Graph Convolutional Schichten **residuale** Verbindungen. In dieser wird die unveränderte Eingabe X aus dem Schritt vor der Graph Operation mit mit den manipulierten Merkmalen $\hat{X}$ addiert. Dadurch ist es möglich ein *backpropagation* Signal bis in die vordersten Schichten des Modells zu leiten [17]. Nachdem die räumliche und zeitliche Graph Convolution drei mal in Folge angewandt wurde enthalten die Graph Features die Attribute aller Verkehrsteilnehmer. Aus dieser wird für jedes Sample aus dem Batch der Agent extrahiert, dessen Position zuvor gespeichert worden sein muss. Nach der Extraktion haben die Graph Features die Dimension $(N, V = 1, T = 20, C = 64)$, die gemeinsam mit der letzten normalisierten Position $L = (N, Pos = 2)$ des Agenten als Eingabe für das *Seq2Seq* Modell dient. Dabei enthält Pos die normalisierten (x, y)-Koordinaten des Agenten zum letzten beobachteten Zeitschritt. Das in Abbildung 4.8 schematisch dargestellte Modell nimmt die Graph Features und leitet diese durch den *Encoder* Part, der die Eingabedimension beibehält. Dort wird die $T = 20$ Zeitschritte lange Ausgabe der Graph Convolution sequentiell durch ein 3-Schichtes GRU-Netzwerk geleitet, wobei sich die GRU-Elemente die Gewichte teilen, um eine *hidden representation* der Zeitreihe zu erstellen. Die *hidden representation* wird, wie in Grafik 4.8 über rote Pfeile schematisch dargestellt, in jedem Zeitschritt von Einheit zu Einheit weitergereicht und nimmt dabei immer mehr Informationen von aktuelleren Zeitschritten auf. Das GRU-Netzwerk benötigt eine initiale *hidden representation* als Eingabe, zu diesem Zweck wird ein Tensor der Dimension $(Layer = 3, N, C = 64)$ mit 0 Einträgen erzeugt.

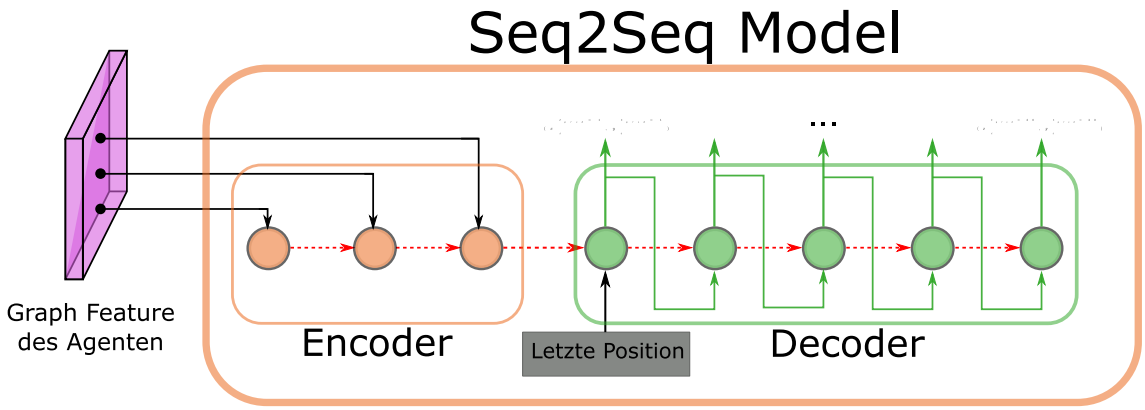


Abbildung 4.8: Seq2Seq2 Modell zur Vorhersage der Trajektorie.

Die erzeugten *latenten* Variablen des Encoders enthalten die aggregierten Zeitschritte in der Form von $hidden = (Layer = 3, C = 64)$, die gemeinsam mit den letzten Positionen der Agenten, in das **Decoder**-Netzwerk des *Seq2Seq* Modells fließen. Um variable Vorhersagelängen zu erzeugen und durch die Tatsache, dass sich die Elemente des GRU Netzwerkes die anpassbaren Gewichte teilen, kann die Ausgabe eines vorherigen Vorhersageschrittes, **regressiv** in einer Schleife für die Vorhersage des nächsten Schrittes herangezogen werden. Um das Training zu stabilisieren, wird das *Seq2Seq* Modell nicht darauf trainiert jede Position für jeden Zeitschritt deckungsgleich mit der *Ground Truth* vorherzusagen. Da dieser Wert in der Regel kontinuierlich wächst, kann es zu Instabilitäten im Lernprozess kommen. Daher wird in jeder Iteration während des *decodings* die Differenz $\Delta x, \Delta y$ zwischen zwei Zeitschritten berechnet. Nachdem die gewünschte Anzahl an Zeitschritten kalkuliert wurde, wird die Ausgabe mit der wahren Trajektorie verglichen, um einen Fehlerterm zu berechnen mit dessen Hilfe im *Backpropagation*-Schritt die Gewichte des Modells angepasst werden können [2]. Für Zeitreihenmodelle, die den Bewegungsverlauf vorhersagen, wird der **Root Mean Squared Error** als Fehlerfunktion verwendet da dieser, aufgrund der Quadrierung in der Summe, größere Abweichungen von der Vorhersage stärker bestraft als kleinere. Gleichzeitig ist das Ziel dieser Funktion den mittleren Fehler für alle Vorhersageschritte klein zu halten.

$$RMSE = L = \sqrt{\sum_{n=1}^{N} \sum_{t=1}^{T_{pred}} \frac{(\tilde{x}_n^t - x_n^t)^2 + (\tilde{y}_n^t - y_n^t)^2}{N T_{Pred}}} \tag{4.12}$$

Beim Training für die volle Vorhersagelänge von $T_{Pred} = 30$ generalisiert das Modell schlecht und lernt im Wesentlichen den Trainingsdatensatz auswendig. Es wurde oft beobachtet, dass das Modell zunächst in die entgegengesetzte Richtung vorhersagte. Dadurch, dass die Ausgabelänge des *Decoders* angepasst werden kann, lässt sich das Modell für unterschiedliche Zeitschritte trainieren. Auf diese Weise kann zunächst darauf trainiert werden die Position nach einem Zeitschritt vorherzusagen und wenn dies gut genug ist für den

nächsten Zeitschritt. Dieser Prozess wird **Curriculum Learning** genannt und es wurde gezeigt, dass diese Trainingstechnik einen positiven Einfluss auf die Konvergenz des Trainings und dessen Qualität hat [4]. Gute Ergebnisse wurden erzielt, wenn das Lernverfahren darauf ausgerichtet ist in den Etappen für $T_{Pred} = [1, 5, 10, 15, 20, 25, 30]$ zu lernen.
Der Trainingsprozess lässt sich wie folgt zusammenfassen: Es wurden insgesamt *vier* Modelle trainiert. Ein *Seq2Seq* Modell ohne Graph Convolutional Layer, eine Variante des Modells mit GCN, wie in Kapitel 2.1.1 beschrieben, sowie zwei Modelle mit Graph Attention Layer, mit 2 bzw. 4 Attention-Heads, wie in Kapitel 2.1.2 beschrieben. Nach jeder Trainingsiteration folgte ein Durchlauf durch den Testdatensatz ohne die Parameter anzupassen. Dieser Schritt dient dazu die Qualität der Vorhersage zu validieren. Nimmt die Performance drei mal in Folge ab, so wird mit der nächsten Vorhersagelänge im *Curriculum Learning* fortgefahren. Alle Modelle wurden auf einem Server mit der folgenden Hardwarespezifikation trainiert:

- CPU: AMD Ryzen 7 2700X Eight Core Processor
- GPU: Geforce RTX 3090
- RAM: 64GiB

Das grundlegende neuronale Netz wurde mit *Pytorch 1.7.1* implementiert. Für die Operationen auf Graphen wurde der Pytorch-Ableger *torch-geometric 1.6.3* genommen. Für das Training mit Hilfe von Grafikkarten wurde die folgende Konfiguration eingesetzt:

- CUDA 11.2
- CUDA-Toolkit 11.0
- Python 3.8.5

Tabelle 4.3 listet die *Hyperparameter* und Komplexität der trainierten Modelle auf. Aus Gründen der Vergleichbarkeit wurde darauf geachtet, dass alle Modelle mit ähnlichen Hyperparametern trainiert werden. Für die Vorhersage der zukünftigen Bewegung hat sich gezeigt, dass bessere Ergebnisse für höhere *Batchgrößen* erzielt werden können.

Da die verwendete Grafikkarte nicht über genug Speicherkapazität verfügt, um ein Modell mit 4- *Graph Attention Heads* der Komplexität von 202.250 Parametern mit einer Batchgrößen von 1024 zu trainieren, musste auf eine Batchgröße von 512 pro Trainingsiteration ausgewichen werden.

Modell	Batch-Size	Optimizer	Learning-Rate	Dropout [%]	Komplexität
Seq2Seq	1024	Adam	0.001	0.6	138.314
GCN	1024	Adam	0.001	0.6	164.042
GAT-2	1024	Adam	0.001	0.6	176.906
GAT-4	512	Adam	0.001	0.6	202.250

Tabelle 4.3: Hyperparameter und Meta-Informationen. Die Komplexität repräsentiert die Anzahl lernbarer Gewichte.

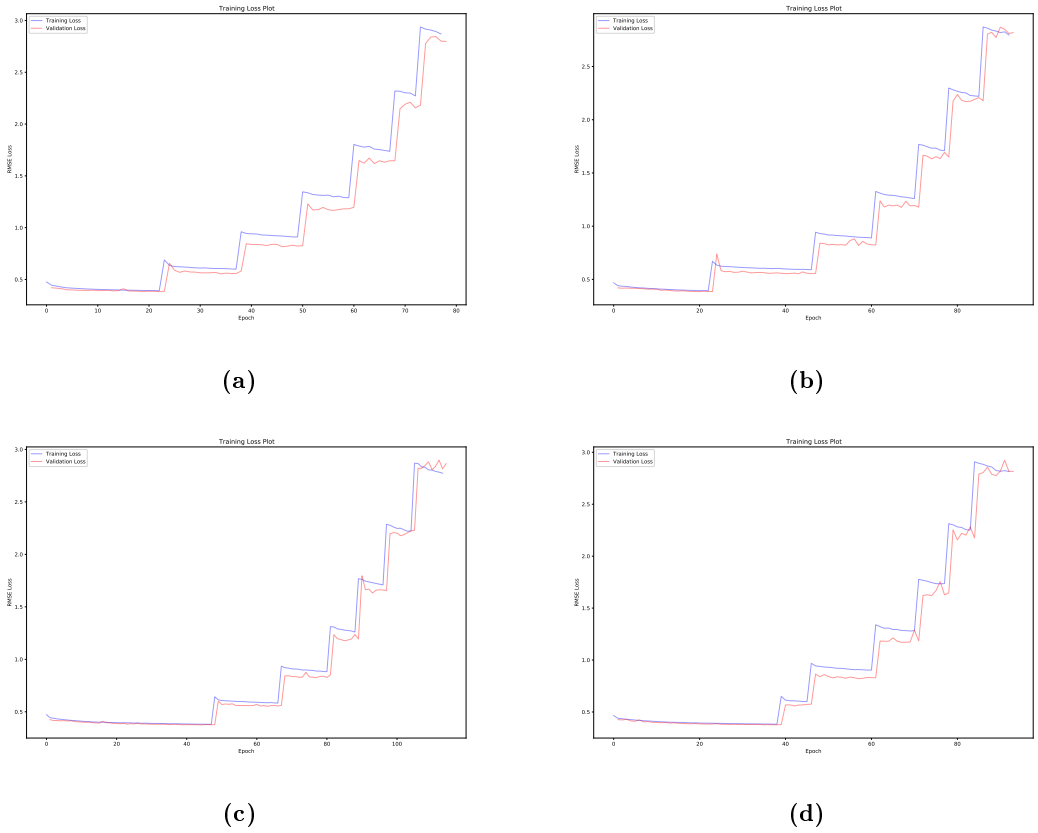


Abbildung 4.9: Verlauf des Trainings für **(a)** Sequence to Sequence Modell, **(b)** Graph Convolutional Modell, **(c)** Graph Attention Modell mit 2 Attention Heads, **(d)** Graph Attention Modell mit 2 Attention Heads

Alle Modelle trainieren verhältnismäßig viele Epochen darauf den ersten Zeitschritt vorherzusagen. Zudem fällt auf, dass die Sprünge zwischen aufeinanderfolgenden Vohersagelängen nicht gleichmäßig sind. Dies könnte sich dadurch erklären lassen, dass mit steigender Voraussicht die Vorhersage unsicherer wird. Darüber hinaus lässt sich den Grafiken entnehmen, dass alle Modelle *underfitten*, was ein Hinweis darauf sein könnte, dass sich die Performance aller Modelle weiter steigern ließe, indem beispielsweise der Prozentsatz für

Modell	meanADE [m]	meanFDE [m]	Miss Rate [%]
Seq2Seq	1,7027	3,829	65,578
GCN	**1,6691**	3.776	63,414
GAT-2	1,6859	3,803	63,614
GAT-4	1,6692	**3,773**	**63,087**

Tabelle 4.4: Vergleich der Modelle. Das Seq2Seq-Modell enthält keine *Graph Convoltional* oder *Attention* Layer.

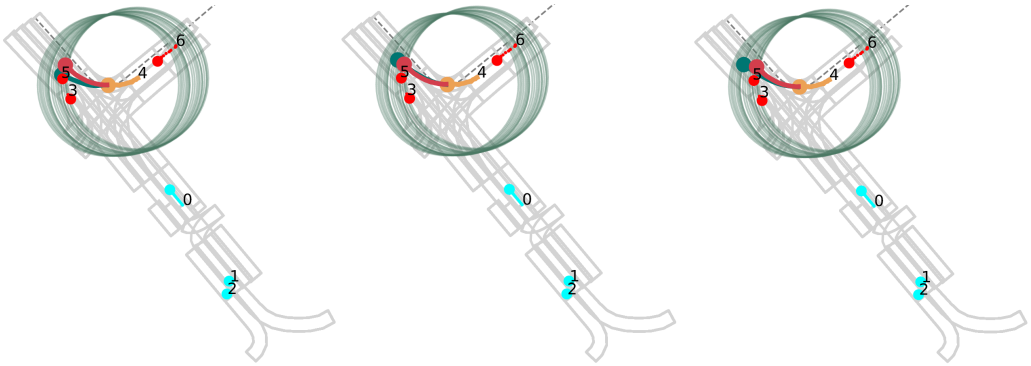
den *Dropout*-Layer herabgesetzt wird.

Neben den angesprochenen Metriken, *FDE* und *Miss Classification Rate*, lässt sich die Qualität der Trajektorievorhersage mit dem **Average Displacement Error** messen. Dieser gibt, an wie weit jede berechnete Position von der wahren Position im Mittel entfernt ist und lässt sich wie folgt berechnen:

$$ADE = \frac{1}{T_{Pred}} \sum_{t=1}^{T_{Pred}} \sqrt{(\tilde{x}^t - x^t)^2 + (\tilde{y}^t - y^t)^2} \tag{4.13}$$

Tabelle 4.4 enthält alle genannten Metriken für die vier trainierten Modelle. Der *mean* FDE bzw. ADE bezieht sich auf das Mittel über jeden Minibatch.

Das Modell mit 4-*Graph Attention Layern* hat die niedrigste *Miss Classification Rate* und den niedrigsten *Final Displacement Error*. Das einfache Graph Convolutional Modell hat den niedrigsten *Average Displacement Error*. Alle Modelle die Graph Convolutions einbeziehen erreichen bessere Messwerte, als das weniger komplexe *Seq2Seq* Modell, welches keine GCN- oder GAT-Schichten verwendet und somit die Trajektorie für den Agenten direkt vorhersagt, ohne die Features der andere Verkehrsteilnehmer einzubeziehen. Die Eingabe wird zunächst durch die 2-dimensionale Convolution in die höhere Dimension mit 64 *Feature Maps* transformiert und anschließend in das *Encoder*-Netz weitergeleitet. Das *Seq2Seq* Modell hat im Vergleich eine um fast 2% schlechtere *Classification Rate*. Bei gleichen Trainingsbedingungen könnte dies ein Hinweis darauf sein, dass die *Graph Features* eine relevantere Repräsentation der Szene enthalten, als der reine Bewegungsverlauf des Agenten, ohne soziale Informationen der räumlichen Nachbarn. Diese Hypothese wird durch die genauere Analyse einiger Vorhersagen für alle Modell gestützt.



(a) (b) (c)

Abbildung 4.10: (a) Vorhersage des Seq2Seq Modells. Die Vorhersage endet bei Fahrzeug 5. **(b)** Vorhersage des GCN Modells. Fahrzeug 5 wird ausgewichen. **(c)** Vorhersage des GAT-4 Modells. Fahrzeug 5 wird ausgewichen.

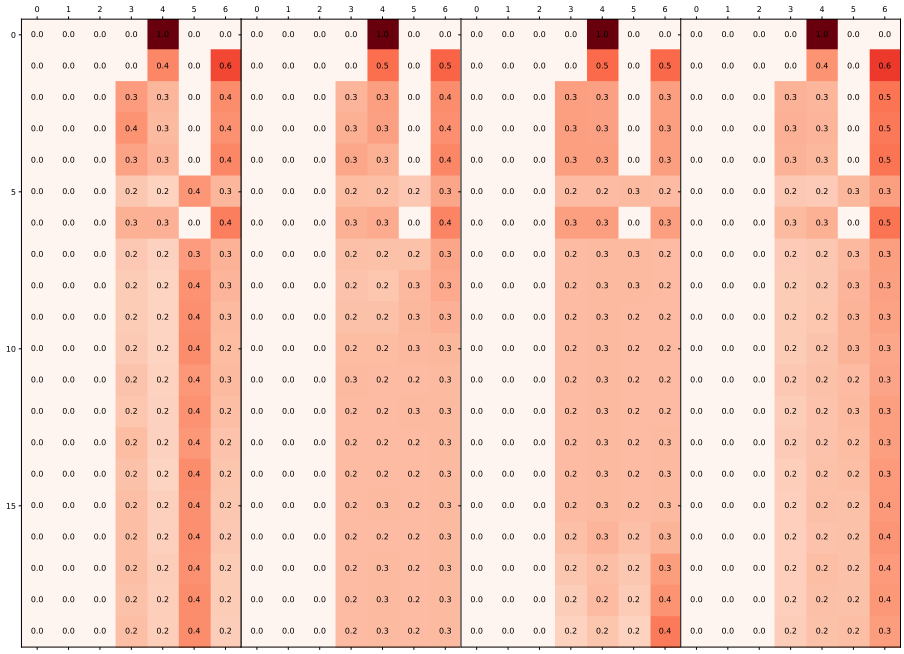


Abbildung 4.11: Attentionmatrix für GAT-4. Die Spaltennummer repräsentiert die Nummer des Verkehrsteilnehmer. Jede Zeile steht für einen Zeitschritt. Je höher das Gewicht ist, desto roter ist die Zelle.

Die berechneten Bewegungsverläufe wurden gemeinsam mit den Straßensegmenten und Mittellinien visualisiert für die Argoverse eine API bereitstellt. Die beobachtete Trajektorie

des Agenten ist **orange** dargestellt und endet bei dem dickeren Punkt. Die türkisen Punkte repräsentieren weitere Verkehrsteilnehmer der Szene, bei denen ein dicker Punkt ebenfalls das Ende der Bewegung markiert. Sind Positionen rot markiert, so bedeutet dies, dass sie in dem Observationsradius des Agenten liegen (grüne Kreise) und eine Kante zu ihnen besteht. Nur für den Agenten ist der wahre und vorhergesagte Verlauf der Trajektorie nach der Beobachtungszeit dargestellt. Die rote Bahn entspricht der wahren Bewegung, wohingegen die grüne Linie die Ausgabe des Modells repräsentiert. Bei einem genauen Vergleich der Szene und dessen Vorhersage fällt auf, dass das *Seq2Seq* Modell nicht zwangsläufig die schlechteste Vorhersage bezüglich des *FDE* macht, allerdings endet die Trajektorie des Agenten in Fahrzeug 5 auf der Gegenfahrbahn. Vergleicht man diese Sequenz mit denen der anderen Modelle, so fällt auf, dass keine einzige Vorhsage Teilnehmer 5 streift. Die Attentionkoeffizienten des GAT-4 Modells lassen sich visualisieren, sodass mittels einer Heatmap schnell beurteilt werden kann, zu welchem Objekt eine starke Aufmerksamkeit besteht. Die Matrix aus Abbildung 4.11 ist eine reduzierte Repräsentation der Adjazenzmatrix. Statt einer einfachen Verbindung, die in der Adjazenmatrix mit einer 1 kodiert wird, gibt es für jeden Teilnehmer, zu dem eine Verbindung besteht, ein Gewicht. Je höher das Gewicht ist, desto roter ist die Zelle. Jede Zeile repräsentiert einen Zeitschritt innerhalb des Beobachtungszeitraums, wohingegen jede Spaltennummer zur Nummer des Verkehrsteilnehmers korrespondiert. Für jeden *Attention Head* gibt es eine eigene Matrix (durch dicke schwarze Linien voneinander getrennt). Die Attentionmatrix zeigt, dass im ersten Kopf für alle Zeitschritte, in denen Fahrzeug 5 innerhalb des Observationsradius vom Agenten ist, eine hohes Gewicht gelegt wird. Zu Fahrzeug 3, welches nur am Rand der Szene steht wird in allen *Attention Heads* wenig Aufmerksamkeit gelegt. Die Vorhersage des *Seq2Seq* Modells unterbietet einige der Modelle in Bezug auf den FDE, jedoch scheinen die Modelle mit Graph Convolutions die Semantik der Aufgabe besser verstanden zu haben.
Enthält eine Szene keinen sozialen Kontext, aus dem die Graph Convolutional Layers *higher level* Features extrahieren können, so sind die gemessenen Metriken für das *Seq2Seq* Modell stellenweise besser, wie die folgende Szene illustriert.

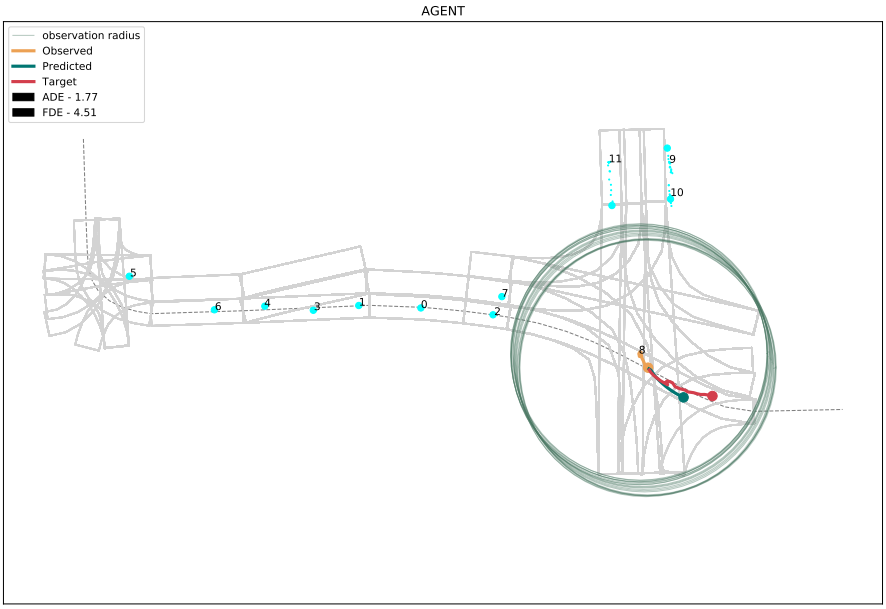


Abbildung 4.12: Vorhersage des Seq2Seq Modells. Die Vorhersage basiert nur auf der vergangenen Bewegung des Agenten.

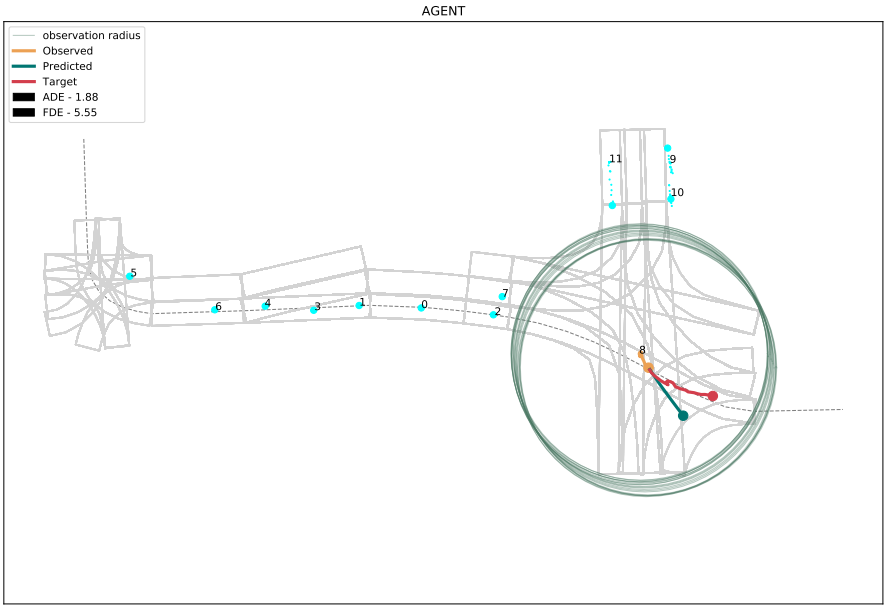


Abbildung 4.13: Vorhersage für das GAT-4 Modell. Ohne sozialen Kontext gleicht die Vorhersage einem linearen Geschwindigkeitsmodell.

Bei der Durchsicht einiger Vorhersagen fällt auf, dass die Modelle, die Graph Convolutions verwenden, bessere Ergebnisse erzielen, wenn der Agent über eine große Nachbarschaft verfügt. Insbesondere dann, wenn diese Nachbarn Trajektorien haben, die nicht parallel zum Agenten verlaufen. Weitere aufgezeichneten Trajektorien befinden sich in Anhang B.

4.4 Adversarial Attacks

Die Eingabe für das Graph Neural Network besteht aus den Knotenmerkmalen X, den Adjazenzmatrizen $A_{spatial}$, die die räumlichen Beziehungen enthalten sowie den Adjazenzmatrizen $A_{temporal}$, die eine zeitliche Verbindung zwischen gleichen Teilnehmern aufbauen. Zur Vorhersage der Trajektorie wählt das Modell aus den aggregierten *Graph Features* diejenigen aus, die zum Agenten gehören. Damit existieren, wie in Kapitel 2.3 beschrieben, für *Node-Level* Vorhersagen, zwei Typen von Knoten. Der **target** Knoten entspricht dem Agenten, wohingegen **attacker** alle zum Agenten adjazenten Nachbarn sind. Auf dieser Basis lassen sich alle *adversarial attacks* in zwei Kategorien einteilen. Die erste Kategorie bezieht sich auf Manipulationen, die den *target*-Knoten direkt manipulieren, ohne dabei andere Objekte zu verändern. Die zweite Kategorie perturbiert ausschließlich *attacker*-Knoten die adjazent zum *target* sind. Für die Eingabe des Modells bedeutet dies, dass *adversarial examples* sowohl die Knotenfeatures, als auch Kanten in den Adjazenzmatrizen manipulieren können. Dabei ist zu beachten, dass sowohl X als auch $A_{spatial}$ dynamisch über T Zeitschritte variieren, wodurch der Raum an möglichen Perturbierungen um den Faktor T steigt. Unter der Annahme, dass die Veränderung eines/r Features bzw. Kante, für einen Zeitschritt, ein *adversarial example* erzeugt, ergeben sich die folgenden Angriffsszenarien:

- Direkte Angriffe auf den *target*-Knoten

- Indirekte Angriffe auf *attacker*-Knoten

Unter direkten Angriffen lassen sich solche zusammenfassen, die entweder die Attribute, die Kante, oder beides gleichzeitig, des *targets* für einen Zeitschritt manipulieren. Abbildung 4.14 illustriert die Merhfache Manipulation der Graphstruktur für den *target* Knoten. Eine Perturbierung für die Adjazenzmatrix kann sowohl das Hinzufügen einer Kante bedeuten als auch das Entfernen einer solchen.

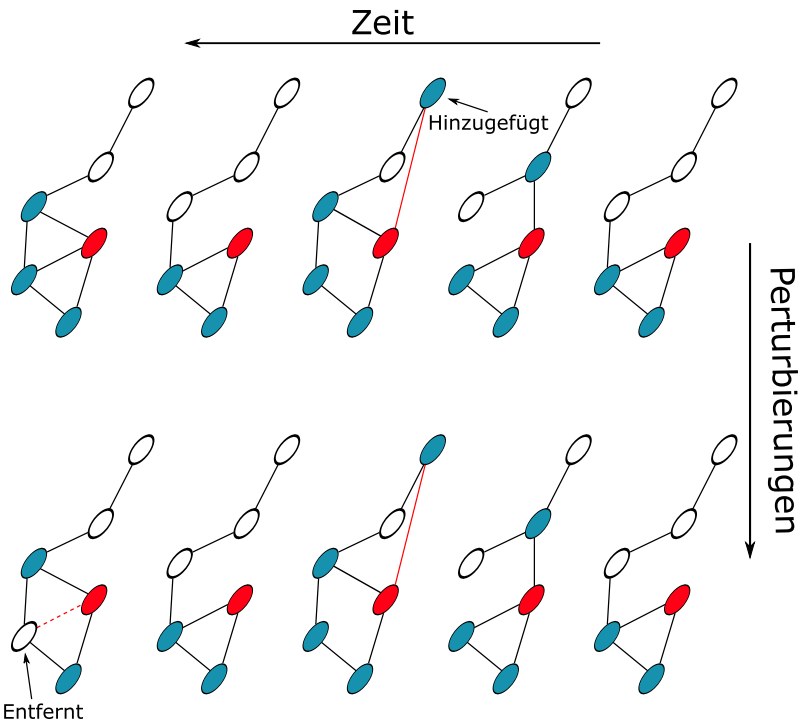


Abbildung 4.14: Direkter Angriff auf den *target*-Knoten. In jedem Perturbierungsschritt können Kanten vom *target* hinzugefügt oder entfernt werden.

Bei direkten Angriffen auf die Struktur können nur Kanten zu Nachbarn entfernt werden, die zu Beginn adjazent waren. Umgekehrt ist es ausschließlich möglich Kanten zu Verkerhsteilnehmern herzustellen, zu denen es zuvor keine Verbindung gab. Weiterhin wird die Invertierung einer Kante für einen Zeitschritt gespeichert, sodass wiederholte Perturbierungen nicht möglich sind. Bei V Teilnehmer einer Szene, die über T Zeitschritte beobachtet wurde, hat die Matrix $A_{spatial}$ die Dimension (VT, VT). Damit existieren für den Agenten in einem Zeitschritt $V-1$ Kanten die invertiert werden können, denn die Verbindung zu sich selbst wird nicht invertiert. Sind Perturbierungen über alle Zeitschritte möglich, so ergibt sich ein möglicher Raum für *adversarial examples* C_{struct} der Größe $T(V-1)$. Sind mehrere Manipulationsschritte geplant, so reduziert sich dieser Raum um je ein Element pro Schritt, da dieselbe Kante nicht mehrfach invertiert werden darf.

Für indirekte Angriffe auf die Graphstruktur gelten dieselben Regeln für Attacker, d.h. alle Kanten können invertiert werden, mit Ausnahme der Verbindungen die zum *target* führen. Das entfernen solcher Kanten ist Bestandteil der direkten Angriffe. Indirekte Angriffe auf die Struktur sind aufgrund der höheren Zahl an Zielen komplexer zu berechnen und enthalten mehr *adversarial examples*. Für jeden Zeitschritt ist es möglich, dass der Agent eine andere Menge an *Attacker* Nodes I_t hat. Ein *attacker* kann in einem Zeitschritt $V-2$ mögliche Verbindungen invertieren, die beiden Ausnahmen bilden der Angreifer und

der Zielknoten. In einem Zeitschritt bedeutet dies für die Menge der *adversarial examples*, dass sie die Größe $I_t(V-2)$ haben. Für alle Zeitschritte summiert sich die Menge zu:

$$C_{struct} = \sum_{t=1}^{T} I_t(V-2)$$

Da sich die Nachbarschaftsbeziehung über binäre Entscheidungen $\{0,1\}$ festlegen lässt ist es möglich den gesamten kombinatorischen Raum für alle *adversarial examples* pro Manipulationsschritt zu bestimmen und aus dieser Menge denjenigen auszuwählen, welcher die Vorhersage des Netzwerkes am stärksten negativ beeinflusst. Bei Angriffen, die die Knotenattribute betreffen, ist es nicht möglich zu bestimmen welche Veränderung zur schlimmsten Vorhersage führt, da sich die Eingabe X in $\mathbb{R}$ befindet. Es gibt unendlich viele Richtungen, in die man mit unendlich vielen verschiedenen Magnituden gehen könnte. Um dennoch, zu strukturellen Angriffen, vergleichbare Szenarien zu schaffen kann ein ϵ festgelegt werden, in die jedes Attribut maximal bewegt werden darf. Die *Fast Gradient Sign Method* erlaubt es für jeden Knoten eine solche Perturbierung durchzuführen.

$$\tilde{X^t_{i,p}} = \tilde{X}^t_{i,p-1} + \epsilon * sign(\nabla_{X_{i,p}} L(\theta, X^t_{i,p}, y)) \tag{4.14}$$

Im p-ten Perturbierungsschritt werden in einem Zeitschritt t alle Features $X^t_{i,p}$ eines ausgewählten Knotens i, um einen Betrag ϵ in die Richtung verschoben, die den Fehlerterm maximiert. Da dieser Lösungsraum nicht konvex ist, führt der Gradient von X nur lokal in die Richtung des steilsten Anstiegs. Wie Goodfellow et. al gezeigt hat reicht die *FGSM* dennoch dazu aus manipulierte Eingaben zu erzeugen, die das Modell täuschen [15]. Wäre nur eine Perurbierung pro Szeneobjekt und Zeitschritt erlaubt, so wäre im Falle von direkten Feature Angriffen, nach 20 Manipulationen der mögliche Raum an Veränderungen erschöpft. Aus diesem Grund ist es zulässig, dass Knotenattribute für ein Verkehrsteilnehmer mehrfach innerhalb eines Zeitschrittes verändert werden können. Wie zuvor bereits festgelegt, werden bei direkten Angriffen nur die Knotenattribute des Agenten manipuliert und bei indirekten Angriffen die Attribute der adjazenten Nachbar des *targets*. Abbildung 4.15 illustriert die Perturbierung von Knotenattributen für 2 Perturbierungsschritte für den indirekten Fall.

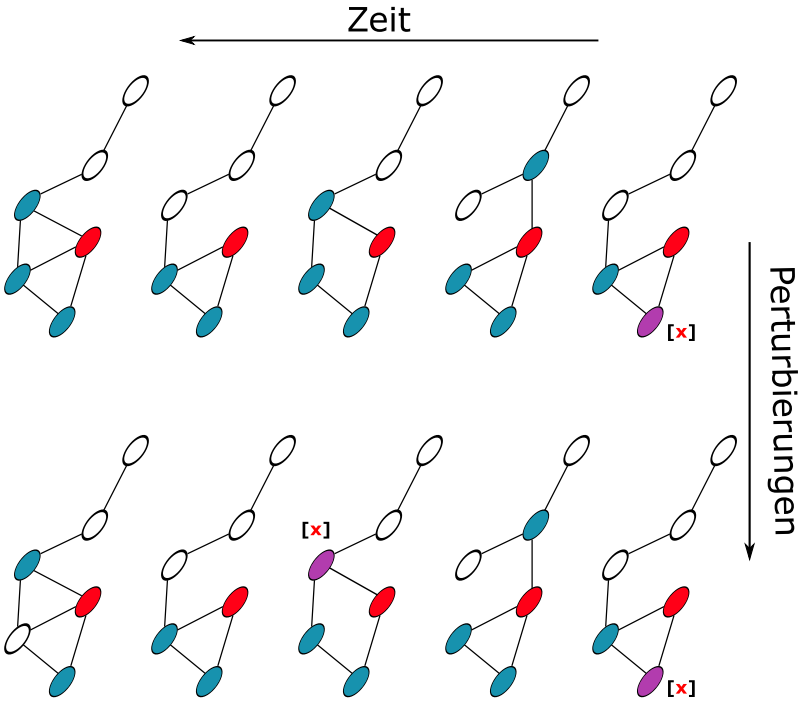


Abbildung 4.15: Indirekter Angriff auf den *target* Knoten. In jedem Perturbierungsschritt können die Attribute von adjazenten Teilnehmern manipuliert werden. Features können für denselben Zeitschritt mehrfach manipuliert werden.

Für direkte Angriffe auf die Features des *targets* bedeutet dies für die Menge der möglichen *adversarial examples* C_{feat}, dass sie die Größe T hat, da für jeden Zeitschritt nur eine Veränderung an den Attributen möglich ist. Für indirekte Angriffe beträgt diese Größe $T(V-1)$. Die Angriffsszenarien auf Struktur und Features lassen sich für beide Kategorien, direkt und indirekt, kombinieren. In diesem Fall wird die Menge der *adversarial examples* zusammengelegt und aus dieser Menge das einflusschreichste Examplar ausgewählt. Um zu bewerten welcher Angriff den größten *Erfolg* hat das Netz zu täuschen wird die Fehlerfunktion herangezogen, mit der das Netz trainiert wurde. Wie in Kapitel 4.2 beschrieben, wird für beide Eingaben separat die Menge an *adversarial examples* berechnet. Aus beiden Mengen wird jeweils das Examplar ausgewählt, welches den höchsten Fehler verursacht hat.

$$\underset{\hat{X}_p \in \Phi(\hat{X}_{p-1}), \hat{A_{spatial}}_p \in \phi(\hat{A_{spatial}}_{p-1})}{\arg\max} \sum_{p=1}^{P} L(f(\hat{X}_p, \hat{A_{spatial}}_p, y)$$

Der Fehler beider Kandidaten wird verglichen und es wird derjenige fortgeführt der den größeren Fehler verursacht hat. Der beste Kandidat dient als Ausgangspunkt für den nächsten Perturbierungsschritt. Der Pseudocode 1 formalisiert die Angriffsprozedur:

Algorithmus 1 : Adversarial Attack

Eingabe: Graph $G^0 \leftarrow (A^0, X^0)$, *target* Knoten v_0, Attacker Knoten $\mathcal{A}$, Perturbierungsbudget Δ

Ausgabe: Modifizierter Graph $G' = (A', X')$

$p \leftarrow 0$;

while $|A^p - A^0| + |X^p - X^0| < \Delta$ **do**

  if *struct* **then**

    if *direct* **then**

      $C_{struct} \leftarrow \phi(A^p, v_0)$;

    else

      $C_{struct} \leftarrow \phi(A^p, \mathcal{A})$;

    end

    $A_{adv} = \arg\max_{e \in C_{struct}} L((e, X^p, v_0)$;

  end

  if *feature* **then**

    if *direct* **then**

      $C_{feat} \leftarrow \Phi(X^p, v_0)$;

    else

      $C_{feat} \leftarrow \Phi(X^p, \mathcal{A})$;

    end

    $X_{adv} = \arg\max_{f \in C_{feat}} L((A^p, f, v_0)$;

  end

  if $L((A^p, X_{adv}, v_0) > L((A_{adv}, X^p, v_0)$ **then**

    $G^{p+1} \leftarrow G^p \pm X_{adv}$;

  else

    $G^{p+1} \leftarrow G^p \pm A_{adv}$;

  end

  $p \leftarrow p + 1$;

end

return G^p

Als Eingabe bekommt der Algorithmus einen Graphen $G^0 \leftarrow (A^0, X^0)$ mit den nicht perturbierten Eingaben, sowie dem *target* Knoten v_0 und eine Liste aller *Attacker* $\mathcal{A}$ für jeden Zeitschritt. Das Perturbierungsbudget Δ legt fest wieviele Manipulation in Summe an beiden Eingaben vorgenommen werden darf. Für die Graphstruktur kann dazu die L_0-Norm aus Kapitel 2.3 benutzt werden, die die Summe aller Einträge ungleich Null wiedergibt. Für $|A^p - A^0|$ ist dies gleich der Anzahl an Manipulationen. Die Perturbierungen

an Knotenattributen müssen mitgezählt werden, da diese nicht indirekt aus einer Norm gewonnen werden können. Anschließend werden die Menge für die *adversarial examples*, je nachdem ob ein direkter oder indirekter Angriff vorliegt, für die Graphstruktur A und Feature X berechnet. Für beide Menge wird jeweils das Objekt ausgewählt, welches als Argument den Fehlerterm maximiert. Die beiden Ergebnisse werden wiederum miteinander verglichen, um den potenteren Angriff zu erhalten. Der stärkere Angriff dient als Grundlage für den nächsten Perturbierungsschritt, sodass sowohl die Graphstruktur, als auch die Knotenattribute iterativ manipuliert werden.
Die Konstruktion der *adversarial attacks* als **Greedy Algorithmus** bietet den Vorteil, dass sie für Perturbierungsbudgets $\Delta \leq 30$ in annehmbarer Zeit berechenbar ist. Jedoch birgt sie den Nachteil, dass sie nicht das globale Maximum für den potentesten Angriff findet. Zu diesem Zweck müsste die Kombinationen von allen Struktur- und Feature-Angriffen berechnet werden. Des Weiteren kann die arg max-Funktion dazu führen, dass die *adversarial attack* vorzeitig in einem lokalen Maximum endet, da aus einem zuvor erfolgreichen Angriff nicht zwangsläufig potentere erschaffen werden können.
Mit dieser Palette an Angriffen können divsere kurzfristige Sensorfehler simuliert werden. Jedoch besteht die Möglichkeit, dass ein Sensor dauerhaft korrupt ist, sodass der Fall untersucht werden muss, welchen Einfluss strukturelle Angriffe haben, die auf alle Zeitschritte gleichzeitig wirken. Aus diesem Grund werden die bestehenden Angriffsszenarien um ein direktes/indirektes Szenario erweitert, welches zum Ziel hat in einem Perturbierungsschritt die Kanten zu einem Fahrzeug für alle Zeitschritte auf einmal zu invertieren. Abbildung 4.16 zeigt dies beispielhaft für den ersten Manipulationsschritt des *target*-Knotens.

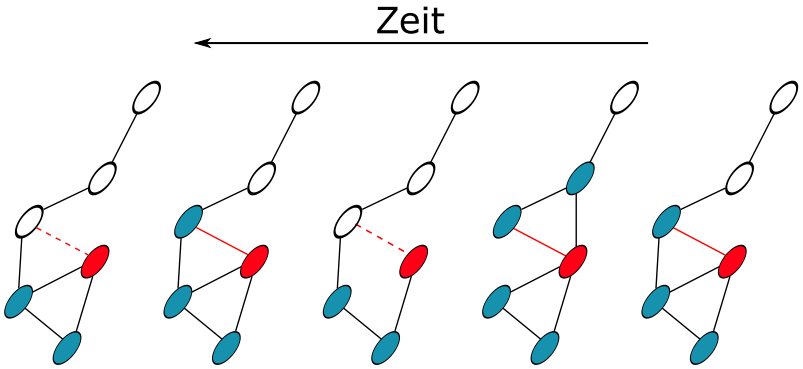


Abbildung 4.16: Direkter Angriff auf den *target*-Knoten. Es wird ein adjazenter Teilnehmer ausgewählt und für alle Zeitschritte werden die Kanten invertiert.

Unter der Annahme, dass die Invertierung einer Verbindung das Modell größtmöglich täuscht, wird die Verbindung zu einem ausgewählten Nachbarn in nur einem Perturbierungsschritt für alle Zeitschritte gleichzeitig invertiert. Für $T = 20$ werden somit 20 direkte strukturelle Angriffe in einem zusammengefasst.
Die beschriebenen Angriffe sollen auf die in Kapitel 4.3 beschriebenen Modelle angewendet

werden, die Graph Convolutions zur Vorhersage einsetzen. Um den rechnerischen Aufwand gering zu halten wurde das Perturbierungsbudget für alle Angriffe auf $\Delta = 30$ gesetzt. Zudem gilt für alle Featureangriffe $\epsilon = 0.001$. Um die Szenarien untereinander vergleichen zu können werden für jedes Modell und jeden Angriff folgende Daten ausgewertet:

- Fehlerterm
- FDE
- MRC

Um die Effektivität der *adversarial examples* zu bestimmen, wird der *Root Mean Squared Error* 4.12 benutzt, der in Kapitel 4.3 eingeführt wurde, um die Modelle zu trainieren. Da einige Angriffe für die vollständige Evaluation von allen 39.400 Testsamples, auf den zuvor beschriebenen Trainingsmaschinen, mehrere Wochen benötigen, wurde der Testdatensatz auf 5000 Samples reduziert. Um weiterhin die Verteilung des Testdatensatzes approximativ zur wahren, wurden die 5000 Samples zufällig aus der Grundmenge ausgewählt. Dies führt dazu, dass die ermittelten Metriken aus Kapitel 4.3, die auf dem gesamten Testdatensatz ermittelt wurden, von denen der Angriffe abweichen können. Daher wird in den nachfolgenden Kapiteln jeweils der Anstieg Δ für den Fehler, FDE und MCR angegeben. Abbildung 4.17 und 4.18 zeigen die Vorhersage der Szene aus 4.10c nach 30 Perturbierungsschritten für das GAT-4 Modell. Die *violette* Linie zeigt die Trajektorie nach 30 Angriffen.

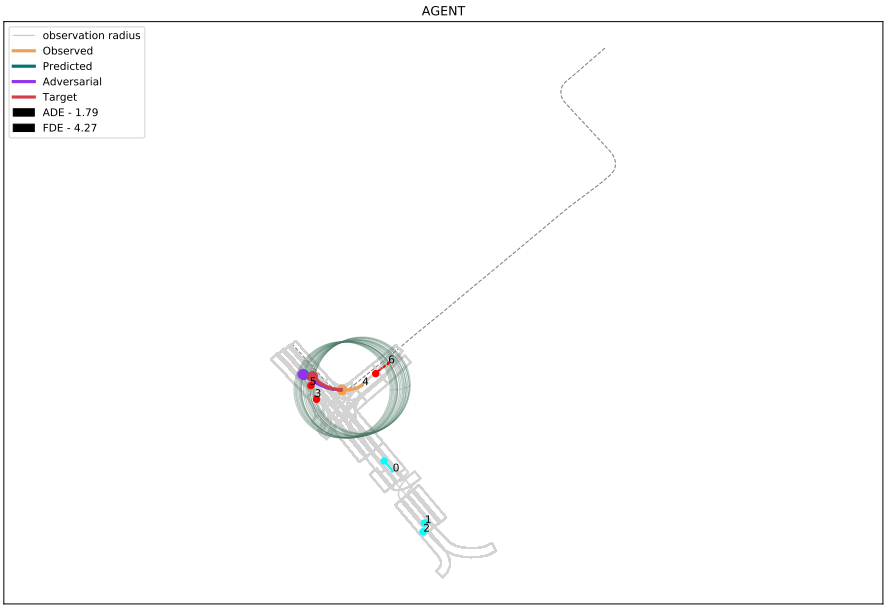


Abbildung 4.17: Dasselbe Sample wie in Abbildung 4.10c nach 30 Angriffen auf die Struktur, die den *target*-Knoten betrifft.

Der FDE hat sich um mehr als 0.5 Meter erhöht. Zu welchen Teilnehmer eine Verbindung hergestellt bzw. entfernt wurde lässt sich indirekt aus der Attention Matrix aus Abbildung 4.18 ablesen.

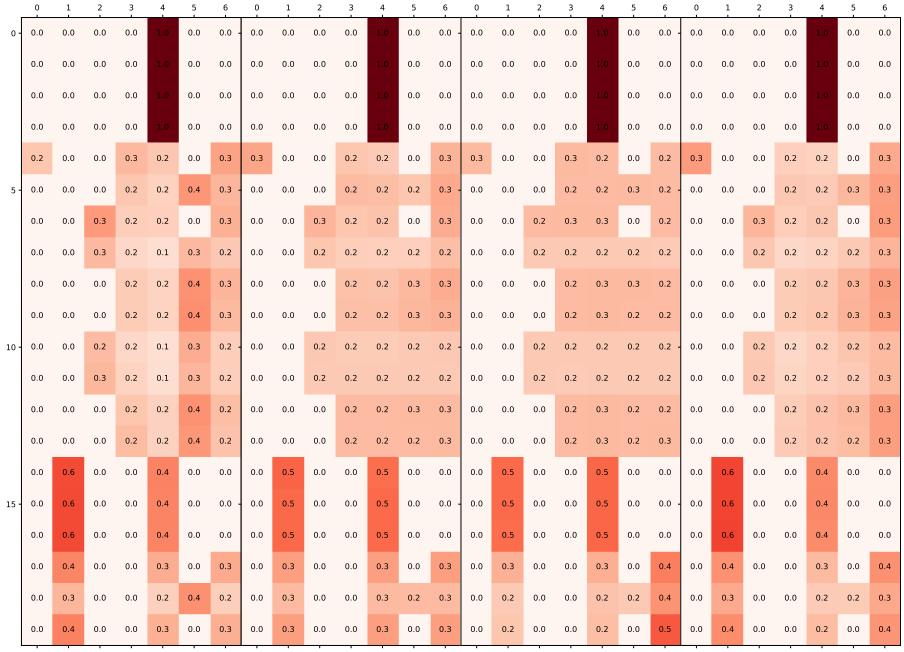


Abbildung 4.18: Die Attentionmatrix zur Vorhersage aus Abbildung 4.17. Die Angriffe fokussieren sich auf spätere Zeitschritte.

Das Angriffsmodell hat vorwiegend neue Kanten zu neuen Teilnehmern in den letzten Zeitschritten hinzugefügt, was das Vorhersagemodell dazu veranlasst hat viel Gewicht auf diese zu legen. Gleichzeitig wurden zu vielen adjazenten Nachbarn in den letzten Zeitschritten Kanten entfernt. Spalte 4 repräsentiert den Agenten und bleibt durchgängig bestehen, da diese von allen Angriffen ausgeschlossen ist.

Kapitel 5

Evaluation

Um die Verwundbarkeit der Graph Neural Networks *GCN, GAT-2* und *GAT-4* zu evaluieren, die in Kapitel 4.3 erstellten wurden, werden die folgenden acht *adversarial attacks* auf den Testdatensatz von *Argoverse* angewendet:

- Direkter Featureangriff
- Direkter Strukturangriff
- Direkter Struktur u. Featureangriff
- Direkter Angriff auf alle Zeitschritte
- Indirekter Featureangriff
- Indirekter Strukturangriff
- Indirekter Struktur u. Featureangriff
- Indirekter Angriff auf alle Zeitschritte

Dabei steht die Frage im Vordergrund, ob sich Charakteristiken bezüglich der Stärke verschiedener Verfahren hervorheben. Für alle Modelle und Angriffe werden der Einfluss auf den *Fehler, FDE* und die *Miss Classification Rate* gemessen. Die Erhöhung bzw. Verringerung dieser Maße wird für alle Angriffe als Differenz zwischen maximaler Perturbierung, bei 30 Veränderungen, und keiner Manipulation der Eingabe angegeben. Auf diese Weise lassen sich die Messungen untereinander vergleichen, sodass objektiv bewertet werden kann welches Szenario den stärksten Einfluss auf ein spezifisches Modell hat. Die Evaluation schließt in Abschnitt 5.3 mit einer direkten Gegenüberstellung aller Angriffsszenarien und Modelle.

5.1 Direkte Angriffe

Direkte *adversarial attacks* im *Node-level* Szenario für Graph Neural Networks zeichnen sich dadurch aus, dass alle Perturbierungen sich ausschließlich auf den *target* Knoten beziehen, dessen Trajektorie vorhergesagt werden soll. Alle Attribute, die nicht zum Agenten gehören, oder Kanten innerhalb der räumlichen Adjazenz-Matrix, die nicht zum *target* adjazent sind, werden bei allen Verfahren dieser Art ignoriert.

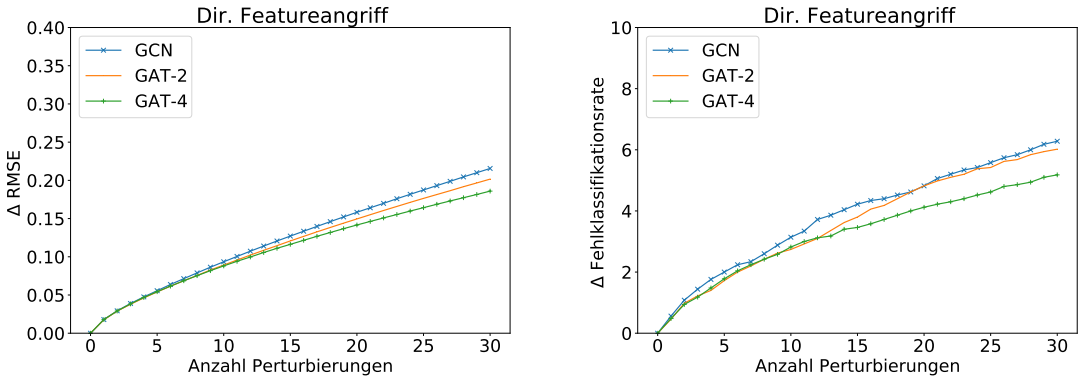
5.1.1 Direkte Featureangriffe

Direkte Feature Angriffe haben die Attribute des Agenten zum Ziel. Diese bestehen in einem Zeitschritt aus dem Abstand zur Mittellinie, der gefahrenen Strecke entlang der Mittellinie, den minimalen Abständen zum Vorder- bzw. Hintermann sowie den normalisierten Koordinaten der Trajektorie. Ein Angriff wählt einen Zeitschritt aus der Beobachtung aus und manipuliert alle genannten Features auf einmal. Dabei kommt die in Kapitel 4.4 vorgestellte *Fast Gradient Sign Method* mit $\epsilon = 0.001$ zum Einsatz. Tabelle 5.1 stellt die Ergebnisse für den Fehler, FDE und Miss Classification Rate (MR) für alle Modelle gegenüber.

Methode	GCN	GAT-2	GAT-4
Feature Fehler	$\mathbf{\Delta 0{,}2155}$	$\Delta 0,2016$	$\Delta 0,1859$
Feature FDE [m]	$\mathbf{\Delta 0{,}40}$	$\Delta 0,37$	$\Delta 0,35$
Feature MR [%]	$\mathbf{\Delta 6.28}$	$\Delta 6,02$	$\Delta 5,18$

Tabelle 5.1: Einfluss der direkten Featureangriffe auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Die eingetragenen Differenzen Δ entsprechen der Abweichung nach 30 Perturbierungsschritten von der Originaleingabe. Die Werte für das GCN übertreffen die des GAT-2/4. Dabei fällt auf, dass der Einfluss von Featureangriffen mit zusätzlichen *Attention-Heads* abnimmt. Dort ist eine Abnahme von $0,84\%$ bei der *Miss Classification Rate* zu beobachten. Dies macht bei einem Testdatensatz mit einer Größe von ca. 39.400 Samples ca. 330 mehr Fehlklassfizierungen aus. Die Grafiken in 5.1 stellen den Anstieg der Differenz für den *RMSE*-Fehler und die *Miss Classification Rate* von 0 bis 30 Perturbierungsschritte dar.

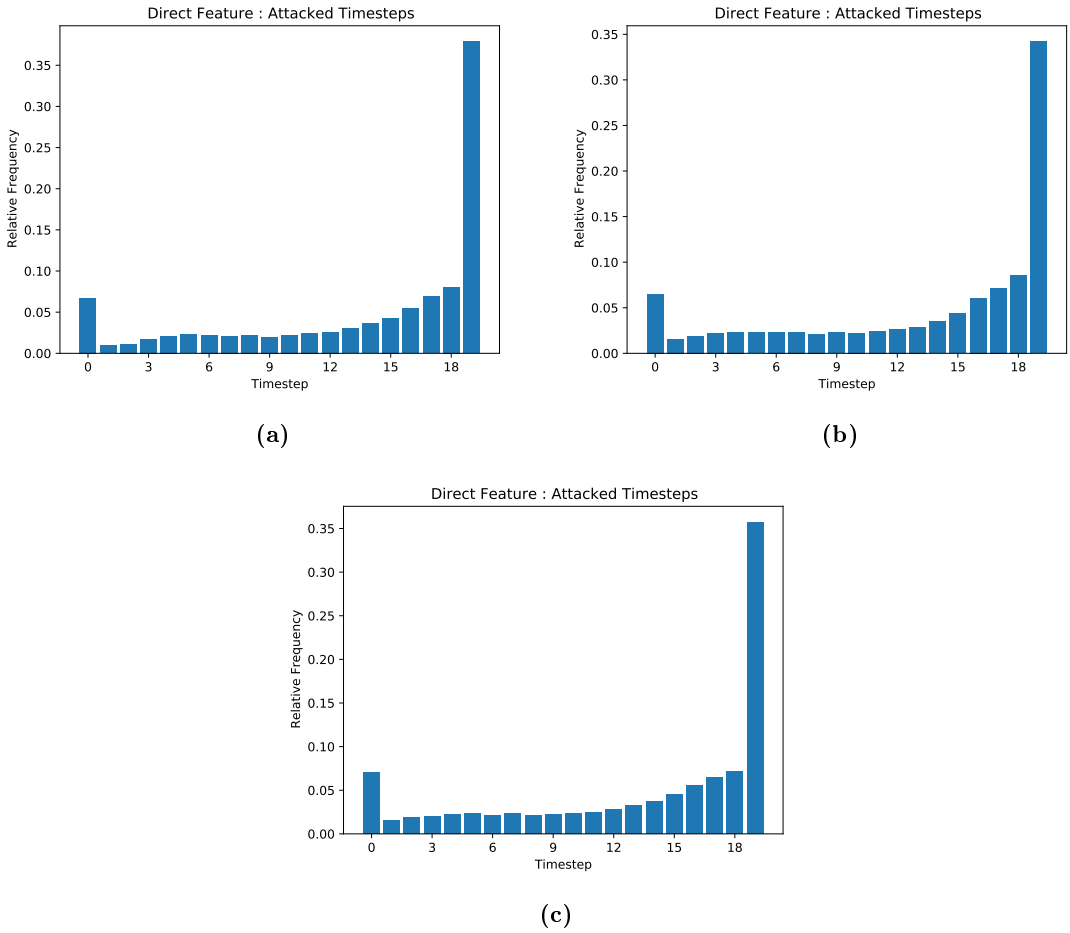


(a) **(b)**

Abbildung 5.1: Direkter Featureangriff. **Achtung** angepasste Skalen zur besseren Interpretierbarkeit. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

Der Vergleich der Modelle bezüglich des RMSE zeigt, dass das GCN eine stärkere Steigung gegenüber den anderen Modellen aufweist. Das GAT-4 scheint am robustensten bzgl. des Fehlers zu sein, jedoch zeigt sich in der nebenstehenden Grafik 5.1b ein anderes Bild. Dort hat die *Miss Classification Rate* für das GCN und GAT-2 einen ähnlichen Verlauf, von dem sich das GAT-4 deutlich abhebt. Die Diskrepanz zwischen RMSE und MCR aller Modelle lässt sich durch den indirekten Zusammenhang beider Messwerte erklären. Der *Root Mean Squared Error* trainiert das Modell darauf die mittlere Abweichung von der Vorhersage gering zu halten und bestraft größere Ausreißer stärker. Die Auswahl von *adversarial examples* auf dieser Metrik bedeutet daher nicht zwangsläufig, dass der FDE erhöht wird. Es ist genauso gut möglich, dass die Trajektorie zu jedem anderen Zeitpunkt manipuliert wird, aber dennoch an der wahren letzten Position auskommt. Auffällig ist zudem, dass die Verlaufskurven für Fehler und MR für keines der Modelle abflacht. Der Verlauf lässt schlussfolgern, dass alle Modelle bei steigender Anzahl an Perturbierungen zu einem linearen Anstieg des Fehlers und MCR führen werden.
Eine Möglichkeit die *adversarial attacks* zu analysieren bieten Histogramme über die angegriffenen Zeitschritte. Die Histogramme für alle angegriffenen Zeitschritte für alle Modelle sind in den Abbildungen 5.2 dargestellt.

Die Verteilung für GCN, GAT-2 und GAT-4 weisen vergleichbare Strukturen auf. Auffällig ist, dass es klar präferierte Zeitschritte gibt. Dies ist zu einem der Zeitschritt 19, der dem letzten beobachteten Zeitpunkt innerhalb einer Szene entspricht. Das *regressive* Seq2Seq-Modell startet von der zuletzt beobachteten Position seine Vorhersage. Da das Modell die neuen Positionen additiv aus vorhergesagten Positionsdifferenzen erstellt, kann eine Manipulation des letzten Zeitschritt einen großen Fehler im FDE hervorrufen, der folglich zu einem höheren Fehler in der *Miss Classification Rate* führt. Eine mögliche Erklärung für bevorzugte Angriffe auf den 0-ten bzw. ersten beobachteten Zeitpunkt innerhalb der



(a) **(b)**

(c)

Abbildung 5.2: Direkter Featureangriff: Verteilung der angegriffenen Zeitschritte: **(a)** GCN, **(b)** GAT-2, **(c)** GAT-4

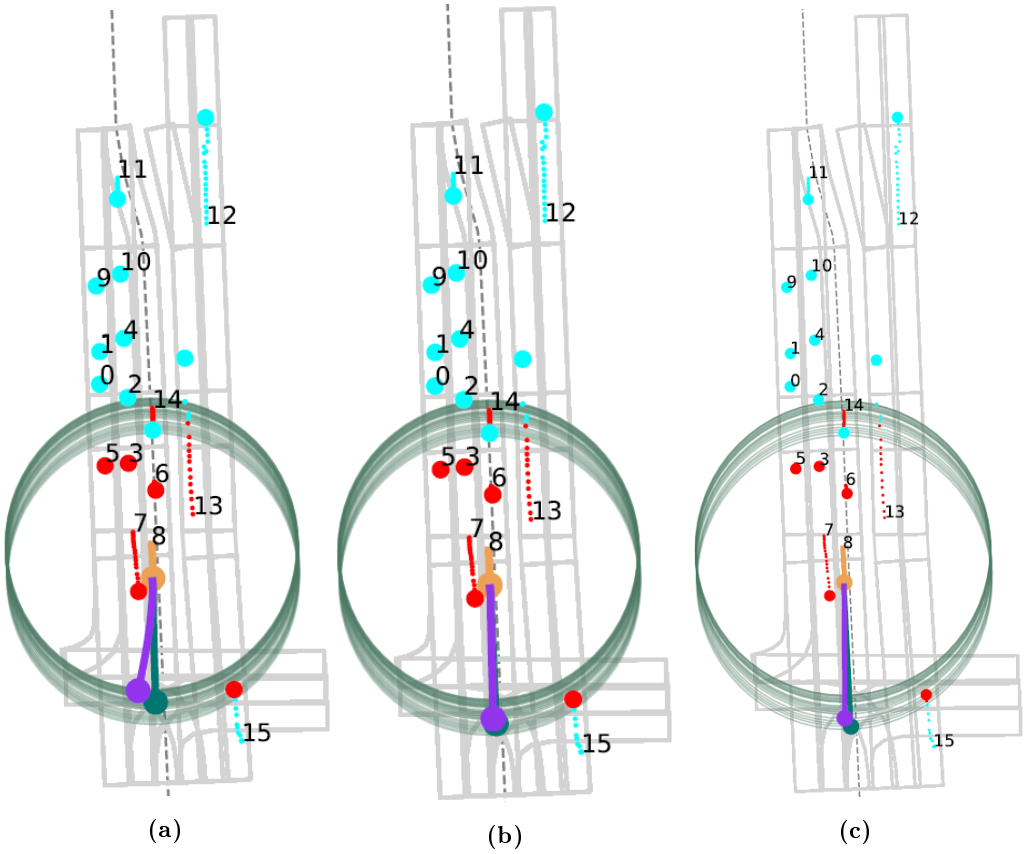


Abbildung 5.3: **(a)** GCN: ΔFDE: 2.89 , **(b)** GAT-2: ΔFDE: 0.76, **(c)** GAT-4: ΔFDE: 1.39

Szene, könnte sein, dass alle Positionen innerhalb der Zeitreihe aus zeitlichen Nachbarn approximiert werden können. Zeitschritt 0 und 19 sind jedoch Randpunkte, deren Informationen sich höchstens aus einem zeitlichen Nachbarn annähern lassen. Die im Kapitel 4.3 vorgestellte *temporal convolution* aggregiert die Features zum Zeitpunkt t eines Knotens mit denen von $t-1$ und $t+1$. Für die äußerten Punkte ist diese Faltung nur in eine Richtung möglich, wodurch Näherungswerte nicht durch die zeitlichen Nachbarn gewonnen werden können. Wird die *temporal convoltuion* für alle Zeitschritte durchgeführt, so verfügt die Zeitreihe über gleichmäßig aggregierte Features, mit den beiden Ausnahmen des ersten und letzten Zeitschrittes, deren Manipulation sich somit stärker auf die Vorhersagegenauigkeit auswirken kann.

Ein Vergleich der Abbildungen in 5.3 zeigt, dass Modelle die individualisierete Kantengewichte vergeben können, weniger stark von direkten Featureangriffen betroffen sind. Das GCN-Modell verlässt die zuvor Vorhergesagte Route, wohingegen die Route der GAT-Modelle gestreckt wird.

5.1.2 Direkte Strukturangriffe

Direkte Strukturangriffe haben Kanten innerhalb der räumlichen Adjazenz-Matrix zum Ziel, die den Agenten mit anderen Verkehrsteilnehmer verbindet. Alle anderen Verbindungen werden bei Perturbierungen dieser Art ignoriert. Da der Graph in der Zeit dynamisch ist, gibt es für jeden Zeitschritt eine unterschiedliche Menge an Kanten die man hinzufügen/entfernen kann. Beispielsweise ist es möglich, dass der Agent zu einem Teilnehmer in Zeitschritt 1 eine Verbindung hat, sich diese aber im Verlauf der Szene auflöst. In diesem Fall würde ein direkter Strukturangriff diese Kante in Schritt 1 entfernen, wohingegen sie in darauffolgenden Schritten hinzugefügt werden würde. Tabelle 5.2 stellt die Ergebnisse für direkte Angriffe auf die Struktur für alle Modelle gegenüber.

Methode	GCN	GAT-2	GAT-4
Struct Fehler	$\Delta 0,0182$	$\Delta 0,4617$	$\mathbf{\Delta 0{,}7035}$
Struct FDE [m]	$\Delta 0,03$	$\Delta 0,37$	$\mathbf{\Delta 1{,}31}$
Struct MR [%]	$\Delta 0,54$	$\Delta 6,02$	$\mathbf{\Delta 16{,}15}$

Tabelle 5.2: Einfluss der direkten strukturellen Angriffe auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Die Ergebnisse verhalten sich invers zu direkten Featureangriffen. GCN reagieren nach 30-Perturbierungen der Adjazenz-Matrix unmerklich in Bezug auf Fehler, FDE und MCR. Dies ändert sich schlagartig für Modelle die Graph-Attention Layer beinhalten. Obwohl die Attention-Koeffizienten zwischen mehreren *Attention-Heads* gemittelt werden, scheint die Menge an *Heads* die Erfolgsquote von direkten Angriffen auf die Struktur positiv zu beeinflussen. Der Unterschied zwischen zwei und vier *Attention-Heads* liegt bei mehr als 10%. Bei einer ursprünglichen Fehlklassifikationsrate von $63,08\%$ (24.853 Sequenzen über 2 Meter) des GAT-4 Modells bedeutet dies nach 30-Manipulationen der Struktur, dass ca. 31.216 Sequenzen eine höhere Abweichung als 2 Metern haben. Die Abbildungen in 5.4 lassen vermuten, dass der Angriff noch mehr Schaden anrichten kann, wenn weitere Perturbierungen durchgeführt werden.

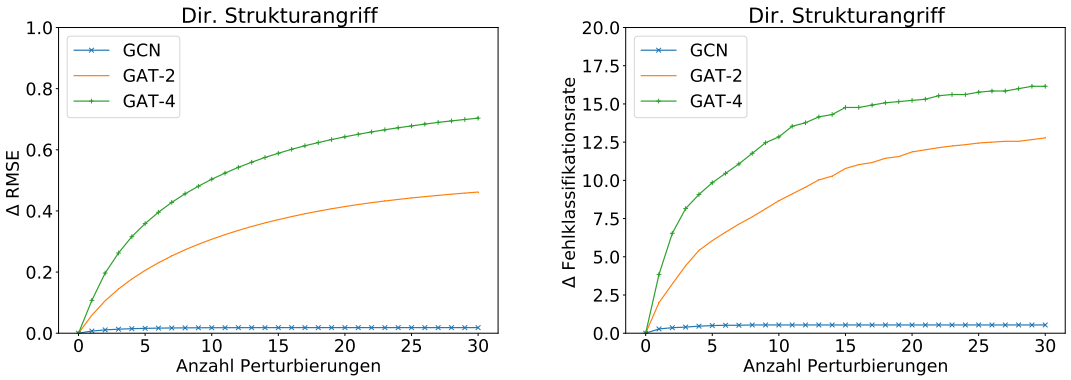
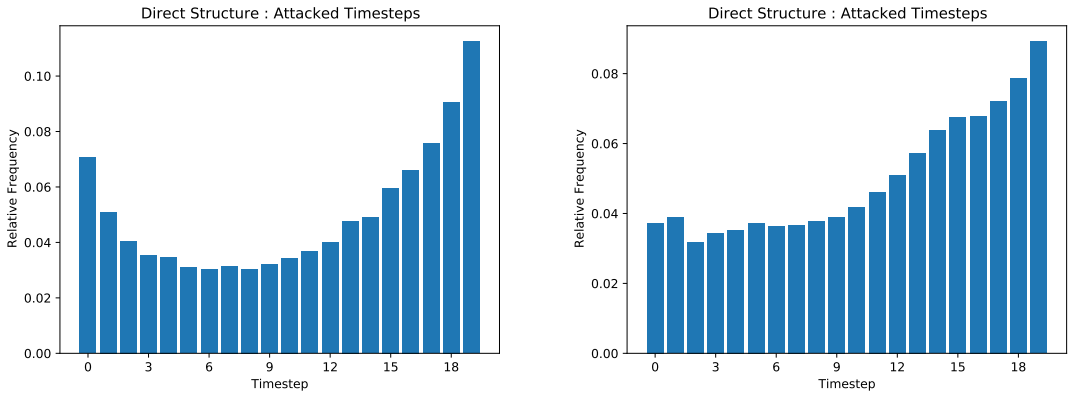


Abbildung 5.4: Direkter Strukturangriff. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

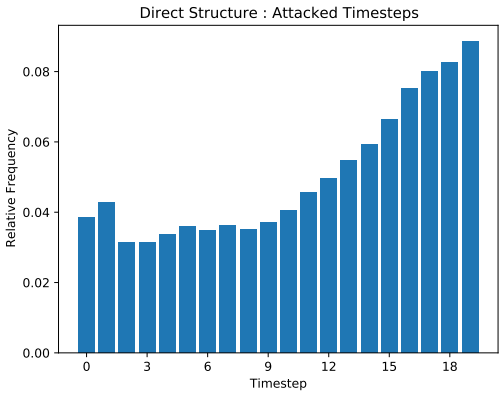
Sowohl die Kurve für den Fehler, als auch für die Fehlklassifikationsrate, flachen für die Attention basierten Modelle ab. Zwei mögliche Erklärungen sind die folgenden. Zum einem ist es möglich, dass keines der *adversarial examples*, die in einem Zeitschritt vorgeschlagen werden vom *greedy* Algorithmus akzeptiert werden, da sie den Modellfehler nicht erhöhen. Ein anderer Erklärungsansatz besteht in der Möglichkeit, dass die Menge der potenten Kandidaten nach wenigen Schritten erschöpft ist. Enthält eine Szene lediglich einen weiteren Verkehrsteilnehmer neben dem Agenten, so gibt es, wie in Kapitel 4.4 vorgestellt, 20 mögliche Manipulationen der Graphstruktur. Die wahre Vulnerabilität dieses Architekturtyps könnte höher liegen, erfordert aber weitere Untersuchungen des Datensatzes.
Die Verteilung der angegriffenen Zeitschritte in den Abbildungen 5.5 spiegelt ein ähnliches Bild wider, wie in Abschnitt 5.1. Der relative Anteil des angegriffenen letztens beobachteten Zeitschritts überwiegt bei allen Modellen.

Die Verteilung der Zeitschritte nimmt beim GCN Modell einen nahezu kontinuierlichen Verlauf an, mit zwei Maxima im ersten bzw. letzten Zeitpunkt. Der Vorzug dieser Zeitschritte könnte damit begründet werden, dass die räumliche Faltung die Features der direkten Nachbarn aggregiert. Da das Seq2Seq-Modell die Graph-Features für alle beobachteten Zeitschritte des Agenten aufnimmt, pflanzt sich dieser Fehler in der Vorhersage fort. Die Begründung für diese Zeitpunkte ist damit analog zu der in Abschnitt 5.1. Die Attention basierten Modelle legen verhältnismäßig mehr Wert auf Zeitschritte, die sich dem Beobachtungsende nähern. Die Zeitpunkte zwischen 0-10 werden in etwa gleich häufig angegriffen. Damit sind zeitlich spätere Perturbierungen für Graph Attention Networks gefährlicher. Die prozentual hohe Verwundbarkeit von GAT-Modellen und der Tatsache, dass die Diskrepanz in der Verteilung der angegriffenen Zeitschritte bei etwa 4% liegt, lässt vermuten, dass sie für beliebige strukturelle Manipulationen empfänglich sind. Das Hinzufügen/Entfernen einer Kante, kann im *Message Passing* Schritt zu einem Sprung in den



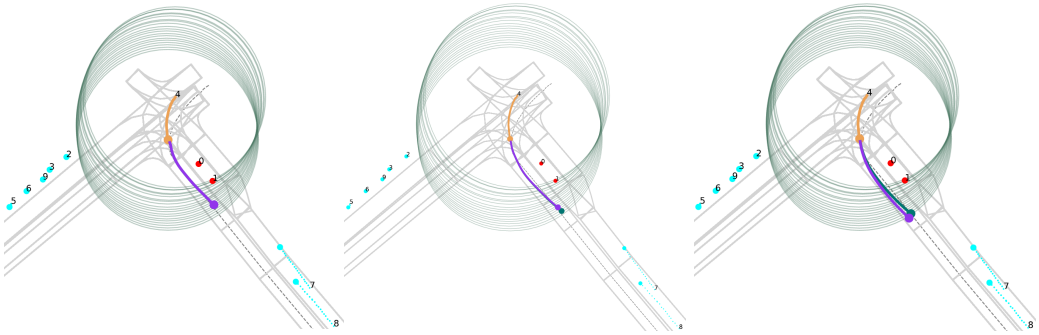
(a)

(b)



(c)

Abbildung 5.5: Direkter Strukturangriff: Verteilung der angegriffenen Zeitschritte: **(a)** GCN, **(b)** GAT-2, **(c)** GAT-4

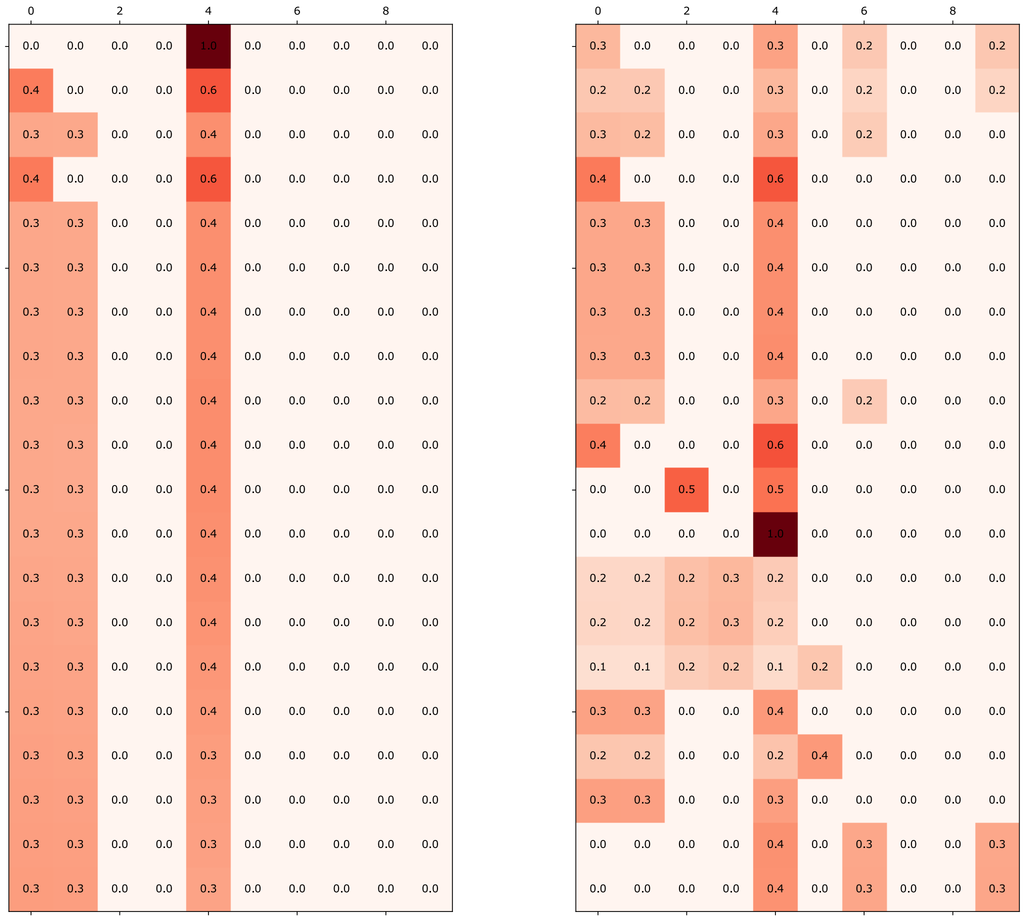


(a) **(b)** **(c)**

Abbildung 5.6: **(a)** GCN: ΔFDE: 0.02 , **(b)** GAT-2: ΔFDE: 1.06, **(c)** GAT-4: ΔFDE: 1.00

aggregierten Features führen. Daher lässt sich vermuten, dass neben späten perturbierten Zeitschritten auch große Änderungen innerhalb der Nachbarschaftsattribute eine relevante Rolle bei diesem Angriffstyp spielen. So führen nicht adjazente und räumlich weit entfernte Knoten zu einem größeren Sprung in der Nachbarschaftsaggregation als nahe.

Ein Vergleich der Abbildungen in 5.6 zeigt, dass die *Attention*-basierten Modelle insbesondere in Szenen leiden, in denen viele Verkehrsteilnehmer sind, die noch keine direkte Verbindung zum Agenten haben. Die Attention-Matrizen in den Abbildungen 5.7 stellen die Aufmerksamkeit vor der Perturbierung und nach dieser vor. Die Manipulationen die sich zu Anfang und Ende der Beobachtungszeit häufen verursachen starke Gewichtsschwankungen.



(a) **(b)**

Abbildung 5.7: GAT-2 Modell Attention Heads vor und nach der Perturbierung **(a)** 1. Attention Head **vor** der Perturbierung **(b)** 1. Attention Head **nach** der Perturbierung

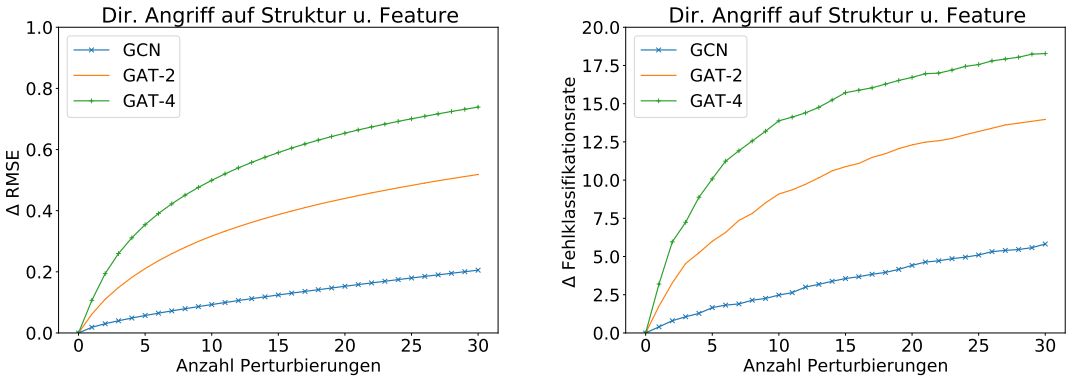
5.1.3 Direkte Angriffe auf Struktur und Features

Direkte Angriffe auf Struktur und Features kombinieren die beiden erzeugten Menge aus den Angriffen 5.1, 5.1.2 und wählt aus der Gesamtmenge diejenige aus, die den Modellfehler maximiert. Die nachfolgende Tabelle 5.3 listet die Erhöhung von Fehler, FDE und Fehlklassifikation auf.

Methode	GCN	GAT-2	GAT-4
Feat Struct Fehler	$\Delta 0,2053$	$\Delta 0,5184$	$\mathbf{\Delta 0,7387}$
Feat Struct FDE [m]	$\Delta 0,38$	$\Delta 0,95$	$\mathbf{\Delta 1,36}$
Feat Struct MR [%]	$\Delta 5,82$	$\Delta 10,77$	$\mathbf{\Delta 18,28}$

Tabelle 5.3: Einfluss von direkten Angriffen auf Struktur und Features auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Für die GAT- Modelle werden die Messwerte aus den separaten Angriffen übertroffen. Dabei ist das Graph Attention Network mit 4 *Attention Heads* weiterhin am empfindlichsten gegenüber Manipulationen. Eine Begründung dafür, dass sich die Werte in dem kombinierten Angriff nicht additiv, je ein Anteil vom Struktur-/Featureangriff, erhöhen ist, dass der Raum, in dem der Modellfehler maximiert werden soll, nicht mehr Eindimensional ist. Es ist möglich, dass sowohl die Struktur, als auch die Knotenattribute, manipuliert werden. Dies kann dazu führen, dass verheerendere Kombinationen erreicht werden. Die Grafiken 5.8 stellen den Einfluss des Angriffsszenarios auf alle Modelle dar.



(a) **(b)**

Abbildung 5.8: Direkter Angriff auf Struktur und Feature. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

Dabei scheidet das GCN-Modell in Bezug auf Robustheit am besten ab. Seine Werte liegen unter denen für direkte Angriffe auf die Attribute des Agenten. Dies liegt daran, dass der *greedy* Algorithmus nicht zwingend die beste Entscheidung treffen kann. Für die Attention-Modelle konnten alle Werte gegenüber den anderen Angriffen gesteigert werden.

Das GAT-4 Modell schneidet dabei um ein vielfaches schlechter ab, als das Modell mit zwei Attention-Heads. Vergleicht man die Aufteilung von Struktur und Featureangriffen, wie in Abbildung 5.9 dargestellt, dann fällt auf, das beide Modellfamilien auf unterschiedliche Angriffstypen reagieren.

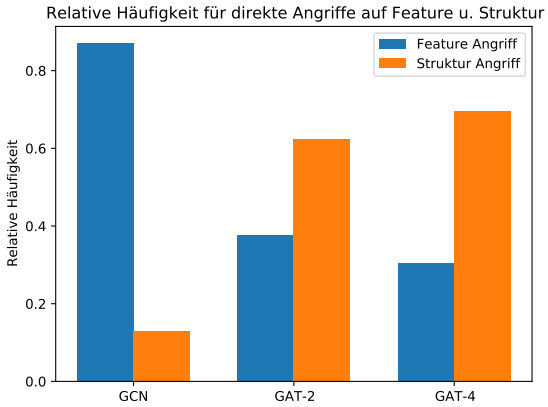


Abbildung 5.9: Relative Verteilung der direkten Struktur bzw. Feature Angriffe für alle Modelle.

Wie in Abschnitt 5.1 und 5.1.2 untersucht reagiert das GCN stärker auf Angriffe auf die Knotenattribute und die GAT Modelle stärker auf Strukturangriffe. Das Säulendiagramm bestätigt beide Erkenntnisse. Insbesondere hebt es hervor, dass das GCN sehr robust gegen räumliche Strukturveränderungen ist, wohingegen die GAT Modelle zu fast $\frac{1}{3}$ von Featureangriffen manipuliert wurden. Weiterhin nimmt der Anteil an Perturbierungen der *target* Attributen mit steigender Zahl an *Attention Heads* ab. Das GAT-4 Modell hat den größten Anstieg der Fehlklassifikationsrate bei gleichzeitig verhältnismäßig wenigen Featureangriffen. Dies unterstützt die These, dass das Graph Attention Network seine Vorhersage stark auf die aggregierten Nachbarfeatures stützt. Die Aufschlüsselung der einzelnen angegriffenen Zeitschritte innerhalb der Adjazenz-Matrix/Attribute liefert keine neuen Erkenntnisse. Die Verteilung entspricht denen aus den vorangegangen Abschnitten 5.1 5.1.2.

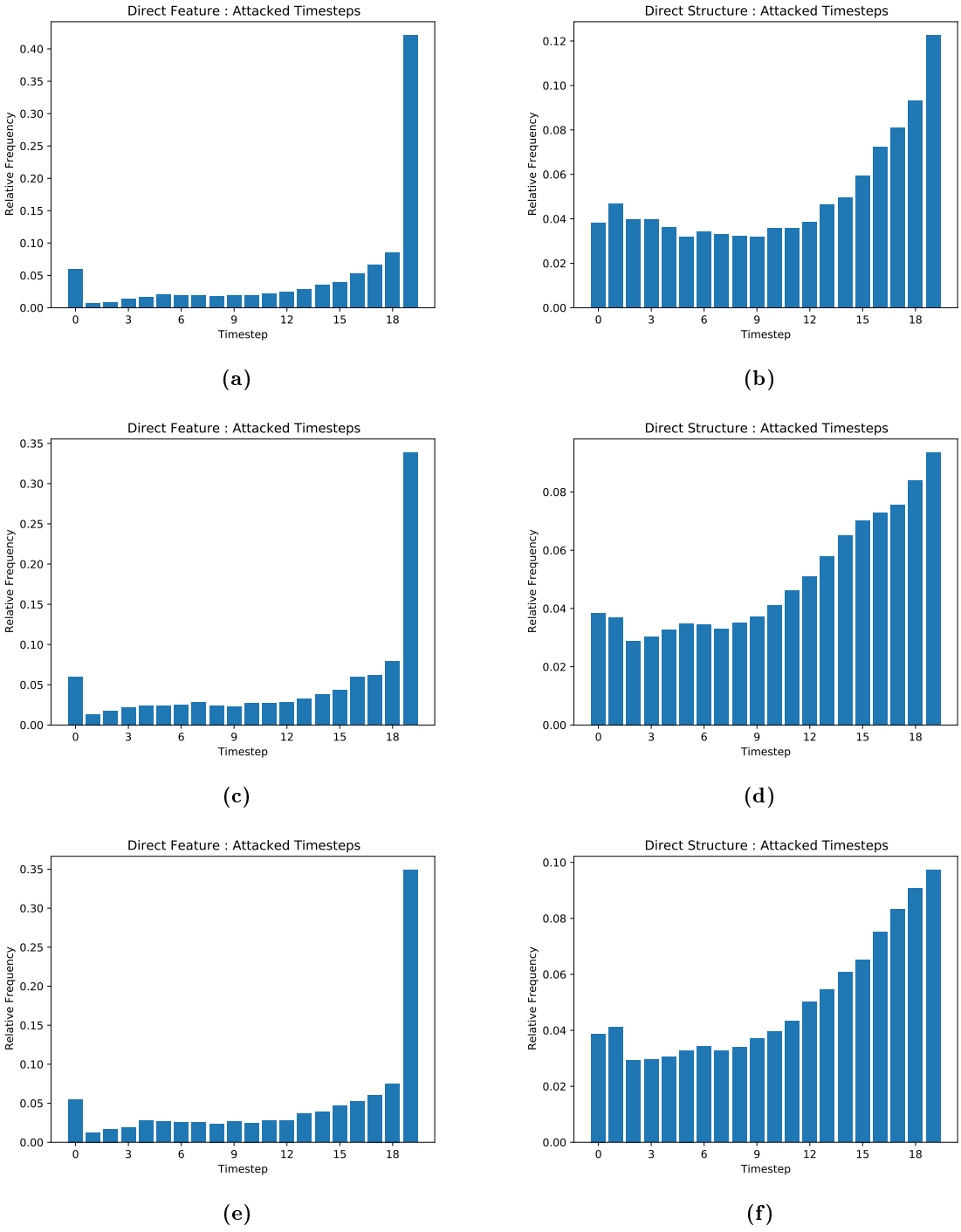
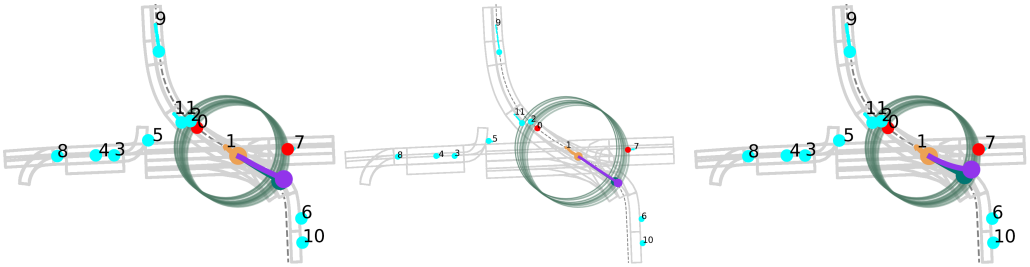


Abbildung 5.10: Direkter Angriff auf Struktur und Features: Verteilung der angegriffenen Zeitschritte: **(a)** GCN Feature, **(b)** GCN Struktur, **(c)** GAT-2 Feature, **(d)** GAT-2 Struktur, **(e)** GAT-4 Feature, **(f)** GAT-4 Struktur

Die Abbildungen in 5.11 stellt die Vorhersage für alle Modelle nach 30-Perturbierungsschritten vor. Das GAT-4 Modell wird am stärksten von den Manipulationen beeinflusst. Die Attention-Matrizen für das GAT-2 Modell in Abbildung 5.12 zeigt, dass Strukturangriffe immer be-



(a) **(b)** **(c)**

Abbildung 5.11: (a) GCN: ΔFDE: 1.64 , **(b)** GAT-2: ΔFDE: 1.21, **(c)** GAT-4: ΔFDE: 3.02

vorzugt werden. Von 30-Perturbierungen wurden drei an der Struktur durchgeführt. Die restlichen haben die Merkmale des Agenten manipuliert.

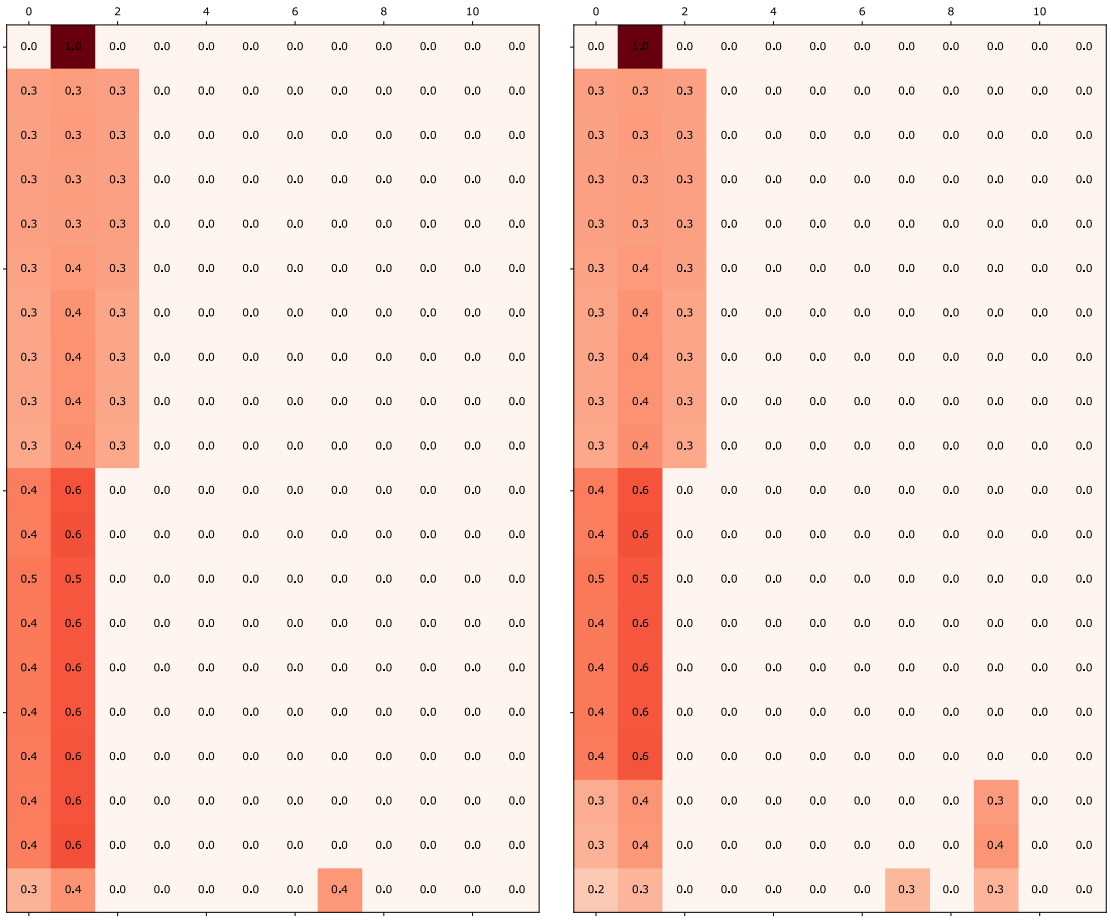


Abbildung 5.12: GAT-2 Modell Attention Heads vor und nach der Perturbierung **(a)** 1. Attention Head **vor** der Perturbierung**(b)** 1. Attention Head **nach** der Perturbierung

5.1.4 Direkte Angriffe auf alle Zeitschritte

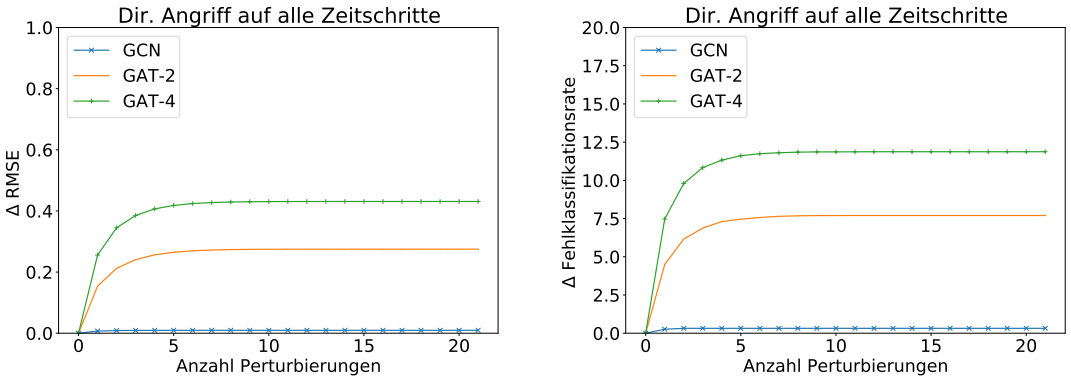
Direkte Angriffe auf Knotenattribute, Verbindungen zwischen dem Agenten und seinen Nachbarn oder deren Kombination greifen alle einen Zeitschritt an. Dabei ist zu beobachten, dass sowohl der erste als auch der zuletzt beobachtete Zeitschritt bevorzugte Ziele sind. Insbesondere das GCN-Modell zeigt sich gegenüber Angriffen auf die Struktur als robust. Um die Annahme zu untersuchen, ob die Modelle Perturbierungn für nur einen Zeitschritt kompensieren können, z.B. durch Mittlung innerhalb der zeitlichen Faltung, werden in einem direkten Angriff auf die Struktur für einen Nicht-Agenten alle Zeitschritte gleichzeitig angegriffen. Für diesen Kandidaten werden alle Verbindungen in der Zeit zum Agenten invertiert. Die so erzeugte Menge an *adversarial examples* wird dem greedy Algorithmus vorgelegt. Die Ergebnisse nach 30- Manipulationen sind in Tabelle 5.4 aufgelistet.

Methode	GCN	GAT-2	GAT-4
All Timesteps Loss	$\Delta 0,0094$	$\Delta 0,2749$	$\mathbf{\Delta 0,4309}$
All Timesteps FDE [m]	$\Delta 0,01$	$\Delta 0,51$	$\mathbf{\Delta 0,80}$
All Timesteps MR [%]	$\Delta 0,32$	$\Delta 7,7$	$\mathbf{\Delta 11,87}$

Tabelle 5.4: Einfluss der direkten Angriffe alle Zeitschritte gleichzeitig auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Das GCN reagiert auf dieses Angriffsszenario mit einer Steigerung von 0.32% auf die Fehlklassifikationsrate, wohingegen die GAT-Modelle ähnlich empfindlich reagieren, wie bei den direkten Strukturangriffen. Das Ergebnis für das GCN-Modell legt die Vermutung nahe, dass es keine relevanten Informationen aus der Nachbarschaft aggregiert und sich das *Seq2Seq* Modell im wesentlichen auf die Features des Agenten verlässt. Die Werte steigen für das GAT-4 nicht so stark wie bei direkten Angriffen auf die Struktur. Die Abbildungen 5.13 liefern einen Erklärungsansatz.

Sowohl die Kurve für den Modellfehler, als auch für die Fehlklassifikationsrate, steigen bei beiden GAT Modellen bis zur vierten Perturbierung steiler an, als die vorausgegangenen Angriffe. Ab diesem Punkt flachen sie abrupt ab und verlaufen parallel zur X-Achse. Dafür gibt es zwei mögliche Erklärungen. Erstens beinhalten die angegriffenen Szenen nicht genügend Verkehrsteilnehmer, sodass bereits nach wenigen Schritten alle Verkehrsteilnehmer angegriffen wurden. Zweitens besteht die Möglichkeit, dass nur eine geringe Menge an Teilnehmern in der Lage ist den Modellfehler zu erhöhen. Wird ein gewisser Schwellwert überschritten, so können weitere Perturbierungen bereits erfolgte Manipulationen kompensieren. Wird die Verbindung zu einem ursprünglichen Nachbarn gekappt, so kann es vorkommen, dass die weitere Manipulation eines Kandidaten diese Verbindung indirekt wiederherstellt. Somit würde der Agent die ursprüngliche Information über Umwege erhalten. Die Vermutung, dass die Manipulation aller Zeitschritte innerhalb der Struktur gefährlicher sind, als einzelne Veränderungen, konnte für die *Attention*-basierten Modelle

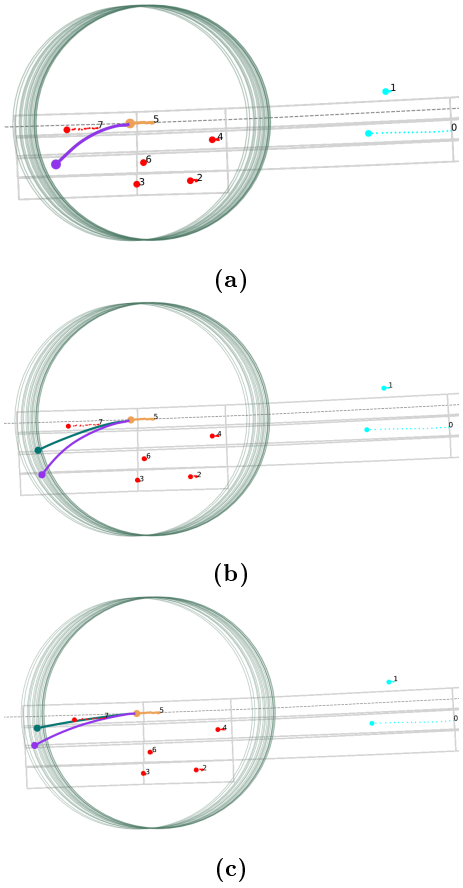


(a) **(b)**

Abbildung 5.13: Direkter Angriff auf alle Zeitschritte. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

nachgewiesen werden. Jedoch fasst dieser Angriffstyp 20 einfache Angriffe in einer Perturbierung zusammen. Um die Effektivität genauer einzugrenzen sollte sie in zukünftigen Untersuchungen mit dem 20-fachen einfacher Strukturangriffe verglichen werden.

Eine Verkehrszene mit mehreren potentiellen *attackern* in den Abbildungen 5.14 zeigt, dass die Manipulation von allen Zeitschritten gleichzeitig das GCN-Modell nicht beeinflusst. Die Abweichung von der ursprünglichen Trajektorie beträgt vier Zentimeter. Für die GAT-Modelle führt diese Art der Perturbierungen zu verheerenden Ergebnissen, die mit Trajektorien in der Gegenfahrbahn enden. Abbildung 5.15 zeigt vollständig neu ausgerichtete Attention-Matrizen nach maximaler Perturbierung. Die Invertierung aller Verbindungen, über die Zeit, hat zu einer neuen Gewichtsverteilung geführt. Gleichzeitig sind neue Merkmale hinzugekommen, die sich räumlich von den zuvor aggregierten Werten unterscheiden.

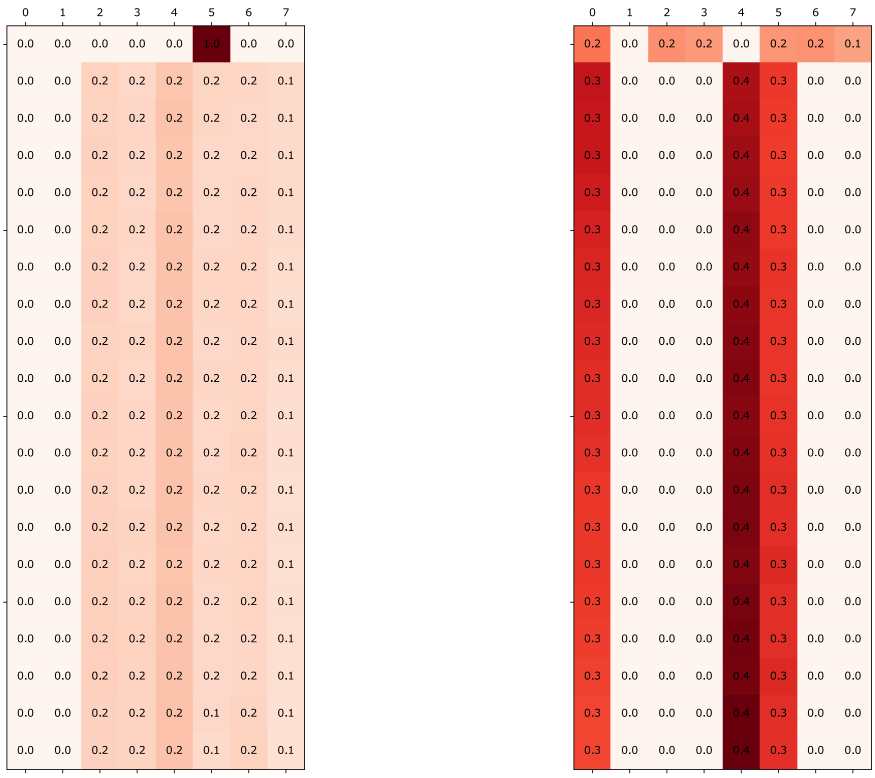


(a)

(b)

(c)

Abbildung 5.14: (a) GCN: ΔFDE: 0.04 , **(b)** GAT-2: ΔFDE: 2.62, **(c)** GAT-4: ΔFDE: 1.58



(a) (b)

Abbildung 5.15: GAT-4 Modell Attention Heads vor und nach der Perturbierung **(a)** 4. Attention Head **vor** der Perturbierung**(b)** 4. Attention Head **nach** der Perturbierung

5.2 Indirekte Angriffe

Indirekte *adversarial attacks* im *Node-level* Szenario für Graph Neural Networks zeichnen sich dadurch aus, dass alle Perturbierungen sich ausschließlich auf Nachbarn des *target* Knoten beziehen. Ziel dieser Angriffe sind alle Attribute der *attacker* sowie Verbindungen von *attackern* zu Nicht-Agenten Knoten. Alle Features des *target* sowie Verbindungen, die von diesem ausgehen, werden bei indirekten Angriffen ignoriert. Enthält eine Szene keine zum Agenten adjazente Knoten, so wird der Angriff vorzeitig beendet.

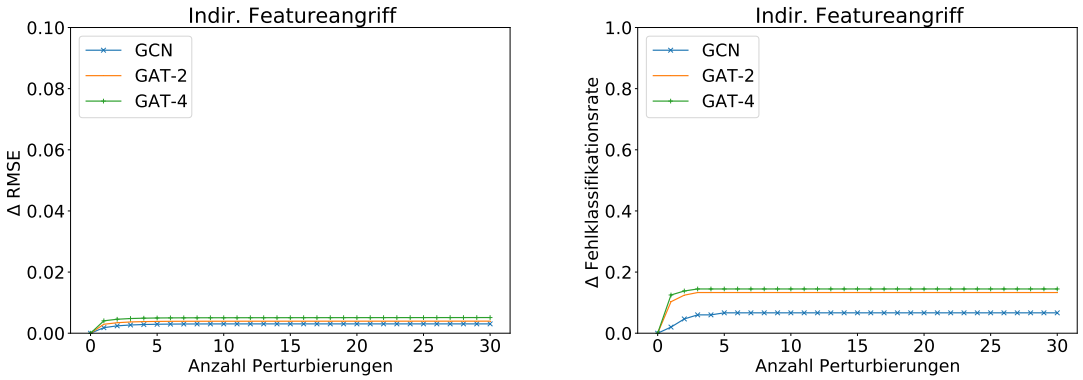
5.2.1 Indirekte Featureangriffe

Indirekte Featureangriffe sind das Gegenstück zu direkten Featureangriffen aus Abschnitt 5.1. Statt den Attributen des Agenten, werden ausschließlich Features der *attacker* manipuliert. Dazu wird in jedem Zeitschritt die Menge der zum *target* adjazenten Knoten bestimmt. Für diese Zeitschritt-spezifische Menge werden die Attribute perturbiert, die als Eingabe für die *greedy adversarial attack* dienen. Das Resultat des Angriffes für 30-Perturbierungen ist in Tabelle 5.5 aufgelistet.

Methode	GCN	GAT-2	GAT-4
Feature Loss	$\Delta 0,003$	$\Delta 0,003$	$\mathbf{\Delta 0{,}005}$
Feature FDE [m]	$\Delta 0,0053$	$\Delta 0,0045$	$\mathbf{\Delta 0{,}0091}$
Feature MR [%]	$\Delta 0,06$	$\Delta 0,13$	$\mathbf{\Delta 0{,}14}$

Tabelle 5.5: Einfluss der indirekten Featureangriffe auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Keines der Modelle reagiert auf indirekte Manipulationen der Knotenattribute mit einer Erhöhung der Fehlklassifikationsrate mit mehr als 0.14%. Dies stellt einen starken Kontrast zu direkten Featureangriffen dar, in dem alle Modelle mit einer Erhöhung der Fehlklassifikationsrate von mindestens 5.18% reagiert haben. Bei diesem Angriffsszenario hat die Robustheit des GCN-Modells stark zugenommen. Hochgerechnet auf den gesamten Datensatz werden bei 0.06% ca. 23 Samples fehlklassifiziert. Den Verlaufskurven folgend müssen dies Sequenzen gewesen sein, die zuvor bereits einen *Final Displacement Error* nahe 2 Metern hatten.



(a) **(b)**

Abbildung 5.16: Indirekter Featureangriff. **Achtung** angepasste Skalen zur besseren Interpretierbarkeit. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

Die Verteilung der angegriffenen Zeitschritte in 5.17 folgt nicht demselben Muster, wie bei direkten Featureangriffen. Für alle Modelle ist weiterhin eine Konzentration auf den ersten und letzten Zeitschritt erkennbar, jedoch sind diese beispielsweise bei GAT-2 und GAT-4 Modell nicht so ausgeprägt.

Für die Verteilung beim GCN ergibt sich eine fast kontinuierliche Verteilung, die ihre Maxima im ersten und letzten Zeitschritt einnehmen. Diese Abweichung von direkten Angriffen und der Tatsache, dass sich dieses Modell kaum beeinflussen lässt, spricht dafür, dass das GCN nicht viel Bedeutung auf die aggregierten Attribute seiner Nachbarn legt. Vergleicht man diese Informationen mit den anderen beiden Modellen, die ein ähnliches Verhalten aufweisen, dann wird diese Annahme dadurch bestärkt, dass sich die Verteilung über die angegriffenen Zeitschritte fast gleichmäßig über den zulässigen Bereich erstreckt (zwischen 2. – 18 bei GAT-2 2. – 15. bei GAT-4). Die letzten beiden Zeitschritte sind bei der Trajektorievorhersage von besonderer Wichtigkeit, da sie den Ausganspunkt sowie die Richtung der Vorhersage vorgeben. Die Angriffsverteilungen weichen von der Struktur aus 5.1 stark ab und ähneln in manchen Bereichen einer Gleichverteilung. Dies legt die Vermutung nahe, dass dieses Angriffsszenario keine präferierten Ziele ausfindig machen konnte und per Zufall *adversarial examples* finden konnte, die den Fehlerterm des Modells erhöhen.

Die Abbildungen in 5.18 vergleichen den Einfluss der *adversarial attack* auf alle Modelle. Die Szene enthält zwei direkte Nachbarn, deren Merkmale in allen Zeitschritten im Agenten aggregiert werden. Dennoch weicht keines der Modelle um mehr als drei Zentimeter von der Vorhersage ab.

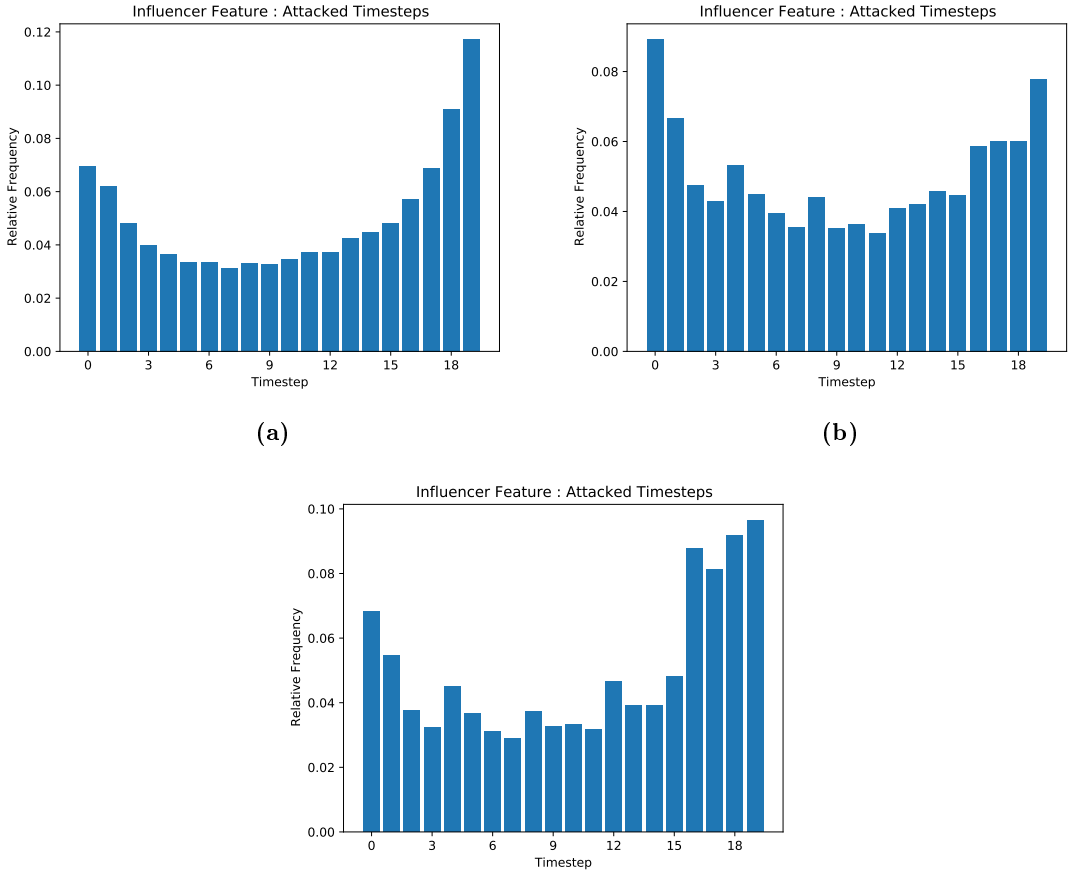


Abbildung 5.17: Indirekter Featureangriff: Verteilung der angegriffenen Zeitschritte: **(a)** GCN, **(b)** GAT-2, **(c)** GAT-4

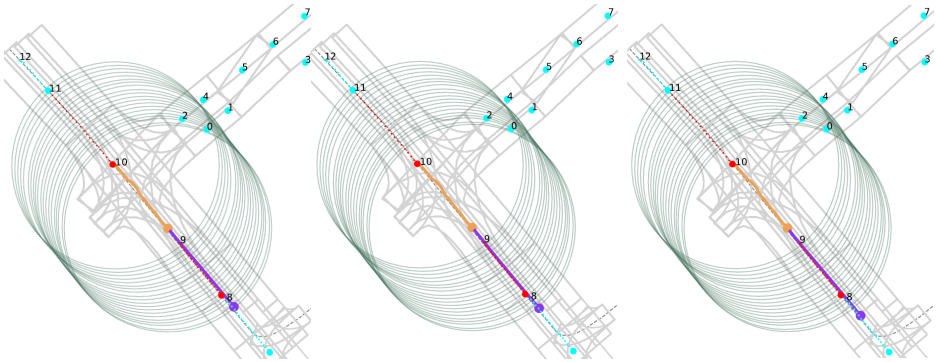


Abbildung 5.18: **(a)** GCN: ΔFDE: 0.01 , **(b)** GAT-2: ΔFDE: 0.00, **(c)** GAT-4: ΔFDE: 0.03

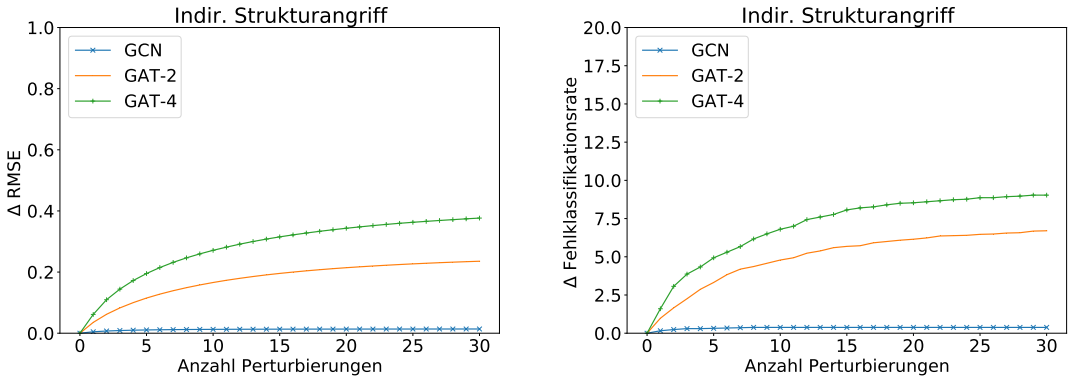
5.2.2 Indirekte Strukturangriffe

Um zu evaluieren wie die vorgestellten Modelle auf Manipulationen ihrer Nachbarn reagieren, werden indirekte Strukturangriffe durchgeführt. Zu jedem Zeitpunkt wird die Menge aller zum Agenten adjazenten *attacker* ermittelt. Für jeden *attacker* und Nicht-Agenten Knoten wird eine Kante hinzugefügt oder entfernt, sofern bereits eine Verbindung existiert. Aus der Menge der *adversarial examples* wird jenes ausgewählt, das den Fehlerterm des Modells maximiert. Tabelle 5.6 stellt die Ergebnisse für die Werte Fehler, FDE und Fehlklassifikationsrate nach 30-Perturbierungsschritten vor.

Methode	GCN	GAT-2	GAT-4
Struct Loss	$\Delta 0,0138$	$\Delta 0,2354$	$\mathbf{\Delta 0{,}3766}$
Struct FDE [m]	$\Delta 0,02$	$\Delta 0,43$	$\mathbf{\Delta 0{,}69}$
Struct MR [%]	$\Delta 0,38$	$\Delta 6,57$	$\mathbf{\Delta 9{,}03}$

Tabelle 5.6: Einfluss der indirekten strukturellen Angriffe auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Das GCN-Modell ist am robustesten und weicht kaum von seiner ursprünglichen *Miss Classification Rate* ab. Wohingegen die GAT-Modelle eine vergleichbare Empfindlichkeit, wie bei direkten strukturellen Angriffen, aufweisen. Dabei ist zu beobachten, dass das GAT-2 Modell empfindlicher auf indirekte als auf direkte Strukturangriffe reagiert. Dies gilt für das Modell mit vier *Attention Heads* nicht. Eine mögliche Erklärung dafür ist, dass der finale *Attention Koeffizient* aus dem Mittel aller Koeffzienten der *Attenion Heads* gewonnen wird. Da die Summe aller Koeffizienten die Summe eins ergibt, kann das hinzufügen/entfernen einer Kante einen Ausreißer hervorrufen, der bei zwei *Heads* stärker ins Gewicht fällt. Die Grafiken in 5.19 für den Fehler und die Fehlklassifikationsrate von $0-30$ Manipulationen weisen eine Abflachung der Verlaufskurven auf.



(a) **(b)**

Abbildung 5.19: Indirekter Strukturangriff. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

Der Verhältnismäßig schwächere Einfluss von indirekten strukturellen Veränderungen gegenüber direkten lässt sich wie folgt interpretieren. Alle Modelle haben drei Graph Convolution Layer, das bedeutet, dass höchstens die Informationen des dritt-nächsten Nachbarn im Agenten aggregiert werden können. Nach einer Faltung sind die gemittelten Informationen des entfernten/hinzugefügten Knoten im *attacker* angekommen. Nach der zweiten Faltung wird das gewichtete Mittel der *Attention Heads* im Agenten aggregiert. Damit sinkt der Einfluss auf den Agenten mit der Entfernung des neuen/gelöschten Knoten.
Die Abflachung der Fehler-, MCR-Kurven weisen darauf hin, dass nach ca. 15-Perturbierungen kaum noch potente *adversarial examples* gefunden werden. Trotz der Verhältnismäßig größeren Menge an indirekten strukturellen *adversarial examples* dienen wenige Verkehrsteilnehmer und Zeitschritte als geeignete Ziele. Eine detaillierte Untersuchung der attackierten Ziele könnte die Frage aufdecken, ob neben dem Zeitschritt auch die relative Position zum *target* eine relevante Rolle spielt. Infolgedessen könnte geschlussfolgert werden, dass die Anzahl der beteiligten Objekte einer Szene eine nicht unerhebliche Rolle für *adversarial attacks* bilden.
Die Verteilungen in den Abbildungen 5.20 der angegriffenen Zeitschritte ähneln stark denen für direkte Angriffe auf die Struktur. Alle Modelle weisen Maxima beim ersten und letzten Zeitschritt auf, wohingegen diese bei den GAT-Modellen nicht so ausgeprägt sind. Der Anteil der angegriffenen Zeitschritte steigt bei *attention* basierten Modellen ab dem 10.-ten Zeitschritt linear an.

Ein Vergleich zu den Ergebnissen in Abschnitt 5.2.1 liefert interessante Einsichten in die Verwundbarkeit der GAT Netzwerke. Werden Attribute der *attacker* manipuliert, so spiegelt sich dies unmerklich in den gemessenen Werten wider. Wird jedoch die Struktur der adjazenten Knoten perturbiert, so zeigt sich, dass GAT-Modelle stark verwundbar sind. Daraus lässt sich schlussfolgern, dass der *attention*-Mechanismus Wert auf die Attribute

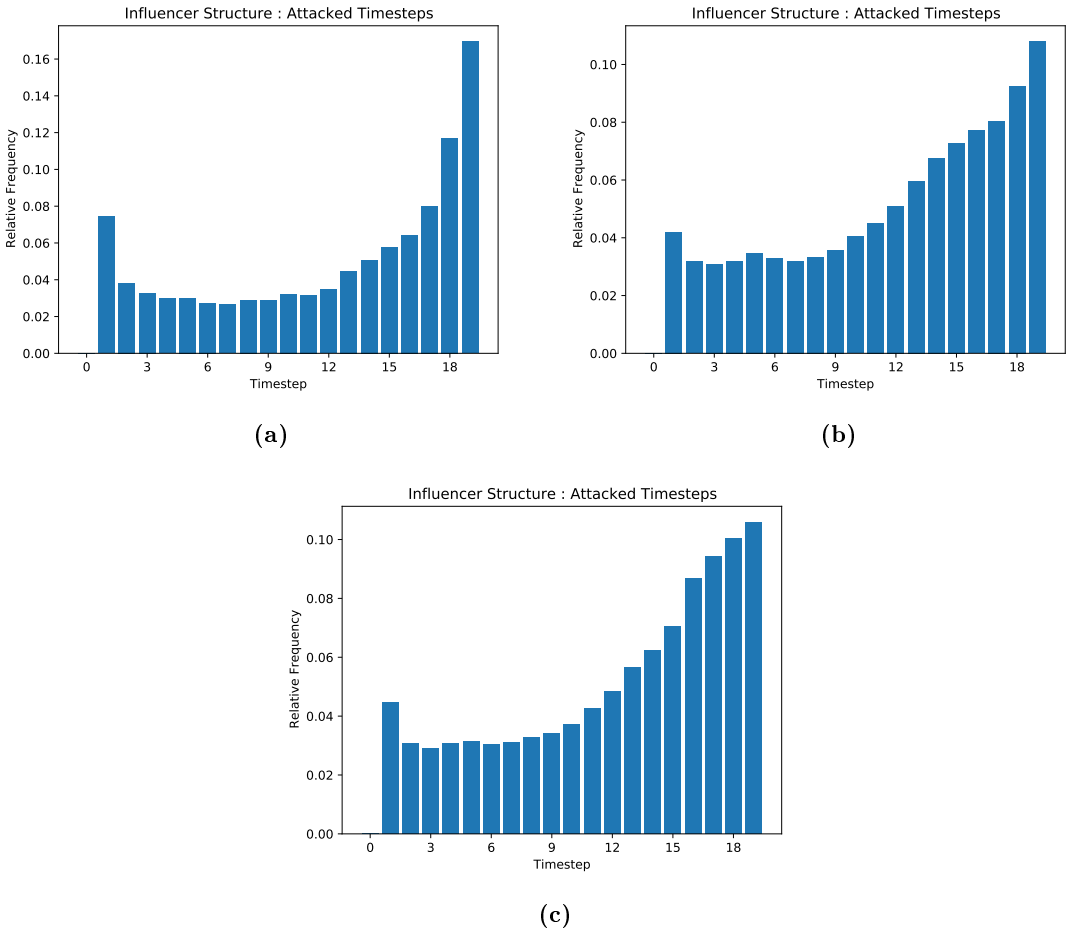
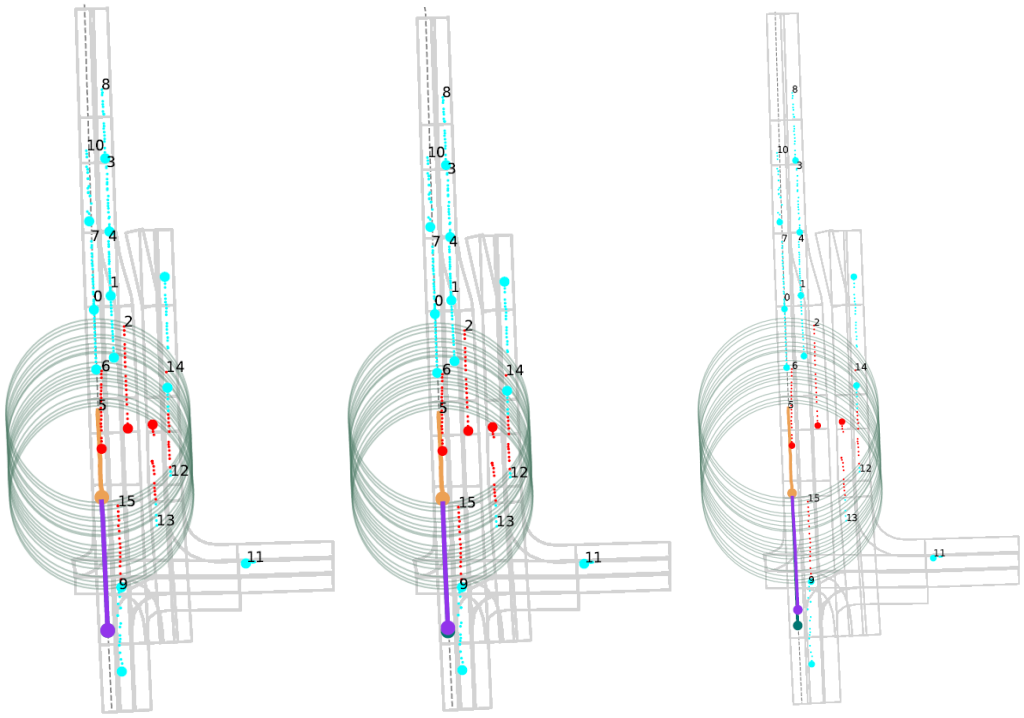


Abbildung 5.20: Indirekter Strukturangriff: Verteilung der angegriffenen Zeitschritte: **(a)** GCN, **(b)** GAT-2, **(c)** GAT-4



(a) (b) (c)

Abbildung 5.21: **(a)** GCN: ΔFDE: 0.02 , **(b)** GAT-2: ΔFDE: 0.60, **(c)** GAT-4: ΔFDE: 3.31

anderer Verkehrsteilnehmer legt und mit Manipulationen der Agenten Attribute in einem gewissen Rahmen zurechtkommt. Die Bedeutung dieser Erkenntnis wird dadurch relativiert, dass Featureangriffe einen Bruchteil der Veränderung hervorrufen, die eine strukturelle Manipulation erreichen kann. Angriffe auf die Features nehmen den Gradienten der Fehlerfunktion bezüglich der Eingabe. Statt einen ϵ Schritt in diese Richtung zu gehen, wird vom Gradienten zuvor das Vorzeichen extrahiert. Manipulationen der Graphstruktur hingegen führen zur plötzlichen Hinzunahme/Entfernung von Knotenattributen, die die Perturbierungen von *FGSM* um ein vielfaches übersteigen. Eine Untersuchung der angegriffenen Knoten kann Aufschluss über das gelernte Konzept der Modelle liefern. Möglicherweise nimmt die relative Richtung, soziale Features und Entfernung eine große Bedeutung ein, die *FGSM* nicht erreichen kann.

Eine Verkehrszene, die sowohl *attacker*, als auch nicht adjazente Knoten enthält ist in Abbildung 5.21 dargestellt. Das Potential dieses Angriffsszenario spiegelt sich im GAT-4 Modell wider. Die *adversarial attack* erhöht den FDE der Vorhersage um 3.31 Metern, wohingegen die Vorhersage des Graph Convolutional Network nahezu unverändert bleibt.

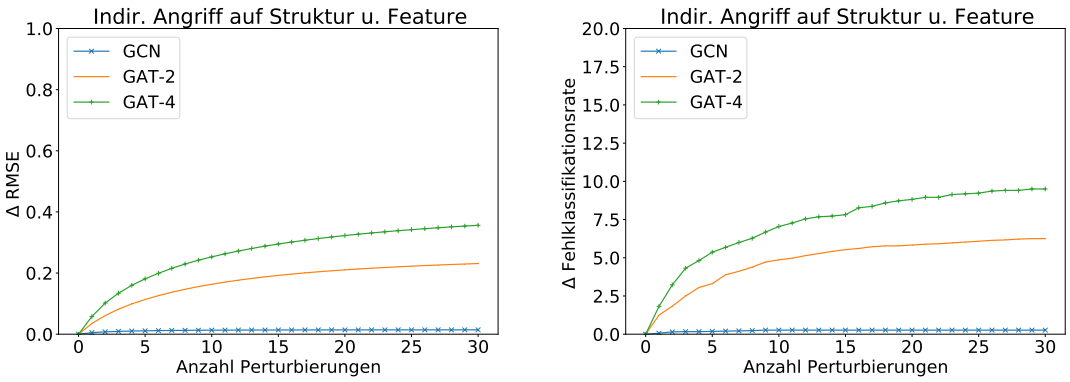
5.2.3 Indirekte Angriffe auf Struktur und Features

Indirekte Angriffe auf Struktur und Features kombinieren die zuvor vorgestellten Szenarien aus 5.2.1 und 5.2.2. Die Mengen der *adversarial examples* werden zusammengelegt. Aus der resultierenden Menge wird der Angriff ausgewählt, welcher den Modellfehler maximiert. Der gewählte Angriff dient als Grundlage für die nächste Perturbierung.

Methode	GCN	GAT-2	GAT-4
Feat Struct Loss	$\Delta 0,0142$	$\Delta 0,2311$	$\mathbf{\Delta 0{,}3561}$
Feat Struct FDE [m]	$\Delta 0,02$	$\Delta 0,42$	$\mathbf{\Delta 0{,}66}$
Feat Struct MR [%]	$\Delta 0,26$	$\Delta 6,24$	$\mathbf{\Delta 9{,}5}$

Tabelle 5.7: Einfluss von direkten strukturellen u. Feature Angriffe auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Tabelle 5.7 repräsentiert den Einfluss von indirekten Angriffen auf Struktur und Features auf die *attacker* des Agenten. Die Fehlklassifikationsrate des GCN steigt um knapp ein viertel Prozent. Damit ist es das robusteste unter den betrachteten Modellen. Die Attention-basierten Netzwerke weisen vergleichbare Werte wie bei indirekten strukturellen Angriffen auf. Dies liegt unter anderem daran, dass der Algorithmus in mehr als 90% der Fälle Angriffe auf die Struktur ausübt (vgl. Grafik 5.23). Damit degeneriert der Angriff auf beide Eingabe zu einem indirekten Angriff auf die Struktur.



(a) **(b)**

Abbildung 5.22: Indirekter Angriff auf Struktur und Features. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

Die Verhältnismäßigkeit für das GCN repräsentiert einen ähnlich nachvollziehbaren Sachverhalt, in der Angriffe auf die Struktur bevorzugt werden. Die Fehlklassifikationsrate für indirekte Angriffe auf die Struktur liegt leicht über der für indirekte Angriffe auf die Knotenattribute.
Analog zu diesen Ergebnissen lassen sich die Verlaufskurven für die Modelle in den Gra-

fiken 5.22 interpretieren. Eine steigende Anzahl an Manipulationen hat einen geringen Einfluss auf die Vorhersagegenauigkeit des GCN-Modells. Für die GAT Netzwerke nimmt der Fehler und die Fehlklassifikationsrate, wie in Abschnitt 5.2.2, zunächst stark zu und nimmt nach ca. der 5.-ten Perturbierung einen immer kleinere Steigung ein. Eine mögliche Begründung folgt analog zu indirekten Strukturangriffen, dass die Anzahl und Verteilung der Verkehrsteilnehmer einen erheblich Einfluss auf die Menge der potenten *adversarial examples* hat. Einerseits steigt der Raum an möglichen Perturbierungen und somit die Menge an erfolgreichen *adversarial attacks*. Andererseits steigt damit die Diversität an Verkehrsteilnehmern, die sich weit vom Agenten entfernt aufhalten oder andere Fahrtrichtungen aufweisen. Das aggregieren von hinzugekommenen/entfernten Features durch die Struktur hat eine größere Magnitude als das manipulieren von Attribute durch *FGSM*.

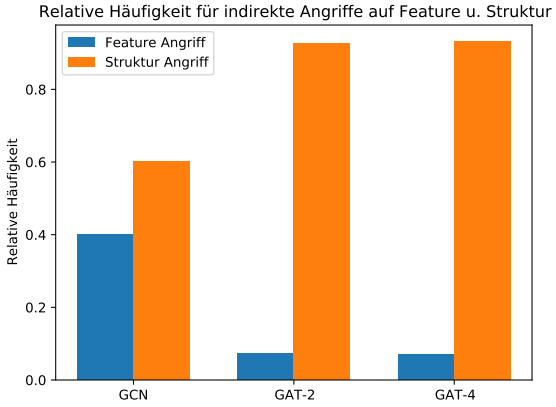


Abbildung 5.23: Relative Verteilung der indirekteen Struktur bzw. Feature Angriffe für alle Modelle.

Eine Gegenüberstellung der angegriffenen Zeitschritte auf die Knotenattribute und die Graphstruktur sind in den Abbildungen 5.24 dargestellt. Die Aufteilung für das GCN-Modell folgt denselben Verteilungen, wie sie in den einfachen Angriffsszenarien diskutiert werden. Ebenfalls folgt die Verteilung der Strukturangriffe für die GAT-Modelle denen für isolierte indirekte Strukturangriffe. In der Verteilung für die Featureangriffe ist kein klares Muster erkennbar. Die ursprüngliche Beobachtung, dass der erste und letzte Zeitschritt bevorzugte Ziele sind gilt hier nicht. Unter Anbetracht der relativ geringen Anzahl an Manipulationen an *attacker* Attributen scheinen diese zufällig gewählt worden zu sein. Eine Untersuchung des Datensatzes kann dabei helfen zu verstehen, wo der Ursprung für diese Diskrepanz und Gleichverteilung der angegriffenen Featurezeitschritte liegt. Werden Strukturangriffe im indirekten Szenario bevorzugt, die auf spezifische Attributskonstellationen reagieren, so sollte sich das Verhältnis von Feature- und Strukturangriffen in Szenen umkehren, die nicht komplex sind. Beispielsweise einseitige befahrene Straßen mit wenigen Verkehrsteilnehmern.

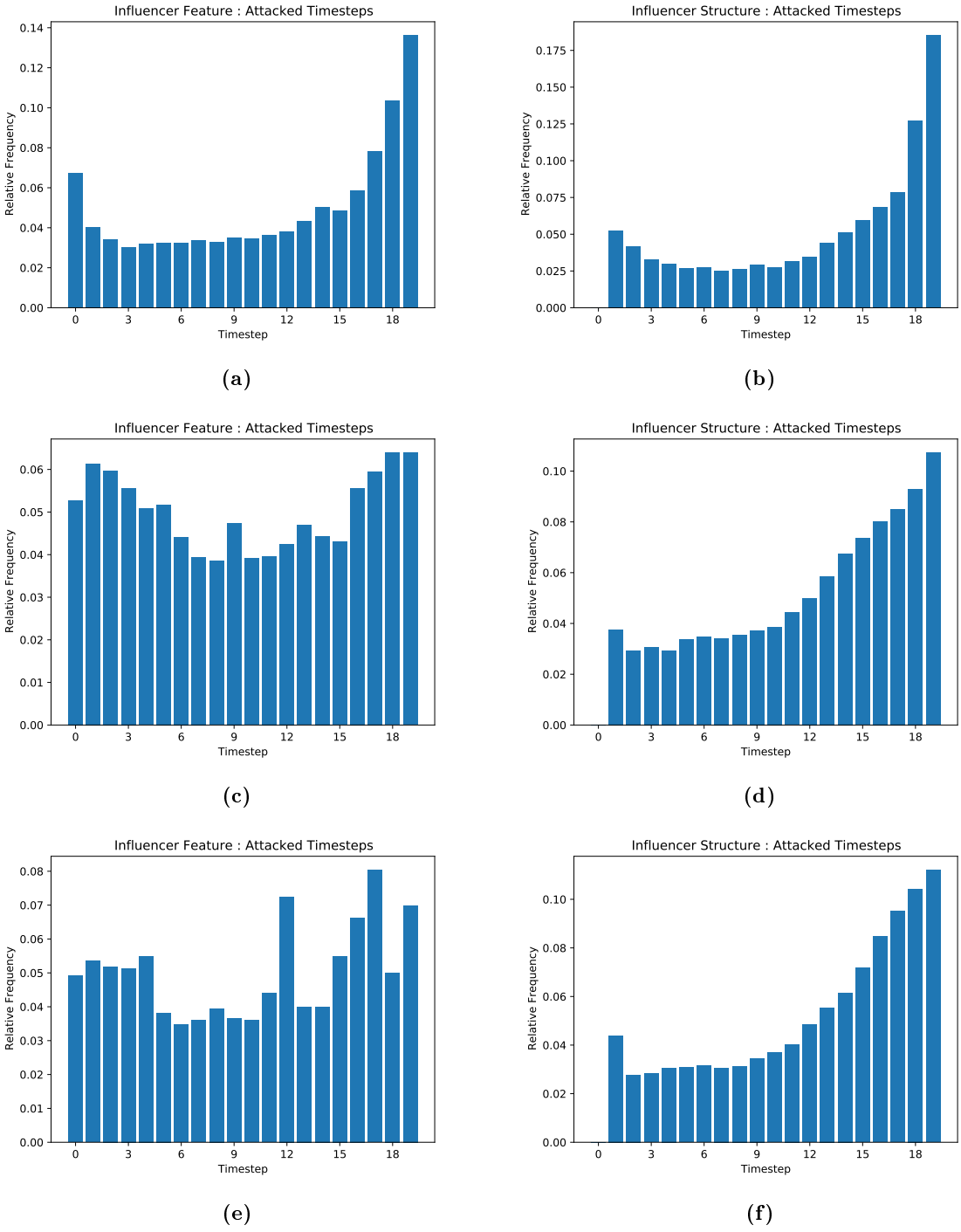
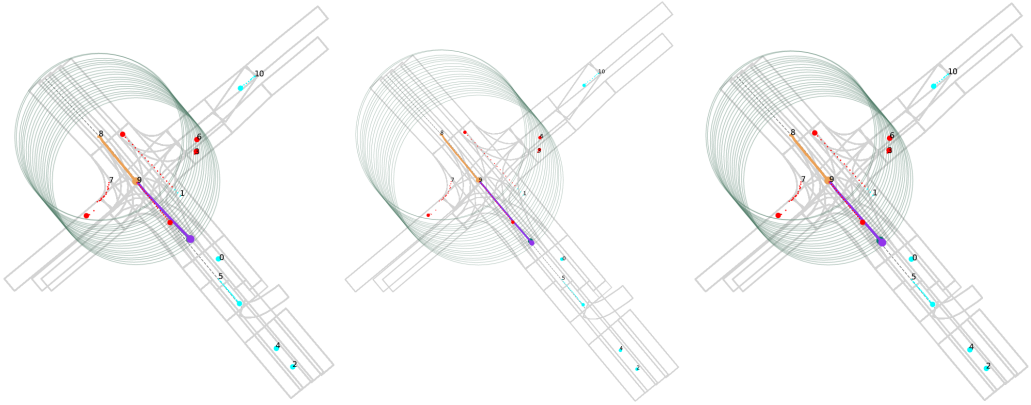


Abbildung 5.24: Indirekter Angriff auf Struktur und Features: Verteilung der angegriffenen Zeitschritte: **(a)** GCN Feature, **(b)** GCN Struktur, **(c)** GAT-2 Feature, **(d)** GAT-2 Struktur, **(e)** GAT-4 Feature, **(f)** GAT-4 Struktur

Die Abbildungen in 5.25 Verkehrsszenen enthalten *attacker* und nicht adjazente Knoten, die für beide Angriffsvarianten Ziele bieten. Der Einfluss des Angriffes steigt mit der Komplexität des Modells. Ausgehend davon, dass bei den *Attention*-basierten Modellen



(a) **(b)** **(c)**

Abbildung 5.25: (a) GCN: ΔFDE: 0.13 , **(b)** GAT-2: ΔFDE: 0.57, **(c)** GAT-4: ΔFDE: 0.67

vorwiegend strukturelle Manipulationen durchgeführt wurden, scheinen diese zu einer Verlängerung der Vorhersage zu führen. Beim GCN-Modell ist stattdessen eine Abweichung der Trajektorie auf die Gegenfahrbahn zu beobachten.

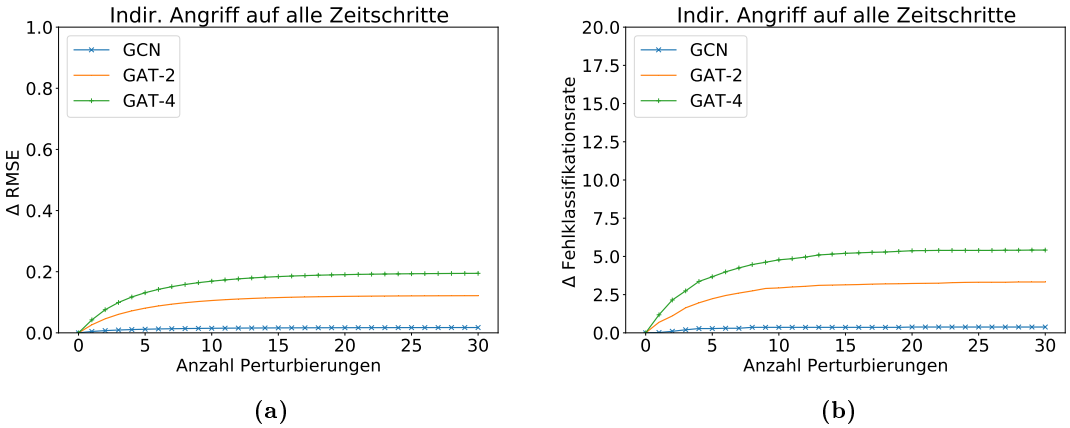
5.2.4 Indirekte Angriffe auf alle Zeitschritte

Um, wie bei direkten Angriffen auf alle Zeitschritte der Graphstruktur, zu überprüfen, ob der erste bzw. letzte Zeitschritt bevorzugt perturbiert wird, da diese aufgrund fehlender zeitlicher Nachbarn nicht kompensiert werden können, wird analog überprüft wie Robust die Modelle in Bezug auf indirekte Angriffe auf alle Zeitschritte sind. Zu diesem Zweck wird in jedem Zeitschritt die Menge der adjazenten *attacker* Knoten bestimmt. Für jeden dieser Knoten werden die Nicht-Agenten Verbindungen in allen Zeitschritten invertiert. Unter der Annahme, dass ein Agent dieselben Nachbarn über mehrere Zeitschritte hinweg hat, pflanzt sich der verursachte Fehler in der Zeit fort. Fällt ein *attacker* im Verlauf der Zeit weg, so hat die invertierte Verbindung keinen Einfluss mehr auf die Trajektorie des *targets*. Die Ergebnisse dieses Angriffes sind in Tabelle 5.8 dargestellt.

Methode	GCN	GAT-2	GAT-4
All Timesteps Loss	$\Delta 0,0173$	$\Delta 0,1215$	**Δ0,1946**
All Timesteps FDE [m]	$\Delta 0,03$	$\Delta 0,22$	**Δ0,36**
All Timesteps MR [%]	$\Delta 0,38$	$\Delta 3,32$	**Δ5,41**

Tabelle 5.8: Einfluss der indirekten strukturellen Angriffe auf den Fehler, FDE und Miss Rate aller Modell nach 30 Perturbierungsschritten.

Bei strukturellen Manipulationen aller Zeitschritte ist das GCN am robustesten. Die Fehlklassifikationsrate steigt um weniger als einen halben Prozent. Die GAT-Modelle weisen eine höhere Empfindlichkeit auf. Alle gemessenen Werte steigen mit Anzahl der *Attention Heads*. Die Abbildungen 5.26 zeigen, dass die Fehlklassifikationsrate für GAT-2 bzw. GAT-4 nach wenigen Perturbationen einen nahezu stabilen Wert bei 3.32% bzw. 5.41% einnehmen.



(a) **(b)**

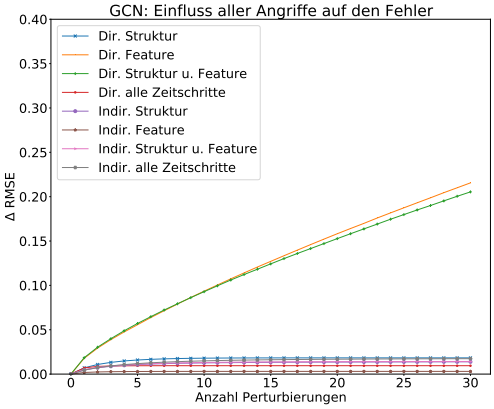
Abbildung 5.26: Indirekter Angriff auf alle Zeitschritte. **(a)** Anstieg des RMSE für alle Modelle. **(b)** Anstieg der Fehlklassifikationsrate (MCR) für alle Modelle.

Die Invertierung aller Verbindungen über die Zeit reduziert den Raum möglicher *adversarial examples* drastisch. Daneben ist nicht garantiert, dass der Agent über alle Zeitschritte hinweg dieselben Nachbarn hat, wodurch nicht alle invertierte Kanten einen Einfluss auf die Vorhersage haben. Verfügen die Szenen über insgesamt wenige Teilnehmer, die zwei Schritte vom Agenten entfernt sind und wechseln diese *attacker* häufig über die Zeitschritte, so können die invertierte Verbindungen nicht ihr volles Potential entfalten. Zusammengenommen können diese Gründe die Konvergenz der Verlaufskurven für den Fehler und die Fehlklassifikationsrate erklären.

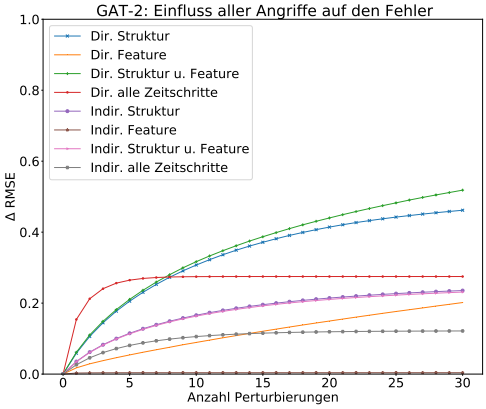
5.3 Gegenüberstellung

Die empirische Evaluation der Robustheit von Graph Neural Networks hat gezeigt, dass keines der untersuchten Modelle gegen Manipulationen der Eingabe geschützt ist. Darüber hinaus konnte gezeigt werden, dass die Architektur einen großen Einfluss auf die Empfindlichkeit gegenüber bestimmten Angriffsszenarien hat. Die Grafiken in 5.27 und 5.28 vergleichen alle Angriffsszenarien für je ein Modell und stellen den wachsenden Modellfehler bzw. Fehlklassifikationsrate von $0 - 30$ Perturbierungen dar.
Der Verlauf der Miss Classification steigt nicht so gleichmäßig, wie der Modellfehler. Dies liegt daran, dass der *RMSE* das Modell darauf trainiert eine Trajektorie vorherzusagen, die jeden Zeitschritt gleich stark bewertet. Ein erhöhter *RMSE*-Wert kann demnach bedeuten, dass sowohl der erste Zeitschritt sowie der letzte gleich weit (oder alle Punkte dazwischen) von der wahren Trajektorie entfernt sind. Da die Fehlklassifikationsrate auf Basis des *Final Displacement Error* mit mehr als 2 Metern definiert ist, kann der *RMSE* steigen, ohne dass die Fehlklassifikationsrate steigt. Daher lässt sich vermuten, dass die Effektivität der *adversarial attacks*, in Bezug auf den MCR, auf allen Modell steigerbar ist, indem das Bewertungskriterium durch eine Funktion ersetzt wird, die den *FDE* stärker gewichtet.

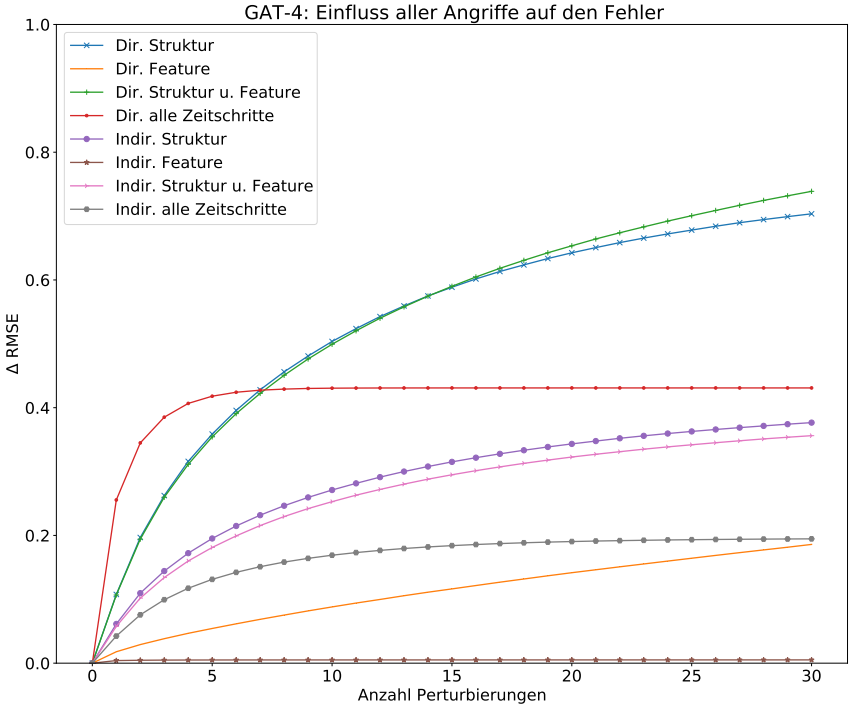
Ein Vergleich der Angriffe auf das Graph Convolution Netzwerk sind in 5.27a und 5.28a dargestellt. *Adversarial attacks* die die Knotenattribute des Agenten direkt verändern haben den größten Erfolg das Modell zu täuschen und die Fehlklassifikationsrate zu steigern. Trotz des Zusammenhangs zwischen *RMSE* und der *MCR* ist bei beiden Messwerten ein simultaner Anstieg zu beobachten. Sowohl indirekte Angriffe jeder Art, wie auch direkte Angriffe auf die Struktur, beeinflussen die Werte im Vergleich dazu geringfügig. Die naheliegende Schlussfolgerung, dass das GCN gegen diese Art von Angriffen robust ist, wird durch die Tatsache in Frage gestellt, dass indirekte Perturbierungen der *attacker* Features die Messwerte nur geringfügig erhöhen. Graph Convolutions erzeugen *higher level* Features, indem in einem Knoten die Attribute seiner Nachbarn aggregiert werden. Der Austausch von adjazenten Verkehrsteilnehmern oder das perturbieren ihrer Features führt zu keiner Überschreitung der Messwerte, aus der sich schlussfolgern ließe, dass das *Seq2Seq* Modell bessere Vorhersagen auf den aggregierten Attributen tätigen kann. Die Ergebnisse der Evaluation für das GCN-Modell rufen die Frage hervor, ob das Netzwerk aus der Graphstruktur lernt und diese ausnutzt, um die Vorhersage der Bewegung zu unterstützen. Bevor weitere Aussagen zur Robustheit von GCN im Bereich der *node-level regression* getätigt werden können, sollte untersucht werden, inwieweit die Graph Convolution in den Verarbeitungsprozess involviert sind. Ein naiver Ansatz wäre das Modell zu duplizieren. Das erste Modell trifft die Vorhersage wie zuvor. Beim zweiten werden unmittelbar die Attribute des Agenten für alle Zeitschritte extrahiert und ohne Umwege in das *Seq2Seq* überführt, wo sie zur Vorhersage der Trajektorie benutzt werden. Ein direkter Vergleich



(a)



(b)

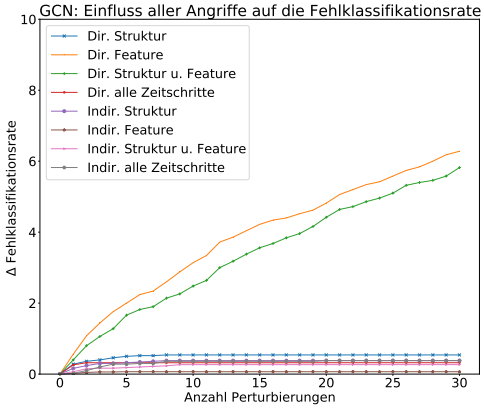


(c)

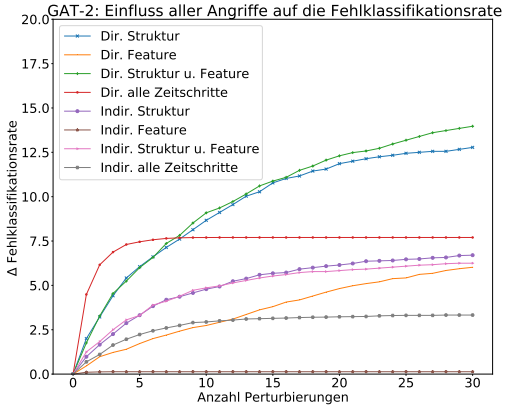
Abbildung 5.27: Verlauf des Modellfehlers zwischen $0-30$ Perturbierungen. **(a)** GCN **Achtung** angepasste Skala zur besseren Interpretierbarkeit, **(b)** GAT-2, **(c)** GAT-4

der Ausgaben beider Modelle kann Aufschluss darüber geben, ob das Modell aus der räumlichen und zeitlichen Struktur der Verkehrsszene zusätzliche Informationen gewinnt.
Die *Attention*-basierte Modelle weisen in nahezu allen Angriffsszenarien eine höhere Empfindlichkeit für Manipulationen auf als das Graph Convolution Netzwerk. Angriffe die Veränderungen der Graphstruktur beinhalten verursachen einen höheren Schaden als reine Angriffe auf Attribute. Eine mögliche Erklärung liegt der Faltungsformel vom GCN zugrunde. Im GCN nach N. Kipf und M. Welling werden Nachbarn durch die normalisierte Laplace-Matrix aggregiert. Das Gewicht der Einträge dieser Matrix hängt vom *Degree* jedes Nachbarn ab. Je mehr Nachbarn ein adjazenter Knoten hat, desto geringer ist sein normalisiertes Gewicht. Dadurch können zwei Situationen entstehen, die einen nachteiligen Effekt auf die Vorhersage der Trajektorie haben. Erstens kann es passieren, dass die Attribute eines Nachbarn mit einem sehr geringen Gewicht aggregiert werden, obwohl diese eine hohe Relevanz für die Vorhersage haben. Zweitens, durch die Tatsache das alle Modelle mit variabler Verkehrsteilnehmerzahl trainiert wurden, werden dieselben Szenen, mit unterschiedlicher Anzahl an Objekten, mit unterschiedlichen Gewichten aggregiert. *Attention*-basierte Modelle umgehen diese Probleme, indem sie für jeden adjazentn Knoten ein Gewicht vergeben, welches unabhängig vom *Degree* des Knotens ist. Somit können sie in wiederkehrenden Szenen, mit unterschiedlicher Teilnehmerzahl, dieselben Gewichte vergeben. Daraus folgt das GAT-Modelle empfindlicher auf strukturelle Veränderungen reagieren können, insbesondere dann wenn das Gewicht für einen entfernten/hinzugefügten Knoten vergleichsweise hoch ist und sich in allen *Attention Heads* repliziert. Ein direkter Vergleich zwischen den GAT-2 und GAT-4 Modell in 5.28b bzw. 5.28c unterstützen diese Vermutung. Alle Angriffe auf das GAT-4 Modell führen zu einer höheren Fehlklassifikationsrate als beim GAT-2 Modell. Direkte Featureangriffe bilden eine Ausnahme. Die finale Differenz des Einflusses ist für das GAT-2 Modell um 0.84% höher. Die spontane Hinzunahme/Entfernung von Knoten führt zu einem sprunghaften Anstieg/Abstieg der aggregierten Attribute. Die Begründung, dass mehr *Attention Heads* diese Umverteilung negativ begünstigen gilt für direkten Featureangriffe nicht.

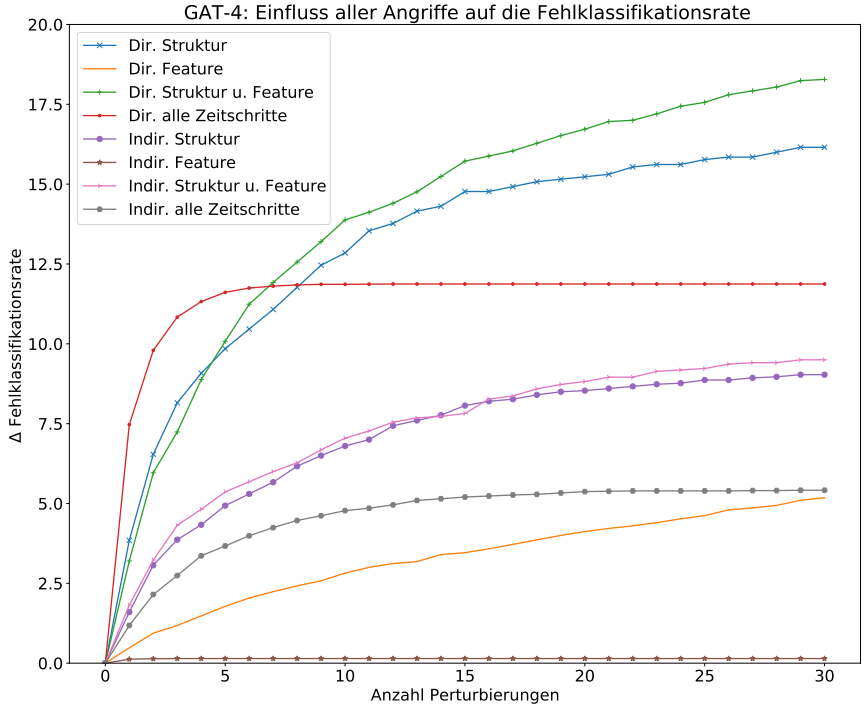
Unter der Annahme, dass mehr *Attention Heads* dazu in der Lage sind die relativen Positionen, Richtungen und sozialen Features anderer Verkehrsteilnehmer besser zu differenzieren, lassen sich damit die Beobachtungen bzgl. struktureller und direkten Featureangriffen begründen. Mehr *Heads* verteilen das Gewicht auf Nicht-Agenten Knoten, wodurch die Attribute des *targets* in den Hintergrund rücken. Die Vorhersage des Modells wird stärker von umgebenden Verkehrsteilnehmer geleitet, als von der Position des Agenten. Weitere empirische Experimente mit wachsender Zahl an *Attention Heads* können Einblicke in das kontraintuitive Verhalten liefern. Daneben ist der kaum vorhandene Einfluss von indirekten Featureangriffen bemerkenswert. Sowohl Feature-, als auch Strukturangriffe, resultieren in manipulierten aggregierten Agentenattributen. Die Annahme, dass indirekte Manipulationen von *attacker*-Attributen einen vergleichbaren Effekt erzielen wie indirekte Angriffe



(a)



(b)



(c)

Abbildung 5.28: Verlauf der Fehlklassifikationsrate zwischen $0 - 30$ Perturbierungen. **(a)** GCN **Achtung** angepasste Skala zur besseren Interpretierbarkeit, **(b)** GAT-2, **(c)** GAT-4

auf die Struktur konnte nicht nachgewiesen werden. Dieses Ergebnis lässt sich dadurch deuten, dass das Hinzufügen/Entfernen von Knoten Sprünge in den aggregierten Attributen hervorruft, wohingegen *FGSM* kleine Perturbierungen an bereits bestehenden Knoten durchführt. Um diese Hypothese zu untersuchen könnten weitere Tests mit variierenden *FGSM* ϵ-Parameter durchgeführt werden, der die Größte der Perturbierung steuert.
Zusammengefasst ist keines der untersuchten Modelle robust bezüglich *adversarial examples*. Insgesamt dominieren direkte Angriffe auf die Struktur und Knotenattribute bezüglich der gemessenen Werte. Jedoch konnte bei keinem Modell sogenannte *blind spots* aufgedeckt werden. Keine der durchgeführten Perturbierungen führte zu einem sprunghaften Abweichung von der ursprünglich vorhergesagten Trajektorie. Eine empirische Evaluation auf einem begrenzten Testdatensatz ist jedoch kein Beweis für die Abwesenheit von *blind spots*. Zu diesem Zweck müssen die Modelle Zertifiziert werden, sodass mathematisch nachweisbar ist, dass keine Perturbierung zu einer Abweichung führt, die einen definierten Rahmen überschreitet. Jedoch deckt der direkte Vergleich aller Modelle auf, dass die verwendete Modellarchitektur einen entscheidenden Einfluss auf *adversarial attacks* hat. Unter der Annahme, dass alle Modelle relevante *higher level* Graphfeatures lernen, konnte gezeigt werden, dass eine Manipulation der Graphstruktur nicht zwangsläufig zu einer Verzerrung der Vorhersage führt. Ebenso wenig führt die Perturbierung von Attributen zu einem allgemein verwendbaren Angriffsmodell gegen Graph Neural Networks.

Kapitel 6

Fazit

Die zentrale Frage dieser Arbeit ist es empirisch zu untersuchen wie robust Graph Neural Networks gegenüber *adversarial attacks* sind. *Deep Learning* basierte Verfahren können dazu beitragen den Straßenverkehr nachhaltig sicherer zu gestalten. Dabei führen komplexere Straßensituation im innerstädtischen Bereich häufiger zu Verkehrsunfällen. Durch die Fähigkeit die Bewegung naher Verkehrsteilnehmer zu antizipieren, können gefährliche Verkehrssituationen im Vorhinein vermieden werden. Damit die Vorhersagen selbst nicht zu einer Verschärfung der Gefahrensituation führt, muss zu diesem Zweck die zuverlässige Vorhersage zu jedem Zeitpunkt gewährleistet sein. Das bedeutet, dass korrupte Sensoren, oder bewusst manipulierte Sensordaten, nicht zu Vorhersagen führen, die nicht nachvollziehbar sind. Zur Erfüllung dieser Ziele sollen mehrere neuronale Netze entwickelt werden, die dazu in der Lage sind zeitreihen- und graphstrukturierte Daten zu verarbeiten, und die Trajektorie von ausgewählten Agenten für einen definierten Zeitraum vorherzusagen. Die trainierten Modelle dienen als Evaluationsgrundlage für eine Palette von *adversarial attacks*. Die Angriffsszenarien erstrecken sich über alle möglichen Eingaben und werden auf die trainierten Modelle angewendet unter Ausnutzung aller Modelldaten, wie trainierte Gewichte und Hyperparameter. Zusätzlich sollen feine, grobe und gezielte Angriffe die Diversität an möglichen Fehlern simulieren, um die Bedeutung spezifischer Angriffe besser untersuchen zu können. Dabei wird kein Wert darauf gelegt wie *unmerklich* die Veränderungen für Mensch und Maschine sind. Um die Aussagekraft der Ergebnisse zu erhöhen werden neben diversen Angriffsszenarien verschiedene Graph Neural Network Architekturen trainiert und untersucht.

Um die Netze in der Anwendungsdomäne der Zeitreihenanalyse zu trainieren wurde das *supervised learning* ausgewählt. Unter Abwägung verschiedener Kriterien, wie Datensatzgröße, Diversität der aufgenommen Verkehrssituationen und öffentliche Verfügbarkeit wurde der *Argoverse* Datensatz ausgewählt. Der öffentlich zugängliche Datensatz verfügt über mehreren Tausend Trainings- und Testdaten, die typische Verkehrsszenen aus den Amerikanischen Großstädten *Miami* und *Pittsburgh* beinhalten. Die fünf Sekunden langen Se-

quenzen wurden so aufgeteilt, dass die ersten zwei Sekunden die Eingabe für das Modell bilden, wohingegen die letzten drei Sekunden die wahre Ausgabe. Die Daten wurden zusätzlich so aufbereitet, dass sie die normalisierten Positionen und soziale Features aller Verkehrsteilnehmer enthalten, die zu mindestens 65% in allen aufgenommen Zeitschritten enthalten sind. Fehlende Datenpunkte wurden interpoliert. Aus diesem Datensatz lassen sich in der Zeit dynamische Graphen extrahieren, wobei jeder Graph in einem Zeitpunkt wiedergibt, welche Verkehrsteilnehmer in einem Umkreis von 20 Metern Sichtkontakt zueinander haben. Mit Hilfe des erweiterten Datensatzes wurden drei Graph Neural Networks trainiert, die auf den zwei Architekturtypen für *Graph Convoltuion* und *Graph Attention* basieren. Alle Modelle teilen sich drei Convolutional Layer, wobei jede Schicht die Daten einmal zeitlich und räumlich faltet. Die räumliche Faltung ist austauschbar und entspricht entweder dem Typen *Graph Convolution* oder *Graph Attention*. Die durch die Convolutions extrahierten Graphfeatures werden an ein rekurrentes *Sequence to Sequence* Modell weitergeleitet, welches regressiv die zukünftige Trajektorie eines ausgewählten Agenten vorhersagt. Ein viertes Netz, ohne Graph Convolution oder Graph Attention, wurde trainiert, um zu zeigen, dass *higher-level* Graphfeatures relevante Informationen aus dem sozialen Kontext der Szene aggregieren können. Dadurch, dass das *Sequence to Sequence* Modell seine Vorhersage nicht ausschließlich auf die Attribute des Agenten stützt, kann sichergestellt werden, dass auf Graphen abgestimmte *adversarial attacks* nicht fälschlicherweise den Eindruck von robusten Graph Neural Networks hervorrufen, falls diese keine Veränderung der Ausgabe hervorrufen. Die Modelle, eines mit Graph Convolution und zwei mit *Attention*-Mechanismen, mit 2 bzw. 4 *Attention Heads*, wurden anschließend unter acht Angriffsszenarien evaluiert. Es wurde die Veränderung der Fehlerfunktion, der *Final Displacement Error* und die Fehlklassifikationsrate gemessen und unter allen Modelle verglichen. Zur Bestimmung der Robustheit von Graph Neural Networks wurden separat die Manipulation von Knotenattribute, Graphstruktur , als auch die kombinierte Manipulation untersucht. Die Angriffsmöglichkeiten wurden weiterhin in die Möglichkeit aufgeteilt sowohl das Ziel der Vorhersage, den *target*-Knoten direkt anzugreifen, als auch seine adjazente Nachbarn, die *attacker* zu manipulieren, um den Agenten *indirekt* zu täuschen. Jeder Angriff hat nur einen Zeitschritt der Zeitreihe zum Ziel, sodass zusätzlich in-/direkte Angriffe auf die Graphstruktur, so ergänzt wurden, dass sie alle Verbindungen des dynamischen Graphen auf einmal perturbieren können. Diese Messung stellt sicher, dass das Modell nicht in der Lage ist veränderte Information durch die zeitliche Faltung zu relativieren.
Die Ergebnisse der empirischen Untersuchung belegen, dass kein Modell gegen alle *adversarial attacks* geschützt ist. Bei keinem Modell wurde für ein Angriffsszenario zu einer Manipulation eine unverhältnismäßige Veränderung der Ausgabe beobachtet, jedoch lässt sich nach 30 minimalen Veränderungen der Graphstruktur eine Erhöhung der Fehlklassifikationsrate von nahezu 20% für das Graph Attention Netzwerk mit vier *Heads* beobachten. Die Ergebnisse lassen den Schluss zu, dass *Attention*-basierte Modelle der Struktur Infor-

mationen entnehmen, die für das nachfolgende *Sequence to Sequence* Modell von Relevanz sind. Die Erkenntnisse beruhen auf der Tatsache, dass Angriffe die sowohl direkte als auch indirekte Manipulationen der Struktur vornehmen einen erheblichen negativen Einfluss auf die Messwerte der Graph Attention Netzwerke haben, wohingegen das Graph Convolutional Network kaum von seiner ursprünglichen Vorhersage abweicht. Der *Message Passing* Schritt in Graph Neural Networks dient dazu die Attribute benachbarter Knoten zu aggregieren, somit beherbergen Angriffe auf Knotenattribute dasselbe Potential wie Manipulationen an der Struktur. Jedoch lässt sich diese Auswirkung von strukturellen Angriffen auf Attention Networks nicht für Perturbierungen der Features nachweisen. Im Gegenteil wurde festgestellt, dass bei steigender Anzahl der *Attention Heads* der Einfluss auf Feature Manipulationen sinkt. Dies ruft die Forschungsfrage hervor, ob dieser Modelltyp die extrahierten *higher-level* Features auf die *attacker* Knoten stützt, statt sich auf die Agenten Attribute zu verlassen. Für Modelle mit dem Graph Convolutional Architekturtyp weisen direkte Angriffe, die dazu in der Lage sind die Attribute des *targets* zu manipulieren, starkes Potential auf die Fehlklassifikationsrate des Modells zu erhöhen. Dieser steigt nach 30 Manipulationen der Agentenattribute um 6.28%. Gegenüber allen anderen Angriffsszenarien erweist sich dieser Modelltyp als äußerst Robust, was sich in einem Anstieg der Fehlklassifikationsrate von weniger als einem Prozent widerspiegelt. Die Ergebnisse werden durch die Tatsache in Frage gestellt, dass direkte Angriffe auf die Struktur eine vergleichbar geringe Veränderung der Messwerte hervorrufen. Dies erweckt den Verdacht, dass sich das *Sequence to Sequence* Modell ausschließlich auf die Attribute des Agenten konzentriert, als auf die aggregierten *higher-level* Graphfeatures. Die normalisierte Laplace-Matrix, die Teil der Faltungsformel des Graph Convoltuional Network ist, ist nicht in der Lage dieselben Kantengewichte für gleiche Verkehrsszenen zu verteilen, in denen die Teilnehmerzahl variiert. Dies wirft die Forschungsfrage auf, ob dieser Modelltyp relevante Informationen extrahieren kann, in denen die Situation unverändert bleibt, die Zahl der Knoten jedoch variiert.

Die Annahme, dass Manipulationen aller Verbindungen im dynamischen Graphen für einen Perturbationsschritt einen größeren Effekt erzielen, lässt sich für die *Attention*-basierten Modelle teilweise bestätigen. Zwar konvergieren die Verlaufskurven für den Modellfehler und der Fehlklassifikationsrate nach wenigen Perturbierungen, jedoch platzieren sich direkte Angriffe dieses Typs mit einer vergleichsweise starken Steigung auf Platz drei der effektivsten Manipulationen.

Die durchgeführten Experimente liefern Aufschlüsse über die Robustheit von diversen Graph Neural Network Architekturen und ihrem Einsatz bei *transduktiven* Lernverfahren. Die Abwesenheit von Sprunghaften Veränderungen der Ausgabe, bei kleinen Manipulationen der Eingabe, ist kein Garant für die Robustheit eines Neuronalen Netzes. Zum einen spiegelt der Datensatz lediglich einen unvollkommenen Ausschnitt der Realität dar, zum anderen bilden die durchgeführten *adversarial attacks* keine umfängliche Gewissheit, dass

nicht andere Manipulationen der Eingabe oder Struktur verheerendere Folgen haben können. Eingebettet in den Kontext der aktuellen Forschung leistet diese empirische Arbeit aufschlussreiche Erkenntnisse, die zur *Härtung* zukünftiger Modelle beitragen können. So können die entworfenen Angriffsszenarien verwendet werden, um den Trainingsdatensatz mit manipulierten Samples zu erweitern, sodass die Modelle lernen die Richtige Vorhersage selbst dann zu treffen, wenn die Eingabe sie täuscht. Zum anderen liefern die Resultate einen Ausgangspunkt, um den Fragen nachzugehen, warum *Attention*-basierte Modelle empfindlich gegenüber Manipulationen der Struktur sind und Graph Convoltional Modelle ausschließlich auf direkte Featureangriffe reagieren. Die Klärung dieser Fragen kann Beiträge zur Vorbereitung der Trainings- und Testdaten, Training der Modelle sowie Robustheit gegenüber *adversarial attacks* leisten. Unabhängig davon liefert diese Arbeit einen Beitrag auf die Frage, welche Deep Learning Architekturen sich für welche Aufgabendomäne eignen. So scheint der *Message Passing* Schritt des Graph Convolutional Modells zu steif zu sein, um gleiche Szenen mit wechselnden Kontext, relevant zu erfassen. Des Weiteren erreichen die erstellten Modelle zum Erstellungszeitpunkt dieser Arbeit konkurrenzfähige Vorhersage bezüglich des *Argoverse* Datensatzes. Die neuartige Verwendung von Graph Neural Networks mit der Kombination von zeitlicher Faltung und dem *Attention*-Mechanismus zur Erstellung von *higher-level* Features können zur Weiterentwicklung von Zeitreihenanalysen beitragen, die sich als Graphen darstellen lassen.

Kapitel 7

Ausblick

Die fundamentalen Grundlagen dieser Arbeit bilden die Graph Neural Networks, mit deren Hilfe die Bewegung eines Agenten im urbanen Straßenverkehr vorhergesagt werden kann, sowie acht *adversarial attacks*, die dazu dienen die Robustheit der zuvor erstellen GNN zu untersuchen. Aufbauend auf diesen beiden Bausteinen, sowie den daraus gewonnen Ergebnissen, lässt sich die Arbeit in mehrere Richtungen weiterentwickeln.
Es besteht die Möglichkeit die Performanz der erstellten Modelle durch weitere Ergänzungen und Untersuchungen zu steigern. Alle Modelle verfügen über drei Faltungseinheiten, die eine räumliche und zeitliche Faltung durchführen. Zudem teilen sich die Schichten keine Gewichte. Diese Designentscheidung beruht im wesentlichen auf der Inspiration *Grip++*. Eine oberflächliche Analyse des *Argoverse* Datensatzes zeigt, dass Szenen über viele und weit verteilte Verkehrsteilnehmer verfügen können. Mit drei Convolutional Schichten ist damit nicht gewährleistet, dass auch Informationen von Nachbarn mit einfließen, die mehr als drei Schritte entfernt sind. Durch unabhängige Gewichte weist jede Schicht einer separaten Nachbarschaft ihre eigene Aufmerksamkeit zu, statt den gesamten Kontext zu lernen. Durch weitere Untersuchungen mit variierender Anzahl an Faltungseinheiten mit geteilten/unabhängigen Gewichten kann die Überlegenheit dieser Architektur neu ausgelegt werden. Analoges gilt für die zeitliche Faltung. Diese aggregiert in jedem Knoten seine zeitlichen Nachbarn in unmittelbarer Vergangenheit und Zukunft auf. Dieser Wahrnehmungshorizont ließe sich dadurch steigern, dass jedem Verkehrsteilnehmer die Möglichkeit gegeben wird auf seine gesamte Vergangenheit zurückzublicken. Indem in der *temporalen* Adjazenz-Matrix, statt Verbindungen zu direkten zeitlichen Nachbarn aufzubauen, gerichtete Kanten zu allen zeitlichen Vorgängern geknüpft werden. Aufbauend auf den Ergebnissen zu *Attention*-basierten Modellen lässt sich das Prinzip der gewichteten räumlichen Faltung auf die *temporale* Faltung übertragen. Auf diese Weise wird dem Modell die Möglichkeit eingeräumt, statt jedem vergangenen Zeitschritt gleich gewichtet mit einzubeziehen, jeden Zeitpunkt individuell stark zu aggregieren. Die so gewonnenen *higher-level* Features fügen sich stärker an die Verarbeitungsstruktur des *Sequence to Sequence*-Modells, welches

iterativ die Vorhersage aus vorherigen Beobachtungen aufbaut. Statt jeden Zeitschritt als gleich gewichtetes Mittel seiner zeitlichen Nachbarn in das *Sequence to Sequence*-Modell zu überführen, würde eine Zeitreihe übertragen werden, die in jedem Zeitpunkt alle vergangenen Informationen in einer *hidden representation* mit sich führt. Eine grobe Durchsicht der erstellten räumlichen Adjazenz-Matrizen hat ergeben, dass Verkehrsteilnehmer die an gegenüberliegenden Seiten einer Kreuzung stehen, nicht erfasst werden. Indem die Daten erneut vorverarbeitet werden, mit einem erhöhten Sichtradius, lassen sich dichtere Adjazenz-Matrizen erstellen, die einen größeren sozialen Kontext erfassen. Dies kann dem Modell dabei helfen bessere Vorhersage zu treffen, welches sich anhand der erfassten Verkehrsteilnehmer orientiert.
Die Ergebnisse der *adversarial attacks* werfen einige Fragen auf, deren Aufklärung Beiträge zum Verständnis von Graph Neural Networks leisten können. Die in der Evaluation aufgedeckte Empfindlichkeit von *Attention*-basierten Modellen gegenüber strukturellen Manipulationen steigt mit der Zahl der *Attention Heads*, wohingegen die Robustheit gegenüber direkten Angriffen auf die Agenten Attribute zunimmt. Durch weitere Untersuchungen der Architektur, mit variierender Anzahl an *Heads* können Grundlagen zur Härtung dieser Modelltypen geschaffen werden. Analog weist das Graph Convolutional Network robuste Eigenschaften gegenüber allen Angriffen auf, die keine direkten Featureangriffen implizieren. Um sicherzustellen, dass das gesamte Modell relevante Graphfeatures lernt, statt diese zu übergehen müssen weitere Studien durchgeführt werden. Andernfalls können die Ergebnisse ein fälschlicherweise robusten Architekturtypen vortäuschen. Einen Ansatzpunkt bildet die normalisierte Laplace-Matrix, welche das Aggregationsgewicht anhand des *Degrees* der Nachbarn festlegt. Eine Durchsicht der Trainingsdaten hat ergeben, dass wiederholt gleiche Kreuzungssequenzen aus unterschiedlichen Perspektiven, mit variierender Teilnehmerzahl auftauchen. Die resultierenden räumlichen Adjazenz-Matrizen unterscheiden sich in der Folge stark, was das GCN dazu veranlassen kann Gewichte zu lernen, die sich untereinander aufheben. Die erstellten *adversarial attacks* könnten dazu beitragen die Verwundbarkeit der Modelle durch sogenannten *adversarial training* zu reduzieren. Indem mit Hilfe der Angriffe neue Samples erzeugt werden, die dem Trainingsdatensatz hinzugefügt werden, kann das Modell lernen die richtige Vorhersage unter manipulierten Bedingungen zu lernen. Weiterführende Arbeiten können damit den Einfluss von *adversarial training* auf Graph Neural Networks empirisch nachweisen. Ebenso lässt sich das bestehende Angriffsmodell dazu umbauen *unmerkliche* Perturbierungen der Eingabe zu erstellen. Diese Art der Angriffe stellen bewusste Manipulierungen dar, mit dem Ziel das System zu täuschen und gleichzeitig weder vom Menschen, noch vom System, erkannt zu werden. Auf diese Weise lassen sich die Ergebnisse von fehlerhaften Sensoren auf bewusst feindselige Handlungen verallgemeinern. Letzteres zu erkennen stellt eine besondere Herausforderung dar und ist mindestens genauso wichtig für die Sicherheit im Straßenverkehr.
Neben weiteren Untersuchungen am Vorhersagemodell und der Plausibilität bzw. Nach-

vollziehbarkeit der Robustheit von Graph Neural Networks, lässt sich das Repertoire der *adversarial attacks* erweitern. Die bisherigen Szenarien untersuchen den Einfluss auf die Vorhersage durch Veränderungen an der räumlichen Adjazenz-Matrix und den Knotenattributen. Die in allen Modellen integrierte zeitliche Faltung mit Hilfe der temporalen Adjazenz-Matrix wurde nicht perturbiert. Insbesondere in der Zeitreihenanalyse mittels Graph basierten Netzwerken bieten temporale Faltungen einen Angriffsvektor. Durch fortführenden Studien, die sowohl Verbindungen aus der zeitlichen Adjazenz-Matrix hinzufügen/entfernen oder diese durch variable Gewichte ersetzen, kann ermittelt werden welchen Einfluss Angriffe auf die Zeit auf die Vorhersage haben. Zusätzlich invertieren die strukturellen Angriffe auf alle Zeitschritte der räumliche Adjazenz-Matrix alle Verbindungen, unter der Annahme, dass dies die größte Veränderung herbeiführt. Unter Berücksichtigung der Tatsache, dass räumliche und zeitliche Faltungen die Informationen von Fahrzeugen in alle Richtungen propagieren können, ist es plausibler, dass der Modellfehler steigt, wenn Verkehrsteilnehmer vollständig aus einer Szene entfernt bzw. hinzugefügt werden.

Der Quellcode zum Vorbereiten der Daten, trainieren der Modelle und zum Durchführen der *adversarial attacks* sind unter folgenden Links zugänglich:

- Vorbereiten der Daten: https://github.com/BuggyFairy/prepareGNN.git
- Trainieren und Angreifen der Modelle: https://github.com/BuggyFairy/attackGNN.git

Anhang A

Weitere Informationen

(a) **(b)**

Abbildung A.1

Anhang B

Trajektorien ohne Perturbierung

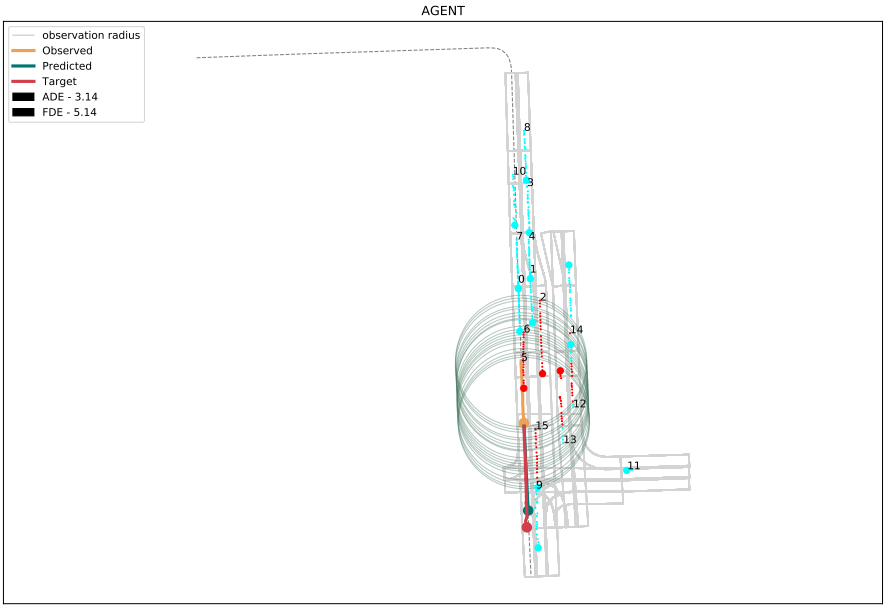


Abbildung B.1: Vorhersage des Seq2Seq Modells.

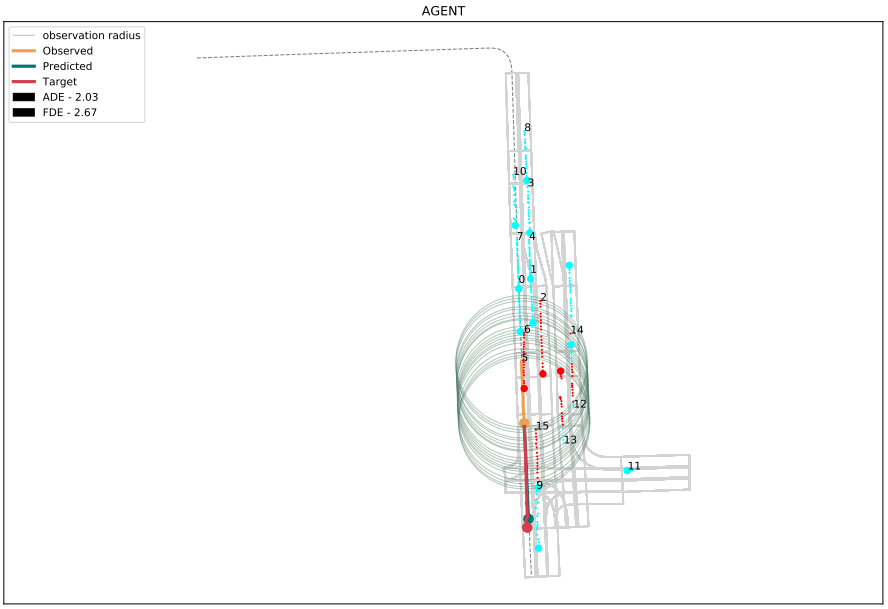


Abbildung B.2: Vorhersage des GCN Modells.

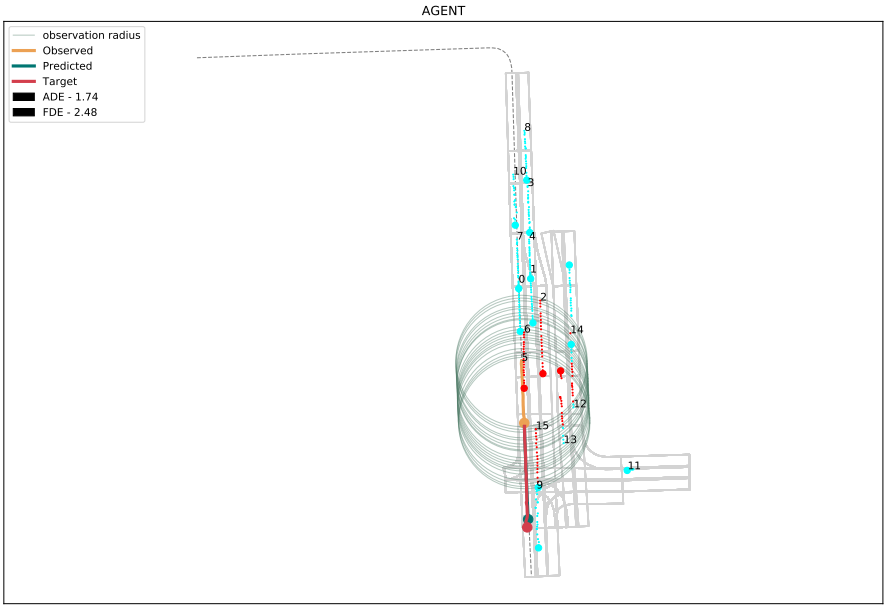


Abbildung B.3: Vorhersage des GAT-2 Modells.

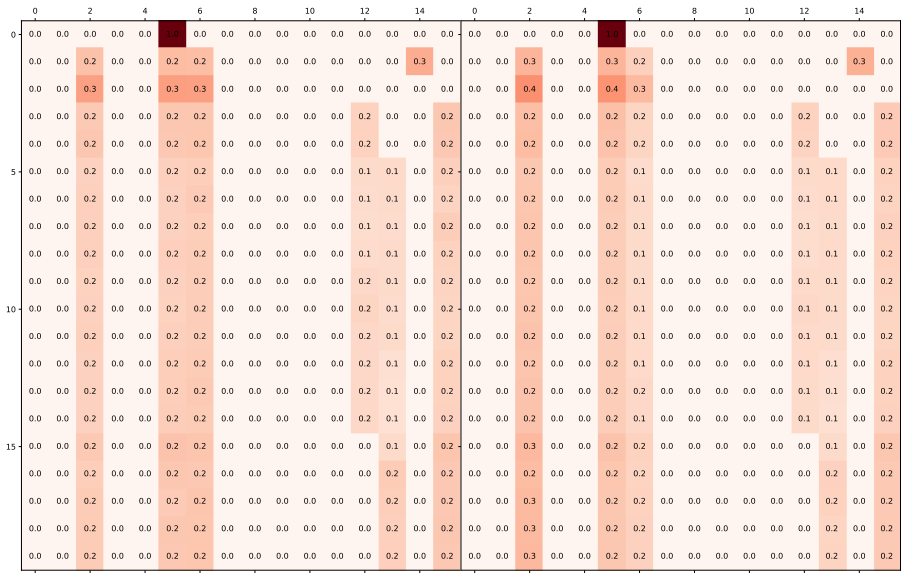


Abbildung B.4: Attentionmatrix für GAT-2. Jede Zeile entspricht einem Zeitschritt. Jede Spaltennummer dem entsprechenden Verkehrsteilnehmer.

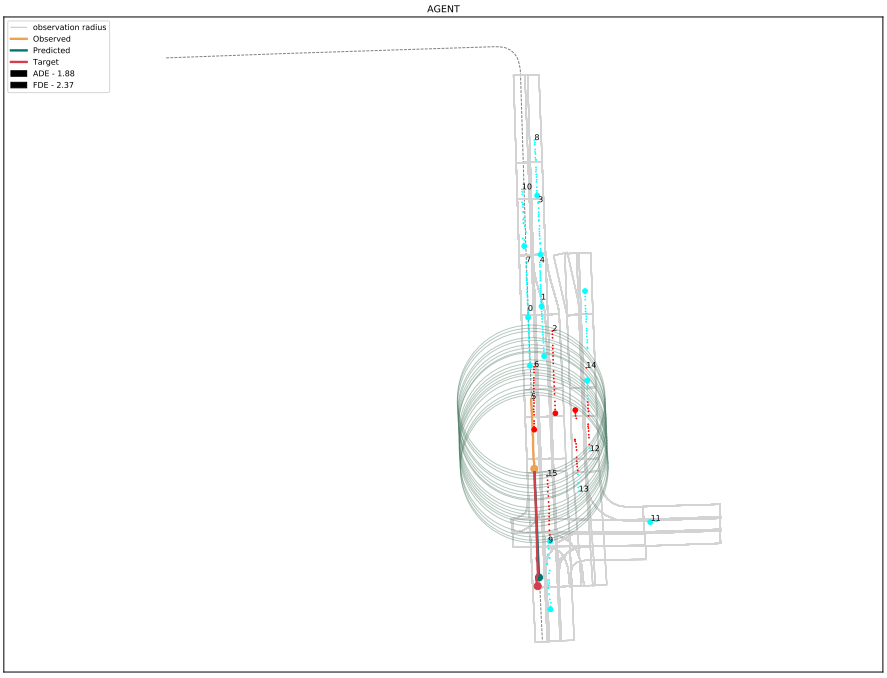


Abbildung B.5: Vorhersage für GAT-4.

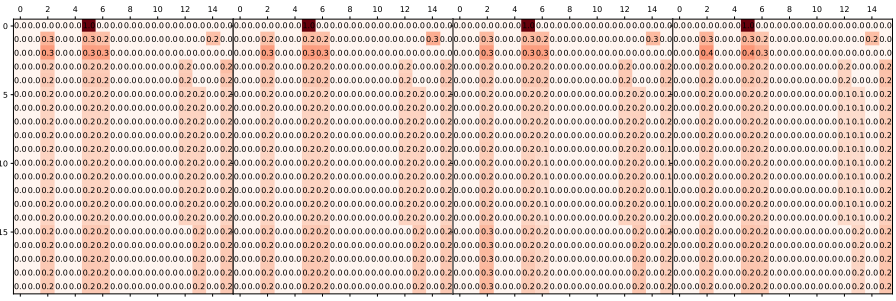
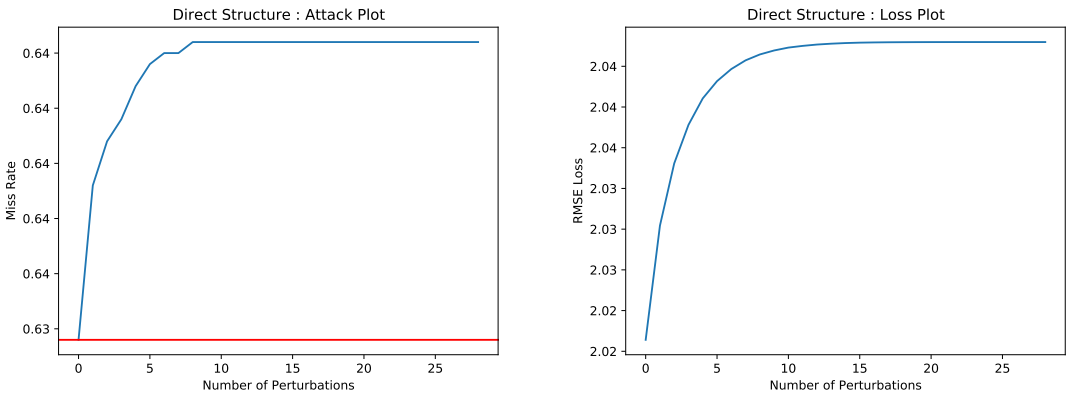


Abbildung B.6: Attentionmatrix für GAT-4. Jede Zeile entspricht einem Zeitschritt. Jede Spaltennummer dem entsprechenden Verkehrsteilnehmer.

Anhang C

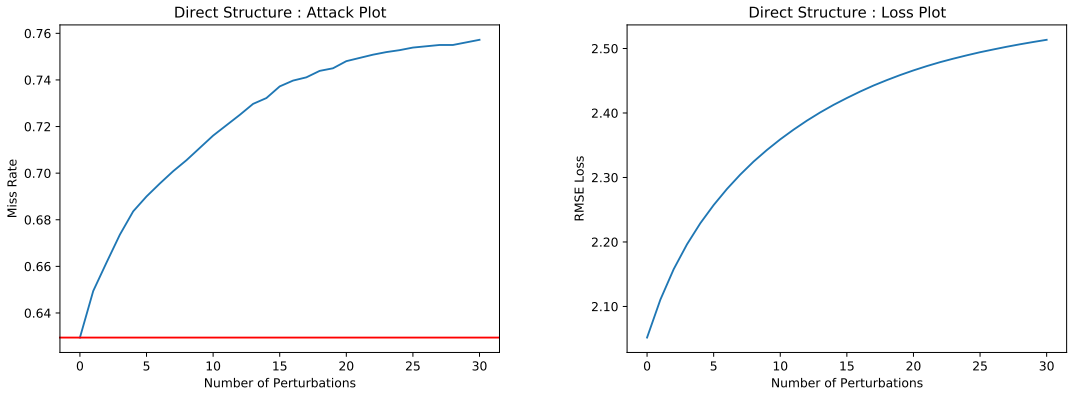
Weitere Grafiken

C.1 Direkte strukturelle Angriffe



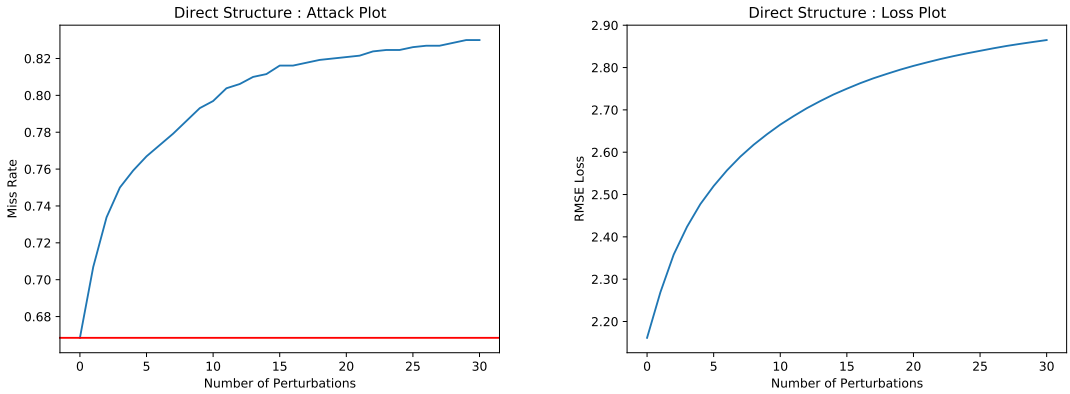
(a) **(b)**

Abbildung C.1: (a) Miss Rate für direkte Strukturangriffe auf das GCN Modell. **(b)** Verlauf des Fehlersterm für direkte Strukturangriffe für das GCN Modell



(a) **(b)**

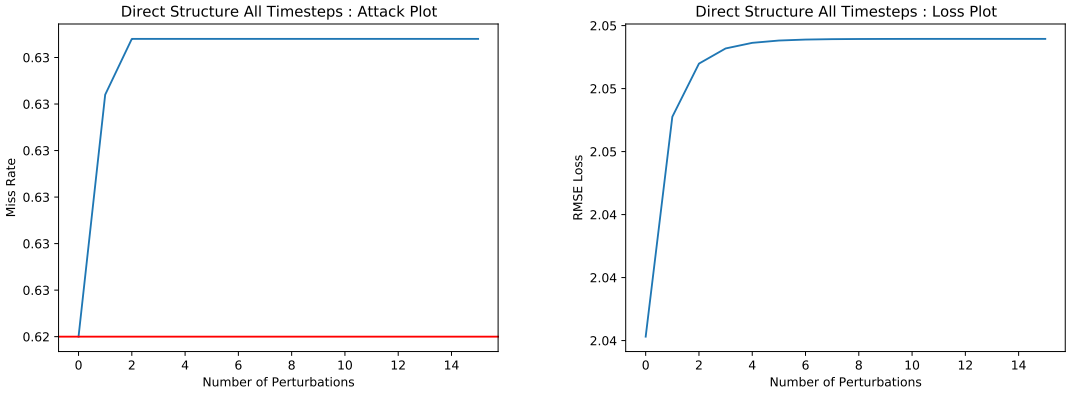
Abbildung C.2: **(a)** Miss Rate für direkte Strukturangriffe auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für direkte Strukturangriffe für das GAT-2 Modell



(a) **(b)**

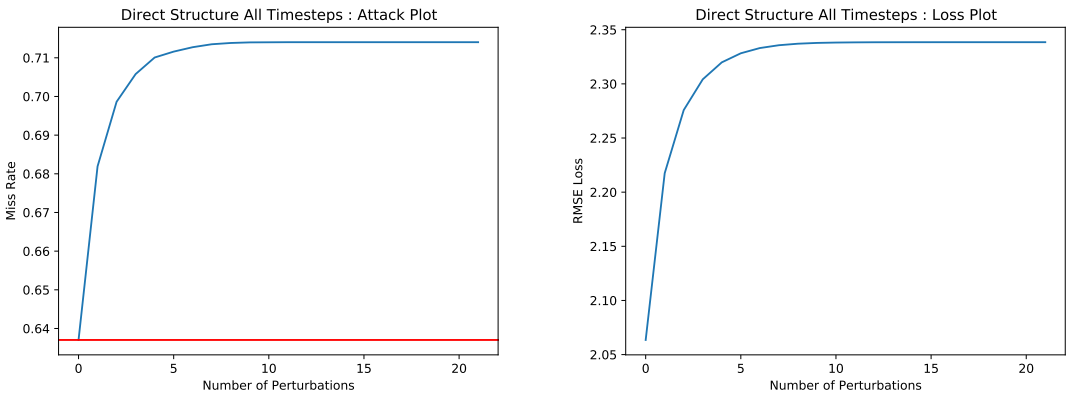
Abbildung C.3: **(a)** Miss Rate für direkte Strukturangriffe auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für direkte Strukturangriffe für das GAT-4 Modell

C.2 Direkte strukturelle Angriffe für alle Zeitschritte



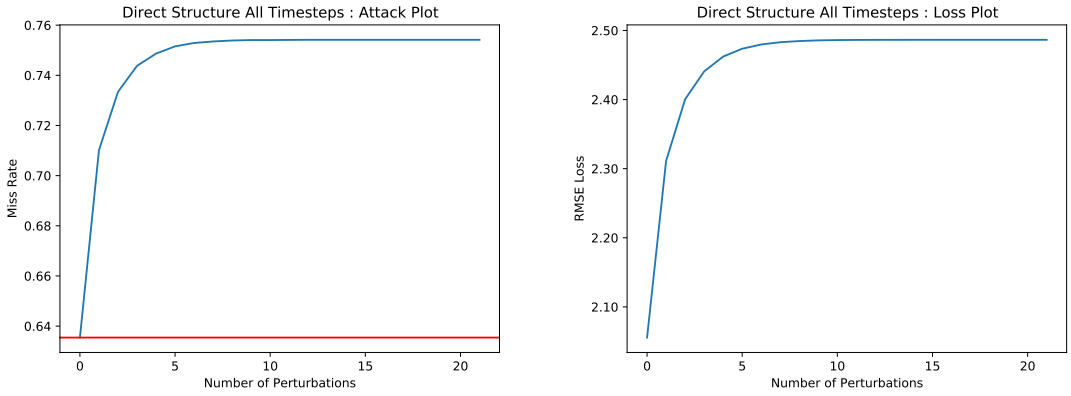
(a) **(b)**

Abbildung C.4: **(a)** Miss Rate für direkte Angriffe auf alle Zeitschritte gleichzeitig auf das GCN Modell. **(b)** Verlauf des Fehlersterm für direkte Angriffe auf alle Zeitschritte gleichzeitig für das GCN Modell



(a) **(b)**

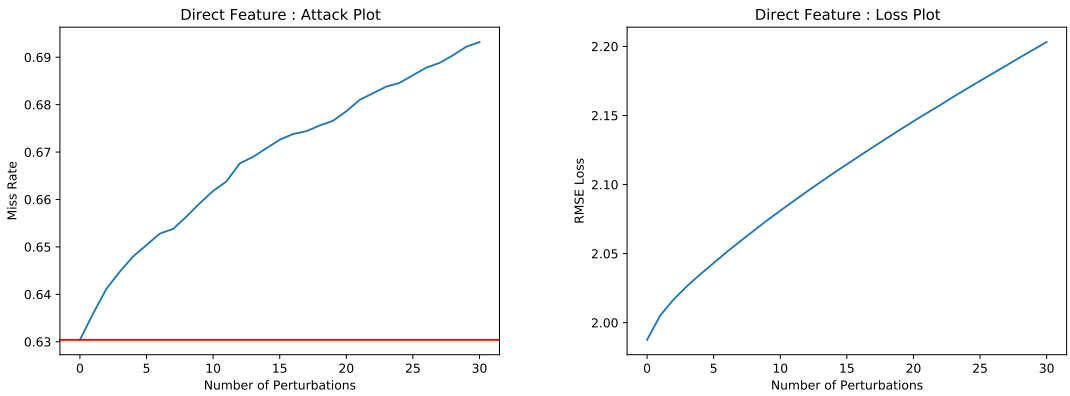
Abbildung C.5: **(a)** Miss Rate für direkte Angriffe auf alle Zeitschritte gleichzeitig auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für direkte Angriffe auf alle Zeitschritte gleichzeitig für das GAT-2 Modell



(a) **(b)**

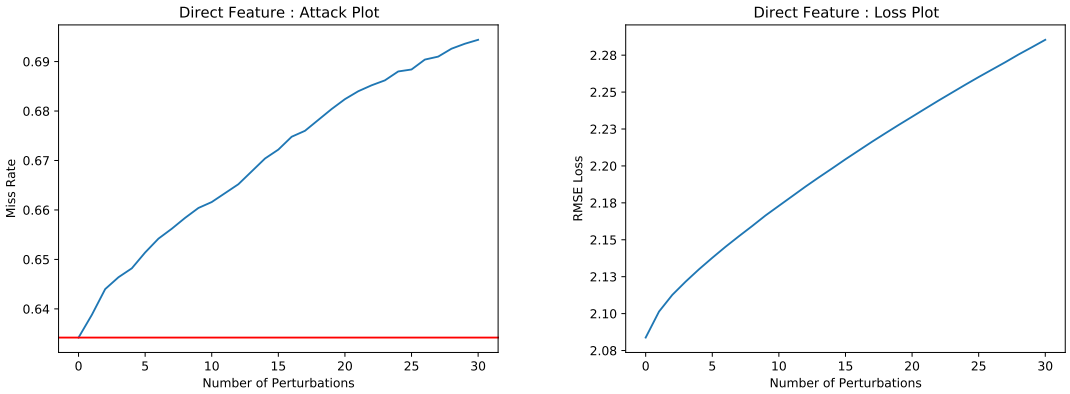
Abbildung C.6: (a) Miss Rate für direkte Angriffe auf alle Zeitschritte gleichzeitig auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für direkte Angriffe auf alle Zeitschritte gleichzeitig für das GAT-4 Modell

C.3 Direkte feature Angriffe



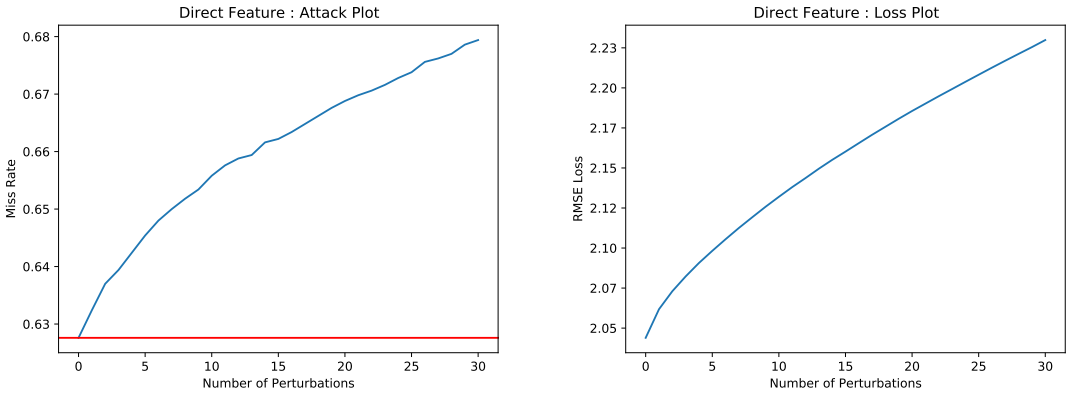
(a) **(b)**

Abbildung C.7: (a) Miss Rate für direkte Featureangriffe auf das GCN Modell. **(b)** Verlauf des Fehlersterm für direkte Featureangriffe auf das GCN Modell



(a) **(b)**

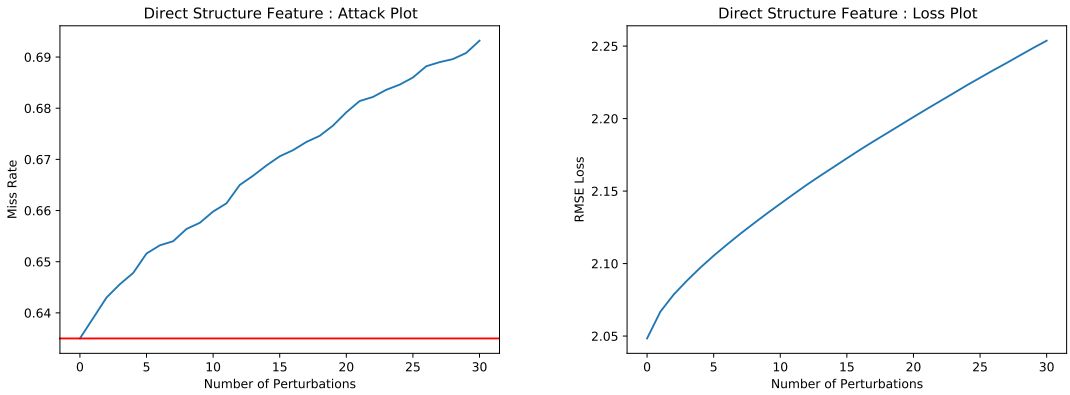
Abbildung C.8: (a) Miss Rate für direkte Featureangriffe auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für direkte Featureangriffe auf das GAT-2 Modell



(a) **(b)**

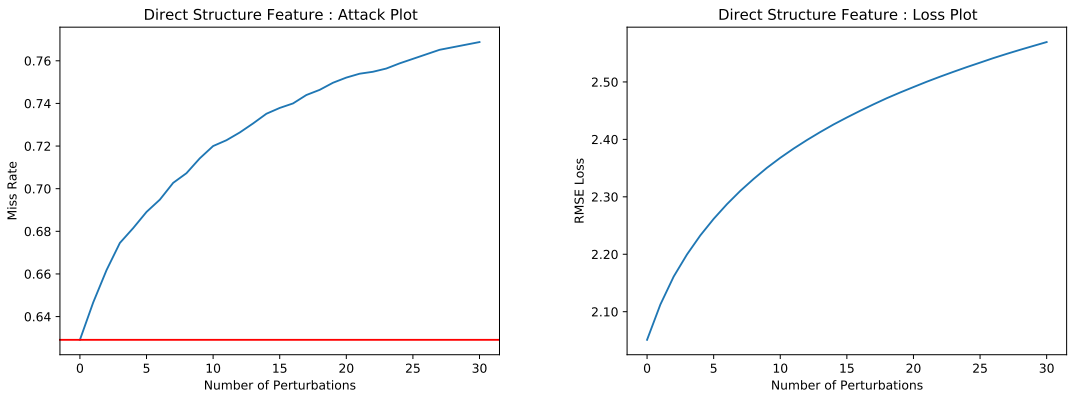
Abbildung C.9: (a) Miss Rate für direkte Featureangriffe auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für direkte Featureangriffe auf das GAT-4 Modell

C.4 Direkte strukturelle u. feature Angriffe



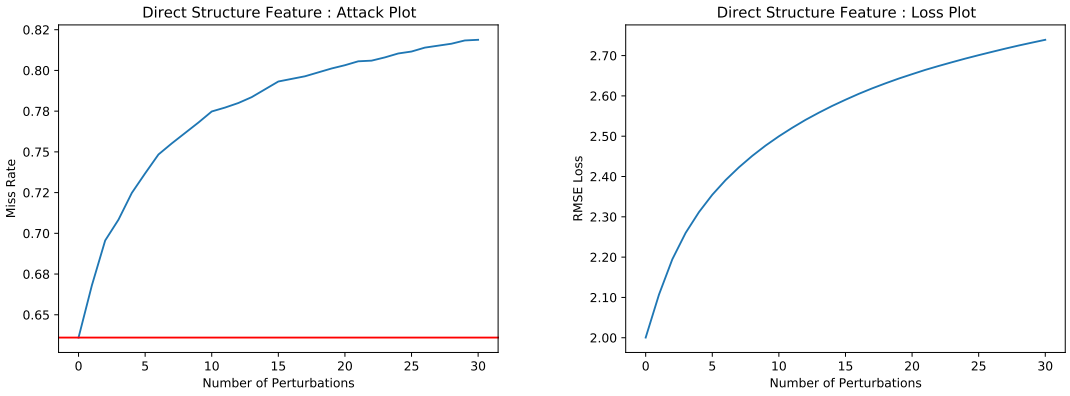
(a) **(b)**

Abbildung C.10: **(a)** Miss Rate für direkte Feature und Strukturangriffe auf das GCN Modell. **(b)** Verlauf des Fehlersterm für direkte Feature und Strukturangriffe auf das GCN Modell



(a) **(b)**

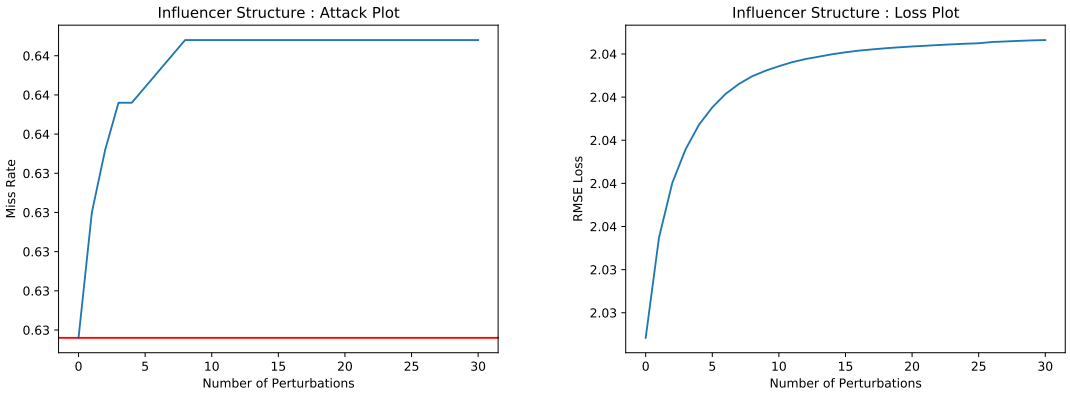
Abbildung C.11: **(a)** Miss Rate für direkte Feature und Strukturangriffe auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für direkte Feature und Strukturangriffe auf das GAT-2 Modell



(a) **(b)**

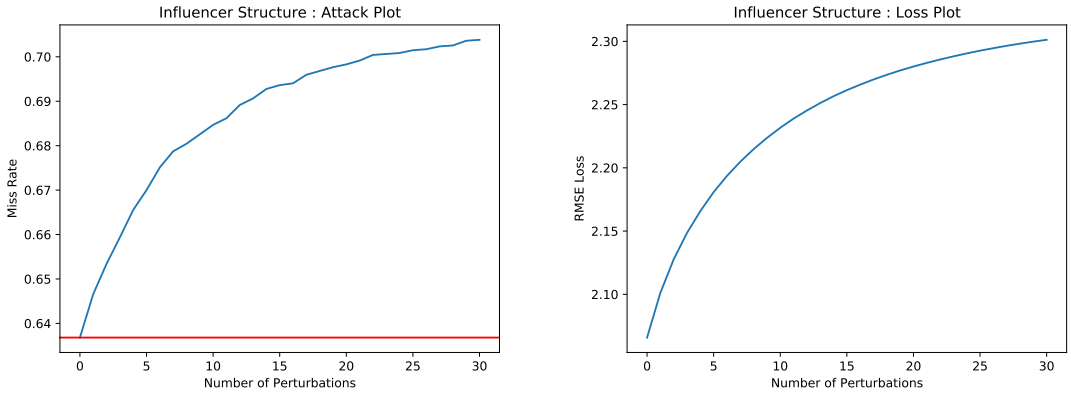
Abbildung C.12: (a) Miss Rate für direkte Feature und Strukturangriffe auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für direkte Feature und Strukturangriffe auf das GAT-4 Modell

C.5 Indirekte strukturelle Angriffe



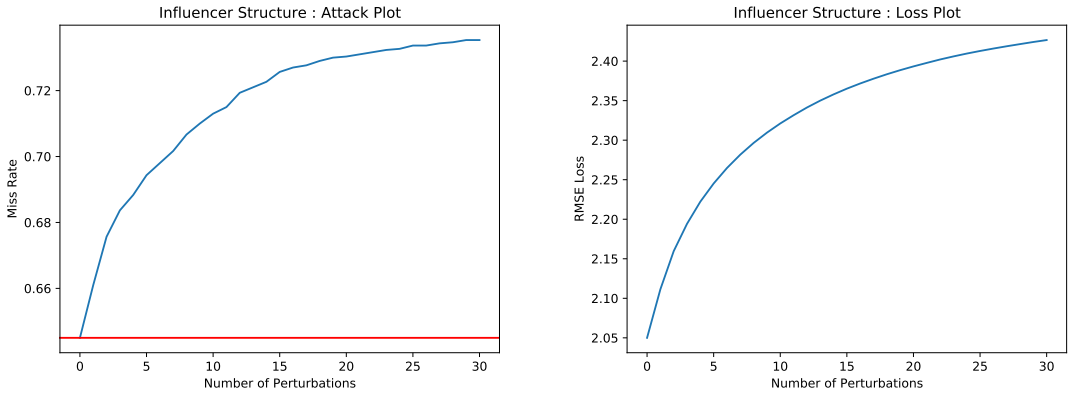
(a) **(b)**

Abbildung C.13: (a) Miss Rate für indirekte Strukturangriffe auf das GCN Modell. **(b)** Verlauf des Fehlersterm für indirekte Strukturangriffe auf das GCN Modell



(a) **(b)**

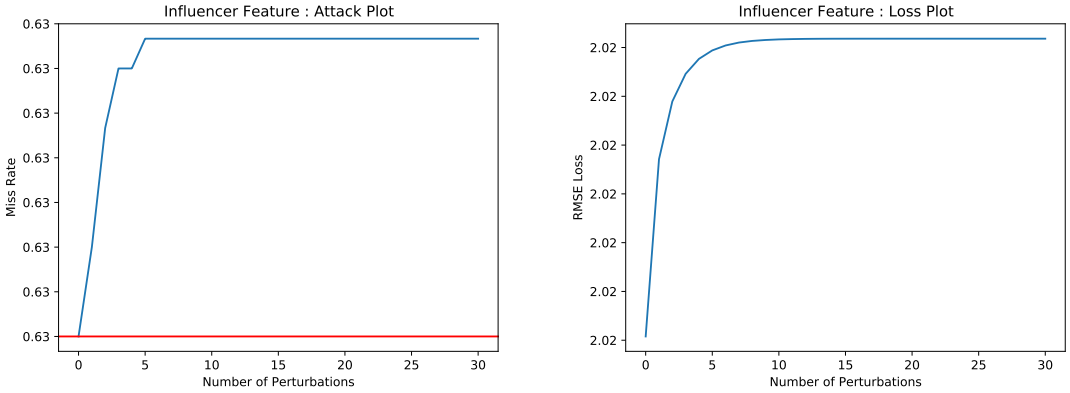
Abbildung C.14: (a) Miss Rate für indirekte Strukturangriffe auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für indirekte Strukturangriffe auf das GAT-2 Modell



(a) **(b)**

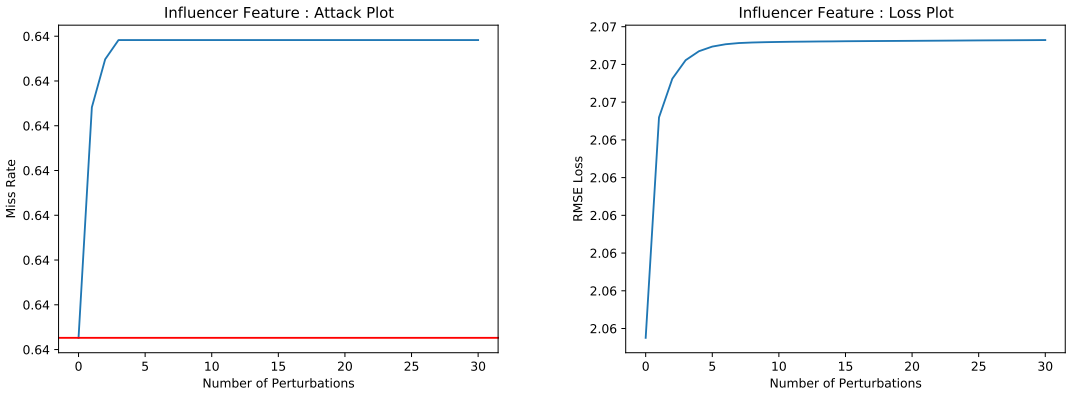
Abbildung C.15: (a) Miss Rate für indirekte Strukturangriffe auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für indirekte Strukturangriffe auf das GAT-4 Modell

C.6 Indirekte feature Angriffe



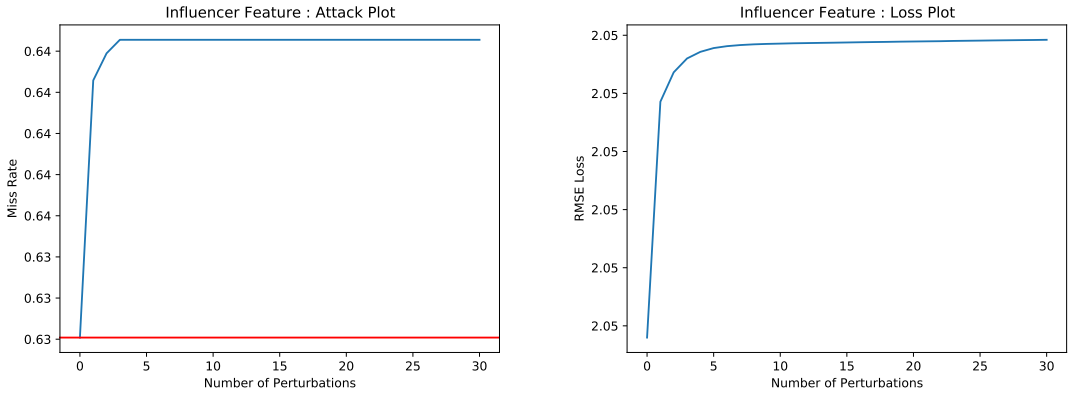
(a) **(b)**

Abbildung C.16: (a) Miss Rate für indirekte Featureangriffe auf das GCN Modell. **(b)** Verlauf des Fehlersterm für indirekte Featureangriffe auf das GCN Modell



(a) **(b)**

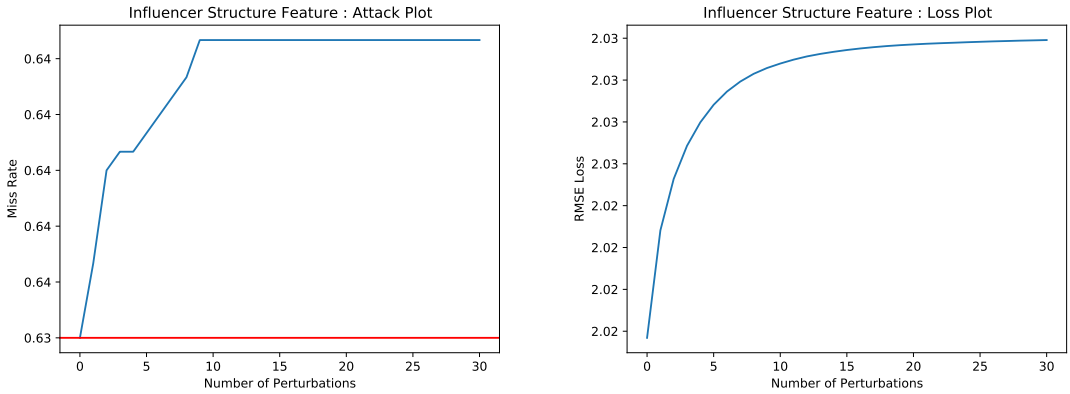
Abbildung C.17: (a) Miss Rate für indirekte Featureangriffe auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für indirekte Featureangriffe auf das GAT-2 Modell



(a) **(b)**

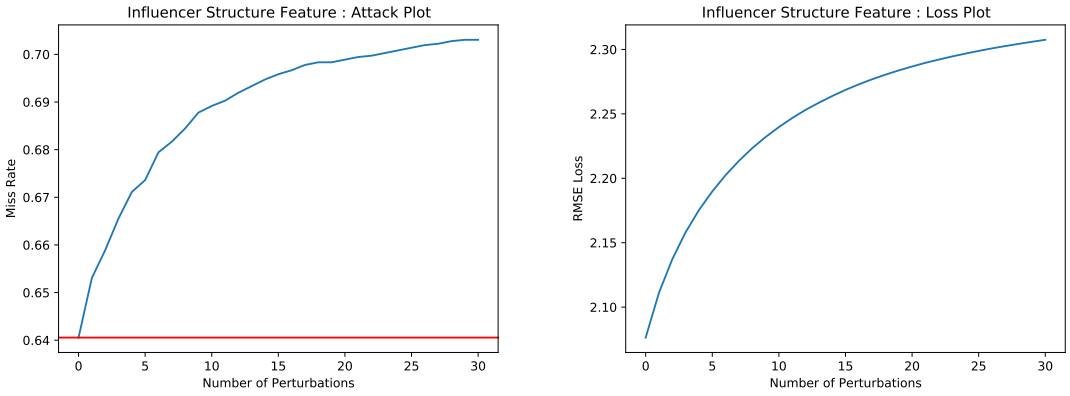
Abbildung C.18: (a) Miss Rate für indirekte Featureangriffe auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für indirekte Featureangriffe auf das GAT-4 Modell

C.7 Indirekte strukturrelle u. feature Angriffe



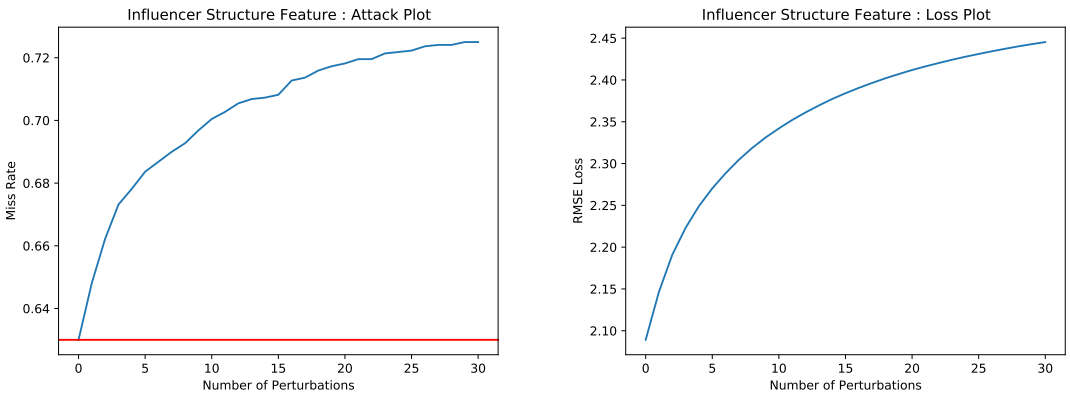
(a) **(b)**

Abbildung C.19: (a) Miss Rate für indirekte Feature und Strukturangriffe auf das GCN Modell. **(b)** Verlauf des Fehlersterm für indirekte Feature und Strukturangriffe auf das GCN Modell



(a) **(b)**

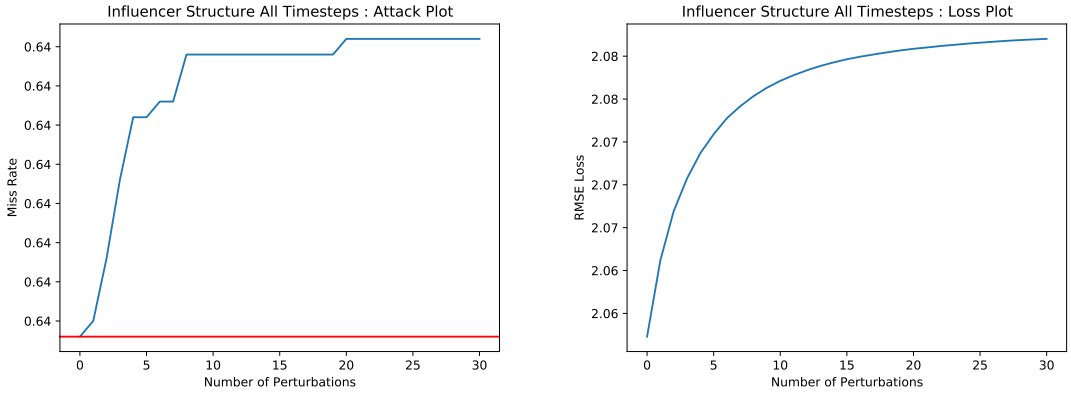
Abbildung C.20: (a) Miss Rate für indirekte Feature und Strukturangriffe auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für indirekte Feature und Strukturangriffe auf das GAT-2 Modell



(a) **(b)**

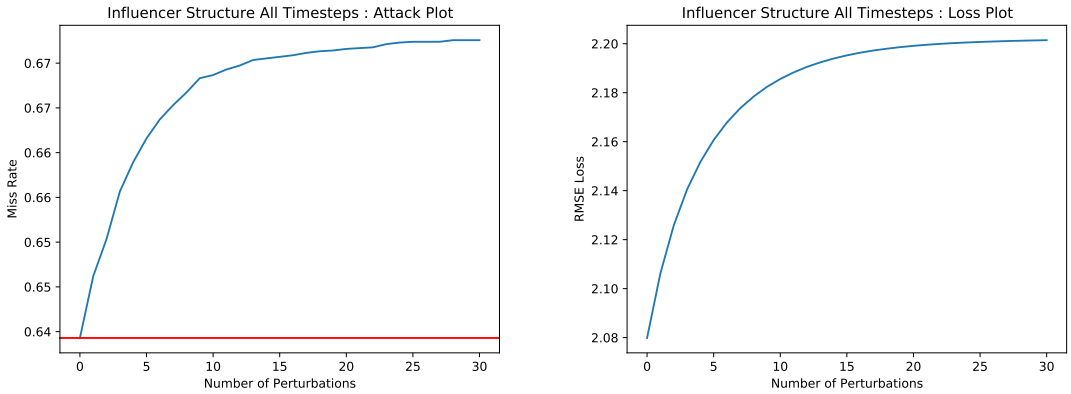
Abbildung C.21: (a) Miss Rate für indirekte Feature und Strukturangriffe auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für indirekte Feature und Strukturangriffe auf das GAT-4 Modell

C.8 Indirekte Angriffe auf alle Zeitschritte



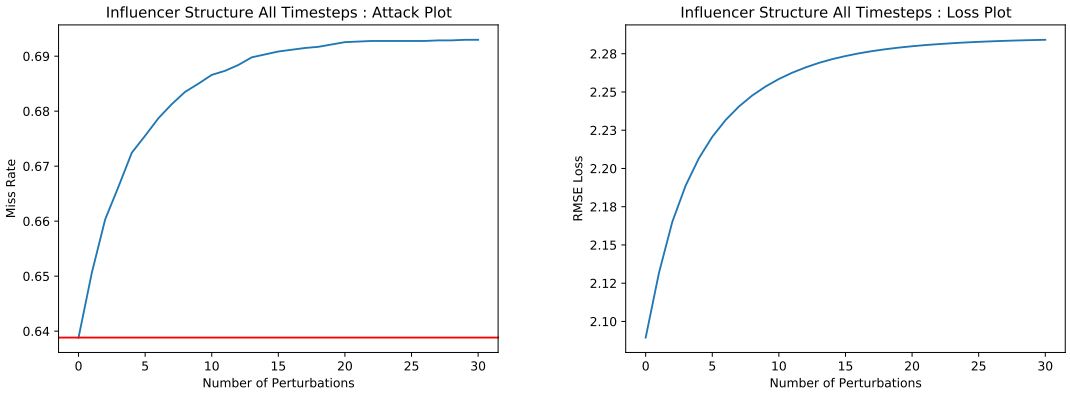
(a) **(b)**

Abbildung C.22: **(a)** Miss Rate für indirekte strukturelle Angriffe auf alle Zeitschritte auf das GCN Modell. **(b)** Verlauf des Fehlersterm für indirekte strukturelle Angriffe auf alle Zeitschritte auf das GCN Modell



(a) **(b)**

Abbildung C.23: **(a)** Miss Rate für indirekte strukturelle Angriffe auf alle Zeitschritte auf das GAT-2 Modell. **(b)** Verlauf des Fehlersterm für indirekte strukturelle Angriffe auf alle Zeitschritte auf das GAT-2 Modell



(a) **(b)**

Abbildung C.24: **(a)** Miss Rate für indirekte strukturelle Angriffe auf alle Zeitschritte auf das GAT-4 Modell. **(b)** Verlauf des Fehlersterm für indirekte strukturelle Angriffe auf alle Zeitschritte auf das GAT-4 Modell

Abbildungsverzeichnis

Liste der Algorithmen

Literaturverzeichnis

[1] Alahi, Alexandre, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei und Silvio Savarese: *Social lstm: Human trajectory prediction in crowded spaces.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, Seiten 961–971, 2016.

[2] Backpropagation: *RBackpropagation*, 2021. [Online; accessed 01-February-2021].

[3] Batch normalization: *Batch normalization*, 2021. [Online; accessed 01-February-2021].

[4] Bengio, Yoshua, Jérôme Louradour, Ronan Collobert und Jason Weston: *Curriculum learning.* In: *Proceedings of the 26th annual international conference on machine learning*, Seiten 41–48, 2009.

[5] Bruna, Joan, Wojciech Zaremba, Arthur Szlam und Yann LeCun: *Spectral networks and locally connected networks on graphs.* arXiv preprint arXiv:1312.6203, 2013.

[6] Carlini, Nicholas und David Wagner: *Towards evaluating the robustness of neural networks.* In: *2017 ieee symposium on security and privacy (sp)*, Seiten 39–57. IEEE, 2017.

[7] Chang, Ming-Fang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan et al.: *Argoverse: 3d tracking and forecasting with rich maps.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seiten 8748–8757, 2019.

[8] Convolution theorem: *Convolution theorem*, 2021. [Online; accessed 23-Januar-2021].

[9] Dai, Hanjun, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu und Le Song: *Adversarial attack on graph structured data.* In: *International conference on machine learning*, Seiten 1115–1124. PMLR, 2018.

[10] Defferrard, Michaël, Xavier Bresson und Pierre Vandergheynst: *Convolutional neural networks on graphs with fast localized spectral filtering.* Advances in neural information processing systems, 29:3844–3852, 2016.

[11] Deng, Yao, Xi Zheng, Tianyi Zhang, Chen Chen, Guannan Lou und Miryung Kim: *An Analysis of Adversarial Attacks and Defenses on Autonomous Driving Models.* arXiv preprint arXiv:2002.02175, 2020.

[12] Deo, Nachiket und Mohan M Trivedi: *Convolutional social pooling for vehicle trajectory prediction.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Seiten 1468–1476, 2018.

[13] Em, Azadeh, Majid Sarvi und Saeed Bagloee: *Using Kalman filter algorithm for short-term traffic flow prediction in a connected vehicle environment.* Journal of Modern Transportation, 07 2019.

[14] Gao, Jiyang, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li und Cordelia Schmid: *Vectornet: Encoding hd maps and agent dynamics from vectorized representation.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seiten 11525–11533, 2020.

[15] Goodfellow, Ian J, Jonathon Shlens und Christian Szegedy: *Explaining and harnessing adversarial examples.* arXiv preprint arXiv:1412.6572, 2014.

[16] Gupta, Agrim, Justin Johnson, Li Fei-Fei, Silvio Savarese und Alexandre Alahi: *Social gan: Socially acceptable trajectories with generative adversarial networks.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Seiten 2255–2264, 2018.

[17] He, Kaiming, Xiangyu Zhang, Shaoqing Ren und Jian Sun: *Deep residual learning for image recognition.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, Seiten 770–778, 2016.

[18] Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever und Ruslan R Salakhutdinov: *Improving neural networks by preventing co-adaptation of feature detectors.* arXiv preprint arXiv:1207.0580, 2012.

[19] Hu, Yeping, Wei Zhan, Liting Sun und Masayoshi Tomizuka: *Multi-modal probabilistic prediction of interactive behavior via an interpretable model.* In: *2019 IEEE Intelligent Vehicles Symposium (IV)*, Seiten 557–563. IEEE, 2019.

[20] Huang, Xiaowei, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu und Xinping Yi: *A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability.* Computer Science Review, 37:100270, 2020.

[21] Huang, Xinyu, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng und Ruigang Yang: *The apolloscape open dataset for autonomous driving and its application.* IEEE transactions on pattern analysis and machine intelligence, 42(10):2702–2719, 2019.

[22] Ioffe, Sergey und Christian Szegedy: *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In: *International conference on machine learning*, Seiten 448–456. PMLR, 2015.

[23] Jin, Wei, Yaxin Li, Han Xu, Yiqi Wang und Jiliang Tang: *Adversarial attacks and defenses on graphs: A review and empirical study.* arXiv preprint arXiv:2003.00653, 2020.

[24] Katz, Guy, Clark Barrett, David L Dill, Kyle Julian und Mykel J Kochenderfer: *Towards proving the adversarial robustness of deep neural networks.* arXiv preprint arXiv:1709.02802, 2017.

[25] Kipf, Thomas N und Max Welling: *Semi-supervised classification with graph convolutional networks.* arXiv preprint arXiv:1609.02907, 2016.

[26] LeCun, Yann, Yoshua Bengio et al.: *Convolutional networks for images, speech, and time series.* The handbook of brain theory and neural networks, 3361(10):1995, 1995.

[27] Lee, Namhoon, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr und Manmohan Chandraker: *Desire: Distant future prediction in dynamic scenes with interacting agents.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Seiten 336–345, 2017.

[28] Lefevre, Stephanie, Dizan Vasquez und Christian Laugier: *A survey on motion prediction and risk assessment for intelligent vehicles.* Robomech Journal, 1, 07 2014.

[29] Li, Xin, Xiaowen Ying und Mooi Choo Chuah: *Grip: Graph-based interaction-aware trajectory prediction.* In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Seiten 3960–3966. IEEE, 2019.

[30] Mahendran Venkatachalam: *Recurrent Neural Networks*, 2019. [Online; accessed 03-February-2021].

[31] Mercat, Jean Pierre, Thomas Gilles, Nicole El Zoghby, Guillaume Sandou, Dominique Beauvois und Guillermo Pita Gil: *Multi-Head Attention for Multi-Modal Joint Vehicle Motion Forecasting.* arXiv: Learning, 2019.

[32] Mozaffari, Sajjad, Omar Y Al-Jarrah, Mehrdad Dianati, Paul Jennings und Alexandros Mouzakitis: *Deep Learning-based Vehicle Behaviour Prediction For Autonomous Driving Applications: A Review.* arXiv preprint arXiv:1912.11676, 2019.

[33] Nearest-neighbor interpolation: *Nearest-neighbor interpolation*, 2021. [Online; accessed 01-February-2021].

[34] NGSIM U-101, I-80: *NGSIM U-101, I-80*, 2017. [Online; accessed 25-Januar-2021].

[35] Pascanu, Razvan, Tomas Mikolov und Yoshua Bengio: *On the difficulty of training recurrent neural networks.* In: *International conference on machine learning*, Seiten 1310–1318. PMLR, 2013.

[36] Rectified Linear Units (ReLU) in Deep Learning: *Rectified Linear Units (ReLU) in Deep Learning*, 2018. [Online; accessed 01-February-2021].

[37] Rhinehart, Nicholas, Rowan McAllister, Kris Kitani und Sergey Levine: *Precog: Prediction conditioned on goals in visual multi-agent settings.* In: *Proceedings of the IEEE International Conference on Computer Vision*, Seiten 2821–2830, 2019.

[38] Simeon Kostadinov: *Understanding GRU Networks*, 2017. [Online; accessed 03-February-2021].

[39] Sobel-Operator: *Sobel-Operator*, 2021. [Online; accessed 23-Januar-2021].

[40] Sun, Lichao, Yingtong Dou, Carl Yang, Ji Wang, Philip S Yu und Bo Li: *Adversarial attack and defense on graph data: A survey.* arXiv preprint arXiv:1812.10528, 2018.

[41] Sutskever, Ilya, Oriol Vinyals und Quoc V Le: *Sequence to sequence learning with neural networks.* arXiv preprint arXiv:1409.3215, 2014.

[42] Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow und Rob Fergus: *Intriguing properties of neural networks.* arXiv preprint arXiv:1312.6199, 2013.

[43] Tramèr, Florian, Nicolas Papernot, Ian Goodfellow, Dan Boneh und Patrick McDaniel: *The space of transferable adversarial examples.* arXiv preprint arXiv:1704.03453, 2017.

[44] Transport, Department for: *Research on the Impacts of Connected and Autonomous Vehicles (CAVs) on Traffic Flow*, 2016.

[45] Unfallstatistiken – Verkehrsunfälle in Deutschland: *Unfallstatistiken – Verkehrsunfälle in Deutschland*, 2018. [Online; accessed 27-Januar-2021].

[46] Universal approximation theorem: *Universal approximation theorem*, 2021. [Online; accessed 23-Januar-2021].

[47] Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio und Yoshua Bengio: *Graph attention networks.* arXiv preprint arXiv:1710.10903, 2017.

[48] Wu, Huijun, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu und Liming Zhu: *Adversarial examples on graph data: Deep insights into attack and defense.* arXiv preprint arXiv:1903.01610, 2019.

[49] Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang und S Yu Philip: *A comprehensive survey on graph neural networks.* IEEE Transactions on Neural Networks and Learning Systems, 2020.

[50] Yan, Sijie, Yuanjun Xiong und Dahua Lin: *Spatial temporal graph convolutional networks for skeleton-based action recognition.* In: *Proceedings of the AAAI conference on artificial intelligence*, Band 32, 2018.

[51] Zachary's karate club: *Zachary's karate club*, 2021. [Online; accessed 22-Januar-2021].

[52] Zhou, Jie, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li und Maosong Sun: *Graph neural networks: A review of methods and applications.* arXiv preprint arXiv:1812.08434, 2018.

[53] Zügner, Daniel, Amir Akbarnejad und Stephan Günnemann: *Adversarial attacks on neural networks for graph data.* In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Seiten 2847–2856, 2018.