technische universität
dortmund

Bachelorarbeit

Kontextsensitives Geometrisches Deep Learning

Ronja van den Berg
September 2019

Gutachter:
Prof. Dr. Katharina Morik
Dr. Thomas Liebig

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für künstliche Intelligenz (LS8)
http://www-ai.cs.uni-dortmund.de

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Die Aufgabe einer präzisen Prognose von räumlich verteilten Zeitreihen ist ein komplexes, jedoch auch sehr interessantes Anwendungsgebiet. Es gibt viele Bereiche, in denen solche Aufgaben relevant sind. Die Vorhersage von Verkehrs- und Wetterdaten sind nur einige wenige Beispiele dafür. Diese Art von Daten fällt überall dort an, wo Zeitreihen mit einem räumlichen Zusammenhang auftreten. Ein räumlicher Zusammenhang kann beispielsweise ein verteiltes Sensornetzwerk sein, wo an jedem Sensor regelmäßig Daten erfasst werden. Es gibt zwei große herausfordernde Aspekte in diesem Bereich:

- Unvorhergesehene Ereignisse und spontane Änderungen in der Zeitreihe
- Räumliche Zusammenhänge innerhalb der räumlichen Strukturen der geometrischen Eingabe

Die unvorhergesehen Ereignisse, wie beispielsweise ein Verkehrsunfall oder der Stillstand einer Maschine, sind für viele Modelle schwierig zu berücksichtigen, da die Prognose entsprechend angepasst werden muss.
Zusätzlich treten räumliche Zusammenhänge auf, wie beispielsweise das Umfahren einer Unfallstelle, welche von den Modellen zusätzlich berücksichtigt werden müssen. Da solche Ereignisse in unregelmäßigen Abständen auftreten, sind sie schwer zu erfassen.
Die räumlichen Zusammenhänge innerhalb der Eingabe stellen zusätzliche Informationen bereit, welche Modelle nutzen können, um die zugrundeliegende Wahrscheinlichkeitsverteilung besser zu lernen und Zusammenhänge innerhalb der Eingabedaten besser zu berücksichtigen. Dafür muss das Modell jedoch in der Lage sein, diese zusätzlichen räumlichen Informationen, wie beispielsweise die Abstände zwischen den Verkehrssensoren, zu erfassen und zu verarbeiten.
Das Ziel dieser Arbeit ist die Entwicklung und Implementierung eines kontextsensitiven Deep Learning Modells, welches Kurzzeitprognosen y_p für räumlich verteilte Zeitreihen erzeugt und geometrisch strukturierte Eingaben verarbeitet. Es soll vor allem in der Lage sein, die zusätzlichen räumlichen Informationen und die Topologie der geometrischen Eingabe zu berücksichtigen. Als Eingabe erhält unser Modell die vergangenen c Beobachtungen $(x_{t-c}, ..., x_t)$. Der Zeitpunkt t beschreibt den Ausgangszeitpunkt für den Vorhersagehorizont p. Die Menge der vorher betrachteten Zeitschritte stellt den Kontext unseres Modells dar [Wöl13]. Das Lernproblem ist grafisch in Abbildung 1.1 dargestellt.
Für die Kurzzeitprognose betrachten wir einen Vorhersagehorizont von $p = \{15, 30, 60\}$ [Jia+18]. Ein Vorhersagehorizont von 60 Minuten entspricht nicht direkt einer Kurzzeitprognose, jedoch können wir durch diese Experimente erkennen, ob unser Modell für längerfristige Prognosen geeignet ist.
Um die Eignung unseres entwickelten Modells zu testen, werden wir Experimente auf zwei verschiedenen Verkehrsdatensätzen durchführen. Bei den beiden Datensätzen han-

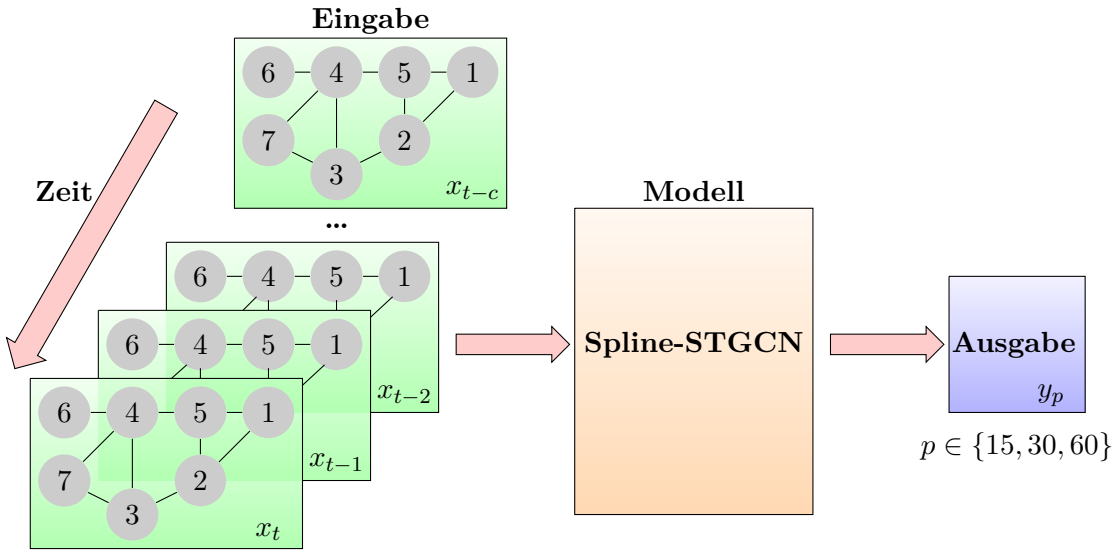


Abbildung 1.1: Die Abbildung zeigt eine grafische Darstellung des Lernproblems. Das Modell erhält eine Menge vergangener Beobachtungen $\{x_{t-c}, ..., x_t\}$ als Eingabe. Die Anzahl der Beobachtungen ist abhängig vom Kontext c. Das Modell erzeugt eine Prognose y_p für einen festgelegten Vorhersagehorizont p.

delt es sich um den METR-LA-Datensatz[1] und den PeMSD7(M)-Datensatz[2]. Die Daten des METR-LA Datensatzes besitzen eine komplexere Struktur. Deshalb werden wir diesen Datensatz verwenden, um geeignete Parameter für unser Modell zu finden. Anschließend werden wir die Vorhersagegenauigkeit unseres Modells testen, indem wir häufig verwendete Fehlermaße für Zeitreihen mit Ergebnissen von anderen geeigneten Modellen vergleichen. Die Nichtlinearität der Daten und unvorhergesehene Ereignisse sind für viele Modelle eine große Herausforderung.
Zu Beginn der Prognose von räumlich verteilten Zeitreihen wurden häufig lineare Modelle, wie beispielsweise der Auto-Regressive Integrated Moving Average (ARIMA) [AC79] eingesetzt. Diese sind jedoch nicht in der Lage den Verkehrsfluss präzise vorherzusagen. Heutzutage gibt es diverse Erweiterungen dieser Modelle, wie beispielsweise ARIMA mit Kalman-Filter oder ARIMA mit künstlichen Neuronalen Netzen [LTL12]. Diese Modelle sind zwar in der Lage die zeitliche Abhängigkeit innerhalb der Eingabedaten zu erfassen, jedoch können sie die räumliche Komponente der Eingabedaten nicht berücksichtigen. Viele eingesetzte Modelle für die Vorhersage von Verkehrsdaten basieren auf Rekurrenten Neuronalen Netzwerken (RNNs) wie beispielsweise Long Short Term Memory RNNs (LSTM-RNNs) [Cho+14]. Auch dieses Modell kann die räumlichen Abhängigkeiten räumlich verteilter Zeitreihen nicht berücksichtigen. Zusätzlich ist die Architektur von Modellen, welche auf RNNs basieren, häufig komplex und der Trainingsprozess sehr zeitaufwendig. Um diese Probleme zu umgehen, können Modelle mit weniger Schichten oder einer geringeren Anzahl an Trainingsepochen verwendet werden. Diese Ansätze führen, genauso wie die Reduktion der Trainingsdatenmenge, zwar zu einer geringeren Komplexität und

[1] `https://github.com/FelixOpolka/STGCN-PyTorch`
[2] `https://github.com/VeritasYin/STGCN_IJCAI-18`

einem geringeren Zeitaufwand, jedoch nicht zu einer höheren Genauigkeit der von diesen Modellen erzeugten Vorhersagen.
Im Jahr 2013 wurden Spatio-Temporal Random Fields (STRFs) [PLM13] entwickelt. Diese sind sehr gut für die Kurzzeitprognose von Verkehrsdaten geeignet. Sie verwenden für die Modellierung Markovprozesse. Die Vorhersage des Modells hängt ausschließlich von der letzten Beobachtung ab. Dadurch kann das Modell sehr gut auf kurzzeitige Änderungen im Verkehrsfluss reagieren. Unser entwickeltes Modell bekommt neben der letzten Beobachtung einen Kontext mit mehreren vergangenen Beobachtungen übergeben. Das bietet dem Modell die Möglichkeit mittel- und längerfristige Abhängigkeiten besser zu erfassen.
Die Arbeit [Ma+17] beschreibt ein Modell, welches auf einem Convolutional Neural Network (CNN) basiert. In dieser Methode werden der Verkehrsfluss und die räumlichen Zusammenhänge als Bild dargestellt und verarbeitet. Das Problem dieses Ansatzes ist der indirekte räumliche Zusammenhang zwischen den einzelnen Sensoren. Dieser kann durch das Modell nicht erfasst werden, da durch die Faltung auf dem Bild ausschließlich die direkte Nachbarschaft der Sensoren betrachtet wird und somit der erweiterte räumliche Zusammenhang, wie beispielsweise die Distanzen zwischen den Sensoren, nicht vollständig berücksichtigt werden kann.
Bisher gibt es wenige andere geometrische Deep Learning Modelle, welche sowohl in der Lage sind die räumlichen als auch die zeitlichen Merkmale der Zeitreihe ausreichend zu erfassen, um die beiden großen Herausforderungen bei der Vorhersage berücksichtigen [YYZ19] [Li+17]. Um die Probleme, wie Komplexität, Trainingsaufwand und die Berücksichtigung räumlicher Zusammenhänge zu lösen, entwickeln wir im Zuge dieser Arbeit ein geometrisches Deep Learning Modell, welches aus zwei Hauptkomponenten besteht. Wir verwenden ein Framework, die Spatio-Temporal Graph Convolutional Neural Networks (STGCNs) [YYZ17], welches durch seinen Aufbau besonders gut für die Extraktion räumlicher und zeitlicher Merkmale der Eingabedaten geeignet ist. Zusätzlich verwenden wir ein Convolutional Neural Network, das Spline-based Convolutional Neural Network (Spline-CNN) [Fey+17], welches geometrische Eingabedaten gut verarbeiten kann und aufgrund seines neuartigen, auf B-Splines basierenden Filters in der Lage ist, nicht offensichtliche räumliche Informationen aus den Eingabedaten zu extrahieren.

1.2 Aufbau der Arbeit

Zuerst werden im Kapitel 2 alle benötigten Grundlagen und Verfahren eingeführt und erklärt. Im Abschnitt 2.1 gehen wir auf die Grundlagen des Deep Learnings ein und stellen häufig verwendete Fehlermaße und die Adam-Optimierung (siehe 2.1.4) vor. Wir beschreiben den Aufbau und die Funktionsweise von Convolutional Neural Networks (CNNs) (siehe 2.2) und gehen darauf ein, was Geometrisches Deep Learning ist (siehe 2.3) und inwiefern sich Graph Convolutional Neural Networks (GCNNs) (siehe 2.4) von CNNs unterscheiden. Zusätzlich erklären wir im Abschnitt 2.5 B-Spline Basisfunktionen. Die Abschnitte 2.6 und 2.7 gehen genauer auf B-Spline Kurven und B-Spline Oberflächen ein, welche wir für das Verständnis der Filterkonstruktion der Spline-CNNs benötigen.
Im Kapitel 3 wird das Modell erläutert, welches im Rahmen dieser Arbeit entwickelt wird. Zunächst gehen wir auf die Spline-CNNs (siehe 3.1) ein, welche für die Extraktion der zeitlichen Merkmale der Eingabedaten verwendet werden. Dabei gehen wir besonders auf den B-Spline Faltungsoperator ein und erklären, inwiefern es sich dabei um ein GCNN handelt. Anschließend erklären wir im Abschnitt 3.2 den Aufbau der verwendeten STGCNs.

Zum Schluss dieses Kapitels erklären wir den Aufbau und die Funktionsweise unseres entwickelten Modells, der Spline-STGCNs (siehe 3). Anschließend gehen wir darauf ein, welche Eigenschaften wir uns von dem Modell erhoffen. Als Nächstes folgt mit Kapitel 4 die Experimentdurchführung auf zwei unterschiedlichen Datensätzen: METR-LA und PEMSD7(M). Auf dem METR-LA-Datensatz werden mehr Experimente mit dem entwickelten Modell durchgeführt, um verschiedene Parameter zu testen und geeignete Parameter für unser Modell zu finden (siehe Abschnitt 4.2.2). Zusätzlich werden im Abschnitt 4.1 kurz die verwendeten Vergleichsmodelle erklärt. Es wird jeweils auf die entsprechende Vorverarbeitung der beiden Datensätze eingegangen und kurz die verwendeten Fehlermaße erläutert, welche für den Vergleich mit anderen, für die Aufgabe geeigneten Modellen, verwendet werden. Der Vergleich mit anderen Modellen dient der Einordnung der Performanz der Spline-STGCNs. Anschließend werden im Abschnitt 4.4 die Ergebnisse der durchgeführten Experiment zusammengefasst und eingeordnet. Im Kapitel 5 gehen wir darauf ein, was in dieser Arbeit untersucht und was durch diese Arbeit gezeigt wurde. Zusätzlich werden offene Fragen aufgezeigt, inwiefern eine Erweiterung des entwickelten Modells sinnvoll ist und welche Fragestellungen nach dieser Arbeit noch offen bleiben.

2 Grundlagen und Theorie

2.1 Deep Learning

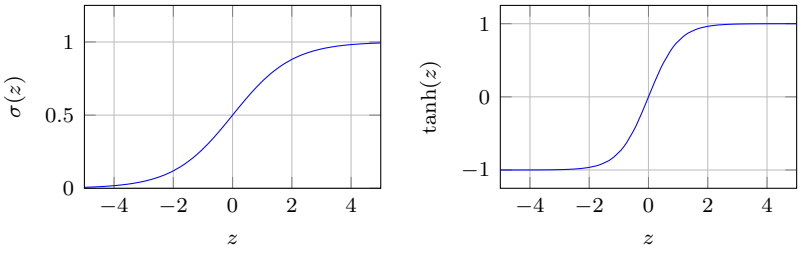
Deep Learning Methoden sind ein Bereich des maschinellen Lernens basierend auf künstlichen neuronalen Netzwerken. Machine-Learning-Algorithmen sollen aus Eingabedaten lernen. Eine häufig verwendete Definition des Begriffs „lernen“ geht auf Tom Mitchell zurück [Mit97]. Die Definition besagt, dass ein Computerprogramm aus der Erfahrung E bezüglich einer Klasse von Aufgaben T und einer Leistungsbewertung P lernt, wenn die Leistung in den Aufgaben von T, welche durch P bewertet werden, durch die Erfahrung E verbessert wird.
Es gibt viele unterschiedliche Arten von Aufgaben, welche mittels Deep Learning Verfahren gelöst werden können. Einige Aufgaben sind beispielsweise Klassifizierung, Regression, Anomalieerkennung, Entrauschen und Imputation. Als Leistungsbewertung dienen häufig Maße, welche angeben, inwiefern die vom Modell erzeugte Ausgabe von der Zielausgabe abweicht. Ein einfaches Beispiel eines Deep Learning Modells ist das Feedforward-Netzwerk. Dieses Modell, auch mehrschichtiges Perzeptron genannt, besteht aus mehreren Schichten künstlicher Neuronen.

2.1.1 Künstliche Neuronen

Ein künstliches Neuron kann als eine einzelne Recheneinheit, auch Unit genannt, in einer Schicht eines neuronalen Netzwerks betrachtet werden. Als Eingabe erhält ein Neuron einen Vektor

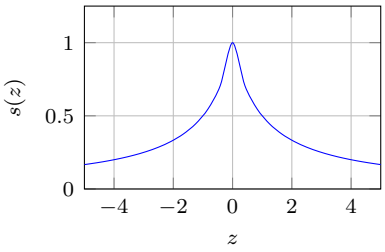
$$\mathbf{x} = (x_1, x_2, ..., x_n) \tag{2.1}$$

mit $n \in \mathbb{N}$ Eingaben.
Auf Basis dieser Eingabe erzeugt das Neuron eine Ausgabe y. Um diese Ausgabe zu berechnen, verwendet das Neuron eine, für das Neuron spezifische, Gewichtsmatrix $\mathbf{W}$. Die Gewichtsmatrix ordnet jedem Wert x_i der Eingabe mit $0 \leq i \leq n$ ein eigenes Gewicht $w_i \in \mathbf{W}$ zu und bestimmt damit, wie wichtig der Teil der Eingabe für die Erzeugung der Ausgabe des Neurons ist. Zusätzlich kann das Neuron einen Bias b als Eingabe erhalten. Durch die Anpassung der einzelnen Gewichte wird die Ausgabe des Neurons verändert.
Um zu bestimmen, ob der berechnete Wert eines Neurons innerhalb des Netzwerks weitergegeben wird, wird eine Aktivierungsfunktion g verwendet. Sie wird nach dem Aufsummieren der einzelnen Produkte von Eingabewert und Gewichtsmatrix angewendet und ist vergleichbar mit einem Schwellwert.
Es gibt unterschiedliche Aktivierungsfunktionen, welche in neuronalen Netzen angewendet werden können. Am häufigsten verwendet werden die Sigmoid-Funktion, der Tangens hyperbolicus, die Softsign-Funktion und Rectified-Linear-Unit (ReLU) [Nwa+18].
In der Praxis kann für jede Schicht eines neuronales Netzwerks eine andere Aktivierungsfunktion verwendet werden.
In dieser Arbeit werden die Sigmoid-Funktion und ReLU eingesetzt.
Die vier genannten Beispiele für gängige Aktivierungsfunktionen sind in Abbildung 2.1

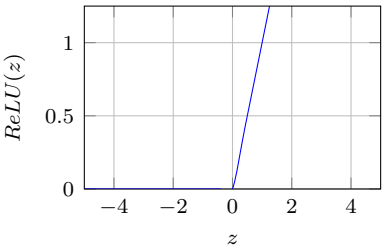


(a) Sigmoid-Funktion

(b) Tangens hyperbolicus



(c) Softsign-Funktion



(d) Rectified-Linear-Unit

Abbildung 2.1: Die Abbildung zeigt vier häufig verwendete Aktivierungsfunktionen.

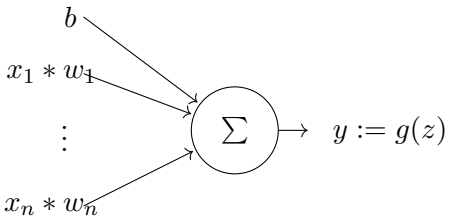


Abbildung 2.2: Ein einzelnes Neuron mit seiner Eingabe $\mathbf{x} = (x_1, ..., x_n)$, den Gewichten $(\mathbf{w} = w_1, ..., w_n)$, dem Bias b und der entsprechenden Ausgabe y. Die Aktivierungsfunktion wird durch g dargestellt. Der Wert z beschreibt die Summe der einzelnen Produkte des Eingabevektors und der Gewichtsmatrix.

dargestellt. Die Ausgabe eines künstlichen Neurons kann durch die mathematische Gleichung

$$y = g(\sum_{i=1}^{n}(x_i * w_i) + b) \tag{2.2}$$

dargestellt werden. Die Abbildung 2.2 stellt den beschriebenen Prozess grafisch dar.

2.1.2 Künstliche Neuronale Netze

Künstliche Neuronale Netze (KNNs) sind mitunter die einfachste Form neuronaler Netze. Sie bestehen aus einer Eingabeschicht, einer verdeckten Schicht und einer Ausgabeschicht. Jede dieser Schichten besteht aus mehreren künstlichen Neuronen. Die Eingabeschicht verarbeitet den Eingabevektor $\mathbf{x}$. Ein einfaches KNN ist in Abbildung 2.3 dargestellt. Das Netzwerk versucht durch Anpassung der Gewichte eine Funktion zu approximieren.
Ein tiefes Feedforward-Netzwerk besitzt mehrere verdeckte Schichten, eine Eingabeschicht und eine Ausgabeschicht. Alle Schichten zwischen der Eingabe- und der Ausgabeschicht

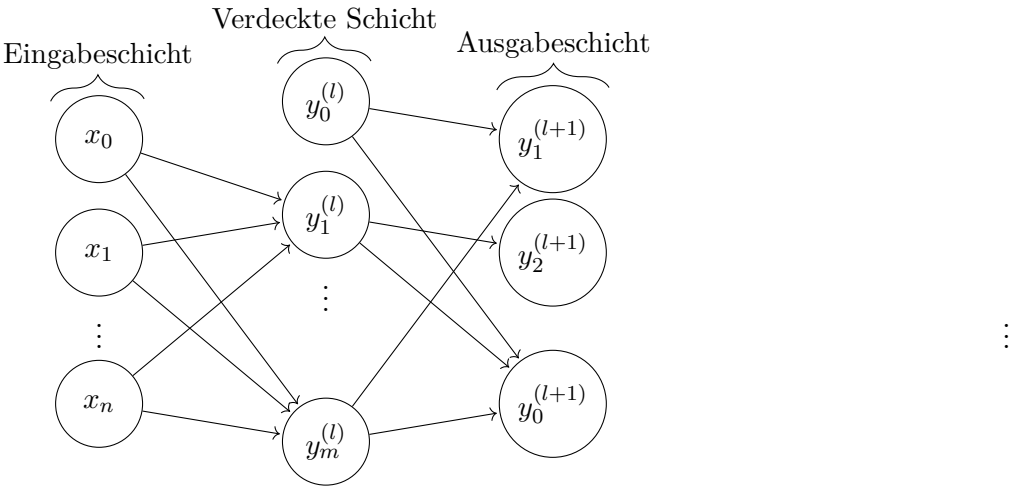


Abbildung 2.3: Beispielgraph für ein KNN mit einer verdeckten Schicht. Das KNN besitzt n Neuronen in der Eingabeschicht und 3 Neuronen in der Ausgabeschicht. Die verdeckte Schicht beinhaltet m verdeckte Neuronen.

werden als verdeckte Schichten bezeichnet. Diese sind für die hauptsächliche Berechnung zuständig. Je tiefer ein Netzwerk ist, das bedeutet je mehr verdeckte Schichten es besitzt, desto komplexere Funktionen kann das Netzwerk approximieren. Für den Feedforward-Prozess sind die Neuronen der $(l-1)$-ten Schicht mit den Neuronen der l-ten Schicht verbunden. So können die Informationen von einer Schicht zur nächsten Schicht weitergegeben werden. Die Gleichung 2.3 zeigt, wie die Informationsweitergabe zwischen den einzelnen Schichten funktioniert. Der Wert $\mathbf{y}^l$ stellt den Ausgabevektor der l-ten Schicht des Netzwerks dar. Ist $l = 0$, wird das Ergebnis direkt aus dem Eingabevektor berechnet. In diesem Fall wird die Eingabe $\mathbf{x}$ mit der Gewichtsmatrix $\mathbf{W}^0$ multipliziert, der Bias b^0 für die erste Schicht addiert (falls vorhanden) und anschließend die gewählte Aktivierungsfunktion g angewendet.
Ist $l > 0$ dient die Ausgabe der letzten Schicht $\mathbf{y}^{(l-1)}$ multipliziert mit der Gewichtsmatrix $\mathbf{W}^l$ für die aktuell betrachtete Schicht l als Grundlage. Auch hier wird, falls ein Bias b^l vorhanden ist, dieser addiert und anschließend die Aktivierungsfunktion g auf das Ergebnis angewendet.

$$\mathbf{y}^l = \begin{cases} g\left(\mathbf{x}\mathbf{W}^0 + \mathbf{b}^0\right), & l = 0 \\ g\left(\mathbf{y}^{l-1}\mathbf{W}^l + \mathbf{b}^l\right), & \text{sonst} \end{cases} \tag{2.3}$$

Abbildung 2.4 zeigt, wie ein solches Feedforward-Netzwerk beispielsweise aussehen kann. Die Anzahl der verdeckten Schichten und die Anzahl der Neuronen je Schicht sind nicht festgelegt. Mittels tiefen Feedforward-Netzwerken können auch komplexere Funktionen approximiert werden. Das Ziel eines solchen Netzwerkes ist es die Ausgabefunktion $g(\mathbf{y})$ so anzupassen, dass sie die Zielfunktion $f^*(\mathbf{x})$ möglichst genau abbildet. Ein weiteres Ziel des Trainings eines Feedforward-Netzes ist, dass das Netzwerk auch gute Ergebnisse für bisher nicht gesehene Daten erzeugt. Diese Fähigkeit wird Generalisierung genannt.
Für das Training werden deshalb mindestens ein Trainings- und ein Testdatensatz benötigt. Ein zusätzlicher Validierungsdatensatz, um die Generalisierungsfähigkeit des Netzwerks zu testen, ist ebenfalls sinnvoll. Die Datensätze sind so aufgebaut, dass aus ihnen hervorgeht, welche Ausgabe (Zielwert) das Netzwerk für eine bestimmte Eingabe erzeu-

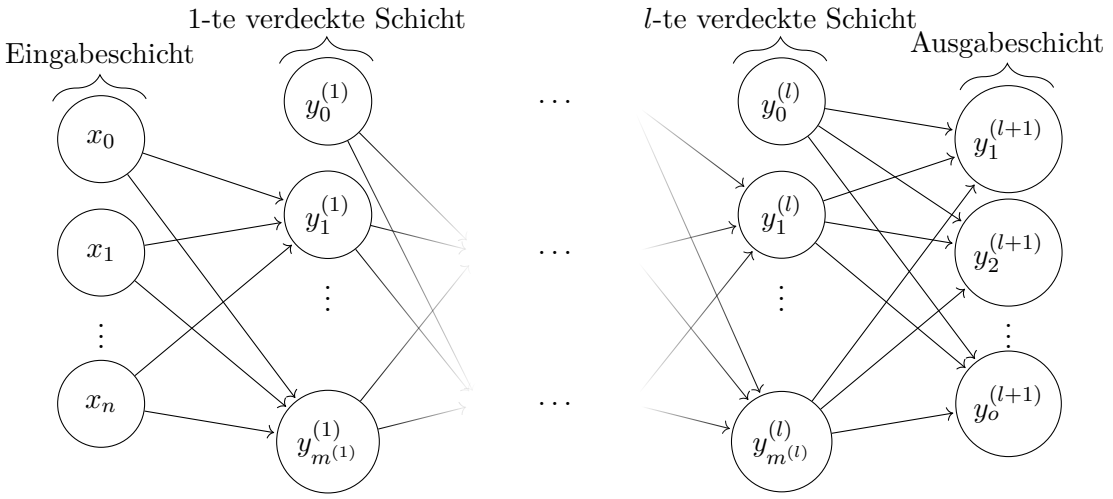


Abbildung 2.4: Beispielgraph für ein Feedforward-Netzwerk mit $(l + 1)$ Schichten mit n Eingabeneuronen und o Ausgabeneuronen. Die l-te verdeckte Schicht beinhaltet $m^{(l)}$ verdeckte Neuronen.

gen soll. Das Netz wird auf dem Trainingsdatensatz trainiert und auf dem Testdatensatz getestet. Die Daten aus dem Test- und Validierungsdatensatz werden nicht für das Training verwendet, damit die tatsächliche Qualität des Netzwerks getestet werden kann. Es besteht sonst die Gefahr, dass das Netzwerk lernt, genau die bereits gesehenen Testdaten zu erzeugen. In diesem Fall würde eine Überanpassung des Modells entstehen.
Overfitting bezeichnet genau dieses Problem, wenn sich das Modell während des Lernprozesses zu sehr auf die betrachteten Daten spezialisiert. Das führt zu einem sehr kleinen Trainingsfehler, da das Modell die betrachteten Trainingsdaten lernt nicht zu zugrundeliegende Wahrscheinlichkeitsverteilung der Daten. In der Praxis kann das daran erkannt werden, dass das Modell auf bisher nicht gesehenen Daten, wie beispielsweise Validierungs- oder Testdaten, nicht gut arbeitet und der Fehler, also die Differenz zwischen Ausgabe und Zielwert, entsprechend hoch ist. Der Gegensatz zum Overfitting ist das Underfitting. In diesem Fall ist das Modell nicht komplex genug und besitzt damit nicht die Möglichkeit, die Wahrscheinlichkeitsverteilung der vorliegenden Daten ausreichend zu lernen. Die Abbildungen 2.5 und 2.7 zeigen jeweils ein Beispiel für ein mögliches Underfitting und ein mögliches Overfitting eines Modells. Die Abbildung 2.6 zeigt eine Modellvorhersage, welche die tatsächliche Funktion gut approximiert [Ped+11]. Deshalb ist es wichtig beim Training zu testen, inwiefern das Netzwerk in der Lage ist zu generalisieren. Dafür ist der Validierungsdatensatz gut geeignet. Wichtig ist, dass Trainings- und Validierungdaten disjunkt sind, damit das Netzwerk die Beispiele aus den Validierungdaten während des Trainings noch nicht gesehen hat. Ein übliches Größenverhältnis zwischen Trainings- und Validierungsdatensatz ist 80 : 20. Tritt trotzdem eine Überanpassung des Modells auf, kann die Dropout-Technik dem Modell helfen, seine Generalisierungsfähigkeit zu steigern [Hin+12]. Bei der Anwendung von Dropout werden innerhalb jeder verdeckten Schicht mit einer frei wählbaren Wahrscheinlichkeit Neuronen ignoriert. Das führt dazu, dass die Berechnung ausschließlich abhängig von den Gewichten und den entsprechenden Merkmalen ist und eine gewisse Co-Abhängigkeit der Neuronen untereinander vermieden wird.
Eine zusätzliche Möglichkeit, um Overfitting zu verhindern, ist aus der Menge der Trainings- und Validierungsdaten die Eingabedaten für eine Trainingsepoche randomisiert zu wählen.

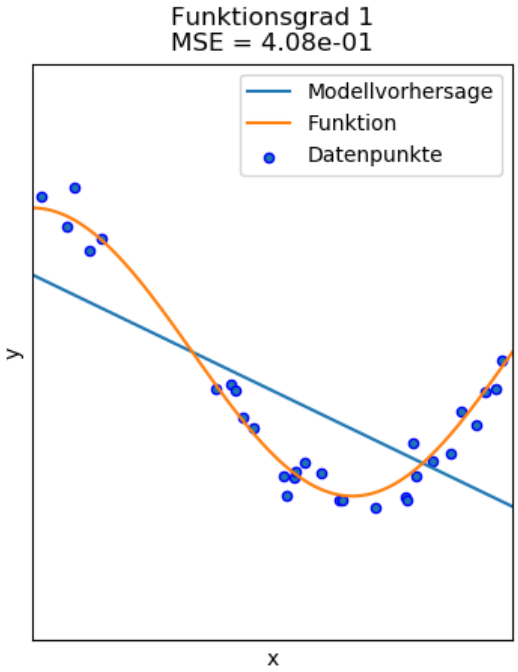


Abbildung 2.5: Beispiel für das Underfitting eines Modells. Der Funktionsgrad, welcher die Vorhersage des Modells annehmen kann, ist deutlich zu gering. Somit ist es dem Modell nicht möglich, die zu lernende Funktion anzunähern.

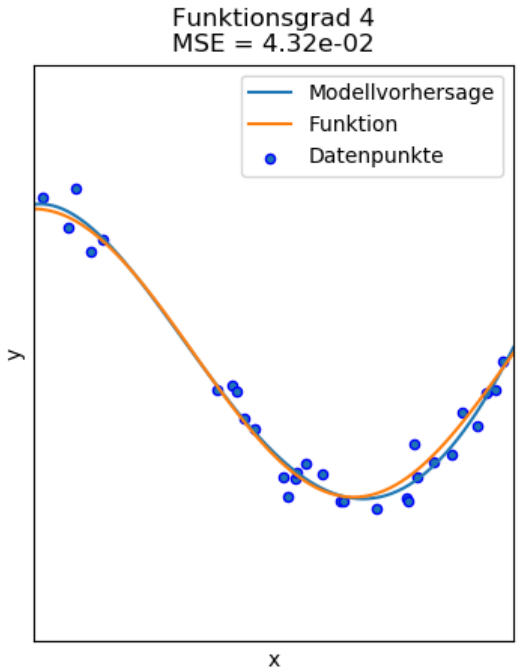


Abbildung 2.6: Diese Abbildung zeigt eine Modellvorhersage, welche die tatsächliche Funktion gut approximiert.

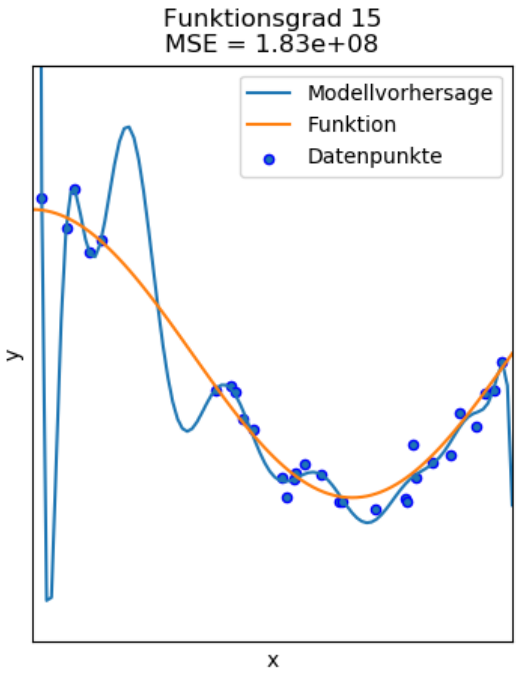


Abbildung 2.7: Beispiel für das Overfitting eines Modells. Es lernt eine deutlich zu komplexe Funktion, welche ausschließlich auf die Datenpunkte in diesem Beispiel passt. Damit ist es dem Modell nicht möglich zu generalisieren.

Das führt dazu, dass das Netzwerk immer noch alle Trainings- und Validierungsdaten berücksichtigt, die Reihenfolge für jede Epoche jedoch verschieden ist.
Während des Trainings können wir mittels verschiedener Fehlermaße prüfen, wie groß die Differenz zwischen dem Zielwert und der vom Netzwerk erzeugten Ausgabe ist. Einige häufig verwendete Fehlermaße für die Vorhersage von Zeitreihen sind beispielsweise der Mittlere Quadratische Fehler (engl. *mean square error*, MSE, siehe Gleichung 2.4) [LC98], der Mittlere Absolute Fehler (engl. *mean absolute error*, MAE, siehe Gleichung 2.5) [SW10], der Mittlere Absolute Prozentuale Fehler (engl. *mean absolute percentage error*, MAPE, siehe Gleichung 2.6) [Lin+18] und die Wurzel der mittleren Fehlerquadratsumme (engl. *root mean square error*, RMSE, siehe Gleichung 2.7) [Bar92].
Für Zeitreihen, welche den Wert 0 beinhalten, ist der MAPE nur bedingt geeignet. Wie wir in Gleichung 2.6 sehen, wird für einen Wert von 0 durch Null geteilt. Ebenfalls resultieren aus Werten nahe der Null sehr groß Werte. Es gibt Erweiterungen dieses Fehlermaßes, welche solche Probleme umgehen. Dazu gehören der *mean arctangent percentage error* (MAAPE) [KK16], der *symmetric mean absolute percentage error* (sMAPE) [MH00] und der *mean absolute scaled error* (MASE) [HK06]. Wir verwenden in dieser Arbeit für unsere Vergleichsexperimente den MAPE, um eine Vergleichbarkeit mit den anderen Modellen herzustellen.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - g\left(y_i\right)\right)^2 \tag{2.4}$$

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}\left|\hat{y}_i - g\left(y_i\right)\right| \tag{2.5}$$

$$\text{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{\hat{y}_i - g\left(y_i\right)}{\hat{y}_i} \right| \tag{2.6}$$

$$\text{RMSE} = \left(\frac{\sum_{i=1}^{n} \left(g\left(y_i\right) - \hat{y}_i\right)^2}{n} \right)^{\frac{1}{2}} \tag{2.7}$$

Das angewendete Fehlermaß wird im Kontext des maschinellen Lernens auch als Verlustfunktion bezeichnet. In den Gleichungen 2.4 bis 2.7 bezeichnet n die Anzahl der Trainingsbeispiele, $\hat{y}_i$ den tatsächlichen Wert und $g\left(y_i\right)$ die Ausgabe des Netzwerks. Der Trainingsfehler beschreibt die Abweichung der erzeugten Ausgabe des Netzwerks und des Zielwertes bezüglich der Trainingsdaten.
Um sicherzustellen, dass das Netzwerk auch auf neuen, ungesehenen Daten gut arbeitet, soll der Generalisierungsfehler ebenfalls während des Trainings minimiert werden. Nach Abschluss des Trainings des Netzwerks kann das Modell auf den Testdaten getestet werden. Der Anteil der Testdatenmenge ist üblicherweise mit einem Anteil von etwa zehn Prozent der Gesamtdatenmenge sehr klein. Die Test- und Validierungsdaten hat das Netzwerk während des Trainings nicht betrachtet. Deshalb sollte ein solcher Datensatz zum Testen der Qualität des Netzwerks verwendet werden. Zu Beginn des Trainings ist die Ausgabe des Netzwerks zufällig und der Trainings- und der Generalisierungsfehler hoch. Da der Generalisierungsfehler oftmals mithilfe des Validierungsdatensatzes bestimmt wird, wird dieser Fehler auch Validierungsfehler genannt. Der hohe Fehler liegt daran, dass das Netzwerk bisher nicht die Möglichkeit hatte, die Gewichte entsprechend anzupassen.
Der Lernprozess des neuronalen Netzwerks besteht darin, die Gewichte mit der Zeit so anzupassen, dass es dem Netzwerk möglich ist, die gewünschten Ausgaben zu erzeugen, welche möglichst nah an dem Zielwert liegen. Es sollen sowohl der Trainingsfehler als auch der Generalisierungsfehler mit der Zeit reduziert werden. Ist es dem Modell nicht möglich den Trainingsfehler zu verringern, liegt eine Unteranpassung des Modells vor. Mögliche Gründe dafür sind eine zu geringe Komplexität des Modells, um die Zielfunktion zu approximieren oder ein zu kleiner Trainingsdatensatz.
Innerhalb dieser Arbeit verwenden wir die Fehlermaße aus den Gleichungen 2.4, 2.5, 2.6 und 2.7. Die Parameter des Modells werden auf Basis des MSEs angepasst. Nach Abschluss des Trainings wird die Generalisierungsfähigkeit des entwickelten Modells auf dem Validierungsdatensatz mithilfe des MAEs, MAPEs und RMSEs beurteilt. Dabei gilt es zu beachten, dass absolute Fehlermaße ausschließlich die absolute Abweichung zwischen der vom Netzwerk erzeugten Vorhersage und der Grundwahrheit messen können. Abhängig davon, in welchem Intervall die Werte des betrachteten Datensatzes liegen, kann die Aussagekraft zwischen einem absoluten und einem relativen Fehlermaß stark abweichen.

2.1.3 Backpropagation und Gradientenabstieg

Für den Lernprozess wird ein Algorithmus benötigt, welcher die Gewichte zwischen allen vorhandenen Schichten stetig anpasst, damit der Fehler der vom Netzwerk erzeugten Ausgabe bezüglich der gewünschten Ausgabe minimiert wird. Eine häufig verwendete Methode ist das Prinzip der Backpropagation, welches auf Rumelhart, Hinton und Williams zurückgeht [RHW88]. Dabei handelt es sich um eine Verallgemeinerung der Delta-Regel und beruht auf dem Gradientenverfahren.
Beim Lernprozess soll das Netzwerk für ein gegebenes Problem mittels gegebenen Trainingsbeispielen (Eingabevektoren) bestimmte Ausgaben (Ausgabevektoren) erzeugen. Das Netzwerk erzeugt für jeden Eingabevektor $\mathbf{x}$ somit einen Ausgabevektor $\mathbf{y}$. Damit das

Netzwerk die inneren Parameter anpassen und somit die Qualität der Ausgabe verbessern kann, werden die Ausgabevektoren mit dem gewünschten Zielwert verglichen. Die Fehlerfunktion verwendet im Fall der Backpropagation den quadratischen Fehler (siehe Gleichung 2.8).
Die Anzahl der betrachteten Trainingsdaten wird durch n repräsentiert. Der Wert t_i beschreibt den gewünschten Zielwert für das Trainingsbeispiel i. Die vom Netzwerk berechnete Ausgabe für das i-te Trainingsbeispiel wird durch y_i dargestellt. Die Differenz der beiden Werte gibt an, wie weit die erzeugte Ausgabe von der gewünschten Ausgabe entfernt ist. Damit diese Differenz immer positiv ist, wird sie anschließend quadriert. Die einzelnen Ergebnisse für die Trainingsbeispiele werden aufsummiert und für eine einfachere Berechnung der Ableitung, welche später noch verwendet wird, mit $\frac{1}{2}$ multipliziert.

$$E = \frac{1}{2} \sum_{i=1}^{n} (t_i - y_i)^2 \tag{2.8}$$

Das Ziel des Netzwerks ist es nun, diesen Fehler im Laufe des Trainingsprozesses zu minimieren. In den meisten Fällen wird bei diesem Prozess ein lokales Minimum gewählt. Das globale Minimum ist aufgrund der hohen Komplexität der Eingabedaten oftmals sehr aufwendig zu finden. Warum das so ist, wird später in diesem Kapitel deutlich.
Nachdem der Fehler berechnet wurde, wird dieser nun von der Ausgabeschicht zurück zur Eingabeschicht propagiert. Bei diesem Prozess werden die Gewichte der Schichten im gesamten Netzwerk angepasst. Dafür wird das Verfahren des Gradientenabstiegs genutzt. Dieses geht auf den französischen Mathematiker Augustin-Louis Cauchy zurück. Dafür wird die Ableitung $f'(x)$ bzw. $\frac{dy}{dx}$ einer Funktion $y = f(x)$ benötigt. Die Werte x und y sind in diesem Fall reelle Zahlen. Die Ableitung der Funktion gibt die Steigung an der Stelle im Punkt x an. Anders gesagt gibt die Steigung an, wie weit der Eingabewert geändert werden muss, um eine Änderung der Ausgabe zu erhalten. Aufgrund dieser Eigenschaft ist die Ableitung für die Minimierung einer Funktion nützlich, da daraus abgelesen werden kann, inwiefern x geändert werden muss, um y zu verändern. Aus den Punkten an denen $f'(x) = 0$ gilt, können keine Informationen über die Bewegungsrichtung der Funktion gegeben werden. Diese Punkte werden als kritische oder stationäre Punkte bezeichnet. An diesen Punkten kann ein Maximum, ein Minimum oder ein Sattelpunkt der Funktion vorliegen. Liegt beispielsweise ein lokales Minimum vor, kann $f(x)$ in diesem Bereich durch infinitesimale Schritte nicht weiter minimiert werden, denn alle Nachbarpunkte weisen einen höheren Funktionswert auf. Analog folgt aus einem lokalen Maximum, dass der Funktionswert in dem lokalen Bereich nicht weiter maximiert werden kann. Eine Funktion hat entweder ein globales Minimum bzw. Maximum oder mehrere globale Minima bzw. Maxima. Für ein optimales Ergebnis ist es wünschenswert, dass während des Optimierungsprozesses genau eine dieser globalen Extremstelle, je nach Optimierungsproblem das Minimum oder das Maximum, gefunden wird.
Aufgrund der sehr komplexen Funktionen, welche beim Deep Learning optimiert werden sollen, besitzen diese Funktionen viele lokale Extremstellen und Sattelpunkte mit einer sehr flachen Umgebung. Dadurch ist es schwierig genau diese globalen Extremstellen zu finden. Deshalb sind manche lokale Extremstellen, welche einen ähnlichen Wert wie die globalen Extremstellen besitzen, oftmals genauso gut geeignet. Diese lokalen Extremstellen benötigen meist einen deutlich geringeren Berechnungsaufwand.
Eine weitere Schwierigkeit besteht darin, dass lokale Extremstellen gefunden werden, welche nicht optimal sind. Diese Schwierigkeit wird durch die Mehrdimensionalität und die somit gesteigerte Komplexität der Eingaben vergrößert. Damit trotz der Mehrdimensio-

nalität der Eingaben eine solche Optimierung möglich ist, muss die Eingabe auf einen skalaren Wert abgebildet werden können. Es gilt $f : \mathbb{R}^n \to \mathbb{R}$.
Um eine solche Abbildung zu ermöglichen, müssen partielle Ableitungen gebildet werden. Eine partielle Ableitung $\frac{\partial}{\partial x_i} f(\mathbf{x})$ gibt die Änderung des Funktionswertes an, wenn nur die Variable x_i des Eingabevektors $\mathbf{x}$ verändert wird. Der Gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ einer Funktion ist der Vektor, welcher alle partiellen Ableitungen der Funktion f für den Eingangsvektor $\mathbf{x}$ beinhaltet.
Betrachten wir nun einen mehrdimensionalen Eingabevektor $\mathbf{x}$, dann sind die stationären Punkte genau diese für die gilt: $\forall x_i \in \mathbf{x}, i \in \mathbb{N}. \frac{\partial}{\partial x_i} f(\mathbf{x}) = 0$.
Um eine Funktion f über einem mehrdimensionalen Eingabevektor $\mathbf{x}$ zu minimieren, müssen wir die Richtung finden, in die der Funktionswert am schnellsten abnimmt. Dafür kann die Richtungsableitung der Funktion genutzt werden. Die Richtungsableitung in Richtung eines Vektors $\mathbf{d}$ (Einheitsvektor) stellt die Steigung der Funktion in Richtung u dar. Die Richtungsableitung kann auch als die Ableitung der Funktion $f(\mathbf{x} + \alpha\mathbf{d})$ bezüglich α für $\alpha = 0$ dargestellt werden.
Für die Berechnung dieser Ableitung muss die Kettenregel $f(\mathbf{x}) = g(h(\mathbf{x})) \to f'(\mathbf{x}) = g'(h(\mathbf{x})) \cdot h'(\mathbf{x})$ angewendet werden. Daraus folgt, dass die Richtungsableitung als Skalarprodukt zwischen dem Richtungsvektor $\mathbf{d}$ und dem Gradienten der Funktion f an der Stelle $\mathbf{x}$ dargestellt werden kann [MK17].

$$\frac{\partial}{\partial \alpha} f(\mathbf{x} + \alpha\mathbf{d}) = \mathbf{d}^\top \nabla_{\mathbf{x}} f(\mathbf{x}) \text{ wenn } \alpha = 0 \tag{2.9}$$

Das Ziel ist es, die Richtung der Funktion f zu finden, in welcher f am schnellsten kleiner wird. Dafür können wir die zuvor bestimmte Richtungsableitung verwenden. Das Skalarprodukt zweier reeller Vektoren, in diesem Fall $\mathbf{d}$ und $\nabla_{\mathbf{x}} f(\mathbf{x})$, kann mithilfe der euklidischen Norm und dem Winkel θ berechnet werden. (siehe Gleichung 2.10) [MK13]. Der Winkel θ bezeichnet den Winkel zwischen den beiden Vektoren.

$$\min_{\mathbf{d}, \mathbf{d}^\top |\mathbf{d}|=1} \mathbf{d}^\top \nabla_{\mathbf{x}} f(\mathbf{x}) = \min_{\mathbf{d}, \mathbf{d}^\top |\mathbf{d}|=1} \|\mathbf{d}\|_2 \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2 \cos\theta \tag{2.10}$$

Zur weiteren Verwendung kann die Gleichung 2.10 wie folgt vereinfacht werden:

$$\min_{\mathbf{d}, \mathbf{d}^\top |\mathbf{d}|=1} \|\mathbf{d}\|_2 \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2 \cos\theta = \min_{\mathbf{d}} \cos\theta \tag{2.11}$$

Für diese Vereinfachung setzen wir $\|\mathbf{d}\|_2 = 1$, da $\mathbf{d}$ ein Einheitsvektor ist und ignorieren für die Vereinfachung alle nicht von $\mathbf{d}$ abhängigen Faktoren. Für die Minimierung sind ausschließlich die Faktoren interessant, welche von dem Richtungsvektor $\mathbf{d}$ abhängig sind. Das Ziel der Minimierung ist dann erreicht, wenn $\mathbf{d}$ in die dem Gradienten entgegengesetzte Richtung zeigt. Das bedeutet, dass der Gradient der Funktion f Null, also $\nabla f = 0$, ist. Der Funktionswert von f kann dann verkleinert werden, wenn die Werte entsprechend der Richtung des negativen Gradienten angepasst werden. Aufgrund dieser Eigenschaft wird das Verfahren Gradientenabstieg oder Methode des steilsten Abstiegs genannt [GB18]. Wird der Weg des steilsten Abstiegs verfolgt, entsteht daraus ein neuer Punkt $\mathbf{x}' = \mathbf{x} - \eta \nabla_{\mathbf{x}} f(\mathbf{x})$.
Der Parameter η kann unterschiedliche Werte annehmen. Es handelt sich dabei um einen sehr kleinen, positiven skalaren Wert, welcher im Kontext des maschinellen Lernens auch als Lernrate bezeichnet wird. Dieser Parameter bestimmt die Schrittweite, mit welcher die Parameter angepasst werden. In der Praxis ist ein gängiger Anfangswert $\eta = 0.001$ [Ben12]. In einigen Modellen wird die Lernrate im Laufe des Trainings angepasst, um eine

eine geeignete lokale Extremstelle zu finden. Das Ziel des Abstiegs ist erreicht, wenn jedes Element des berechneten Gradienten Null ist. Dann ist eine globale Extremstelle erreicht. Da dies in der Praxis nicht immer möglich ist, sondern oftmals eine geeignete lokale Extremstelle gewählt wird, sind die einzelnen Elemente des Gradienten in der Praxis oftmals nahe der Null.
Bei vielen Deep Learning Verfahren wird für die Anpassung der Gewichte innerhalb des Netzwerks der stochastische Gradientenabstieg (engl. *stochastic gradient descent*, SGD) angewendet. Dieses Verfahren ist auf Robbins und Monro zurückzuführen [RM51]. Es ist eine Erweiterung des beschriebenen Gradientenabstiegs. Um einen geringen Generalisierungsfehler zu erhalten, müssen Deep Learning Modelle oftmals auf einer großen Menge von Trainingsdaten trainiert werden. Bei einer großen Menge an Trainingsdaten steigt der Rechenaufwand für den Gradientenabstieg immer weiter an. Aus dem stochastischen Gradientenabstieg geht hervor, dass der Gradient wie ein Erwartungswert behandelt werden kann [GBC16]. Dieser Erwartungswert kann mithilfe einer verhältnismäßig kleinen Menge von Stichproben aus dem Trainingsdatensatz geschätzt werden. Diese kleine Menge an Trainingsdaten wird häufig Mini-Batch genannt. Die Batchgröße gibt an, wie viele Beispiele das Modell je Iteration betrachtet. Oftmals beträgt sie nur einen Bruchteil der Größe der Trainingsdaten und liegt zwischen einem Trainingsbeispiel und mehreren hundert Beispielen. Es gibt jedoch auch Modelle, bei denen die gesamten Trainingsdaten als ein Batch verarbeitet werden. Wichtig ist, eine geeignete Batchgröße für das zu testende Modell zu finden und diese während des Trainings unverändert zu lassen. Auch wenn die Größe der Trainingsdaten geändert wird, beispielsweise durch einen größeren Trainingsdatensatz, sollte eine funktionierende Batchgröße des Modells nicht verändert werden. Dadurch kann die Verlust- bzw. Kostenfunktion des gesamten Algorithmus als Summe der Verlustfunktionen, angewendet auf eine kleinere Menge von Trainingsbeispielen, dargestellt werden [GBC16]. So kann der Schätzwert des Gradienten $\mathbf{g}$ mit einer Batchgröße von m' wie folgt gebildet werden:

$$\mathbf{g} = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L\left(\mathbf{x}^{(i)}, y^{(i)}, \theta\right) \tag{2.12}$$

Dabei werden jeweils m' Trainingsbeispiele je Batch betrachtet. Der Verlust wird durch $L\left(\mathbf{x}^{(i)}, y^{(i)}, \theta\right)$ beschrieben. Für ein Minimierungsproblem muss der Algorithmus dem Gradienten abwärts folgen (siehe Gleichung 2.13).

$$\theta \leftarrow \theta - \eta \mathbf{g} \tag{2.13}$$

2.1.4 Adam-Optimierung

Die Adam-Optimierung ist ein im Kontext des Deep Learnings häufig verwendeter Optimierungsalgorithmus [KB15]. Sie bietet eine effiziente stochastische Optimierung mit anpassbarer Lernrate. Zu Beginn des Trainings wird eine Anfangslernrate angegeben. Von da aus passt die Optimierungsmethode die Lernrate mit fortschreitendem Training immer wieder an. Für die Optimierung werden ausschließlich Gradienten ersten Grades benötigt. Der Optimierungsalgorithmus erhält als Eingabe eine Schrittgröße α, welche angibt, inwiefern die Parameter mit jedem Berechnungsschritt geändert werden. Zusätzlich werden zwei Verkleinerungsraten $\beta_1, \beta_2, \in [0, 1)$ für die Momentenschätzer benötigt. Für die Anwendung benötigt der Algorithmus außerdem eine stochastische Funktion $f(\theta)$ mit einem initialen Parametervektor θ.

Zu Beginn werden der erste Momentenschätzer m_0, der zweite Momentenschätzer v_0 und der erste Zeitstempel t mit Nullen initialisiert. Solange der Parametervektor θ_t nicht konvergiert, versucht der Algorithmus die Parameter weiter zu optimieren.
Sei $f(\theta)$ ein ableitbarer stochastischer Prozess mit Bezug zu seinen Parametern θ. Das Ziel der Optimierung ist es, den Erwartungswert $\mathbb{E}[f(\theta)]$ dieses Prozesses zu minimieren. Die Funktionen $f_1(\theta), \ldots, f_T(\theta)$ beschreiben die Ausführung des Prozesses an bestimmten Zeitpunkten $1, \ldots, T$. Der Gradient wird durch $g_t = \nabla_\theta f_t(\theta)$ dargestellt. Das kann beispielsweise der Vektor der partiellen Ableitungen von f_t mit Bezug auf θ zu einem Zeitpunkt t sein.
Der Algorithmus passt die gleitende Durchschnitte des Gradienten (m_t) und den quadrierten Gradienten (v_t) an. Dafür verwendet der Algorithmus die Hyperparameter β_1 und β_2. Diese Parameter steuern die exponentielle Verringerung dieser gleitenden Durchschnitte. Für die Optimierung im Zeitschritt t wird der Gradient der Funktion $f_t(\theta_{t-1})$ durch $g_t = \nabla_\theta f_t(\theta_{t-1})$ berechnet. Anschließend werden mittels β_1 und β_2 die Parameter m_{t-1}, v_{t-1}, m_t und v_t wie folgt berechnet:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{2.14}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2.15}$$

Nun werden die gleitende Durchschnitte mittels eines Bias korrigiert:

$$\widehat{m}_t = m_t / \left(1 - \beta_1^t\right) \tag{2.16}$$

$$\widehat{v}_t = v_t / \left(1 - \beta_2^t\right) \tag{2.17}$$

Mit diesen Variablen kann anschließend der Parametervektor θ angepasst werden:

$$\theta_t = \theta_{t-1} - \alpha \cdot \widehat{m}_t / \left(\sqrt{\widehat{v}_t} + \eta\right) \tag{2.18}$$

Mittels dieses Prozesses werden die Parameter innerhalb von θ solange angepasst, bis diese konvergieren. Da für dieses Verfahren ausschließlich Gradienten ersten Grades benötigt werden, ist der Zeit- und Speicherbedarf geringer, als beim Stochastischen Gradientenabstieg.

2.2 Convolutional Neural Networks

Die in Abschnitt 2.1.2 beschrieben tiefen Feedforward-Netzwerke bilden die Grundlage für Convolutional Neural Networks (CNNs). CNNs gehen auf LeCun und Bengio zurück [LB95]. Diese Art von Netzwerken ist besonders gut für rasterförmige Daten geeignet. Ein Beispiel für solche Daten sind Bilder. Sie bestehen aus einem mehrdimensionalen Raster aus Pixeln. Die Pixel enthalten entsprechende Farbinformationen. Der Begriff „Convolution“ bedeutet Faltung. Dabei handelt es sich um einen mathematischen Faltungsoperator, welcher während der Verarbeitung der Daten in einem CNN verwendet wird. Es handelt sich genau dann um ein CNN, wenn ein neuronales Netzwerk anstelle der sonst verwendeten Matrixmultiplikation den genannten Faltungsoperator verwendet [GBC16].

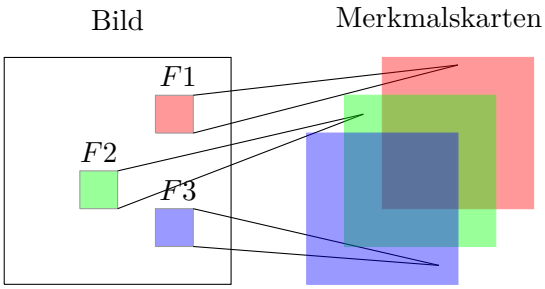


Abbildung 2.8: Darstellung der Erzeugung von Merkmalskarten durch unterschiedliche Kernel. Als Eingabe dient ein RGB-Bild. $F1$,$F2$ und $F3$ stellen die einzelnen Kernel der jeweiligen Ebenen dar. Jeder Kernel erzeugt seine eigene Merkmalskarte als Ausgabe.

2.2.1 Faltung

Die Faltungsoperation $*$ ist eine lineare Operation, welche auf zwei Funktionen f und g angewendet werden kann. Das Ergebnis $y(x) = (f * g)(x)$ dieser Operation kann mit dem gewichteten Mittelwert der Funktion f verglichen werden.
Folgende Gleichung resultiert daraus, wenn die Faltungsoperation auf jeden Eingabewert x angewendet wird:

$$y(x) = \int f(a)g(x-a)da \tag{2.19}$$

Dabei stellt g die Gewichtung von f dar. Die Variable a gibt an, wie stark dieser Wert in das Ergebnis mit eingeht. Daraus folgt für die Gleichung 2.19 folgende Schreibweise:

$$y(x) = (f * g)(x) \tag{2.20}$$

Der Faltungsoperator $*$ ist für alle Funktionen definiert, in denen das Integral aus der Gleichung 2.19 definiert ist. Im Bezug auf CNNs stellt die Funktion f die Eingabe des Faltungsoperators $*$ dar. Die Funktion g wird im Kontext der CNNs als Kernel oder Filter bezeichnet. Die Kernel sind meist sehr klein im Verhältnis zur Eingabe und quadratisch. Es können jedoch auch, abhängig vom verwendeten Modell, rechteckige Filter verwendet werden. Die Ausgabe der Operation wird Merkmalskarte oder auch Featuremap genannt. Als Eingabe liegen meistens mehrdimensionale Daten vor. Dabei kann es sich um die direkten Eingabedaten oder um vorher erzeugte Featuremaps handeln. Bei einem farbigen Bild ist die Eingabe beispielsweise dreidimensional. Zwei Dimensionen beschreiben die Höhe und die Breite des Bildes. Die Werte der drei RGB-Farbkanäle stellen die Tiefe und damit die dritte Dimension der Eingabe dar. Die Kernel dienen dazu, bestimmte Merkmale der Eingabe zu extrahieren bzw. zu erkennen. Für jedes Merkmal wird ein eigener Kernel verwendet. In Abbildung 2.8 ist vereinfacht dargestellt, wie unterschiedliche Kernel auf einer dreidimensionalen Eingabe arbeiten und jeweils eine Merkmalskarte als Ausgabe erzeugen.
Abhängig von der verwendeten Kerneloperation und der Kernelgröße wird aus dem vom Kernel betrachteten Ausschnitt ein gewichteter Mittelwert erzeugt. Dieser Wert stellt einen Punkt in der Merkmalskarte dar. Die Größe der Merkmalskarte ist abhängig von der Schrittweite (Stride) des Kernels und ob der Filter über den Rand der Eingabe hinaus geschoben wird (Padding). Wird Padding verwendet, wird die Eingabe in ihrer Länge und

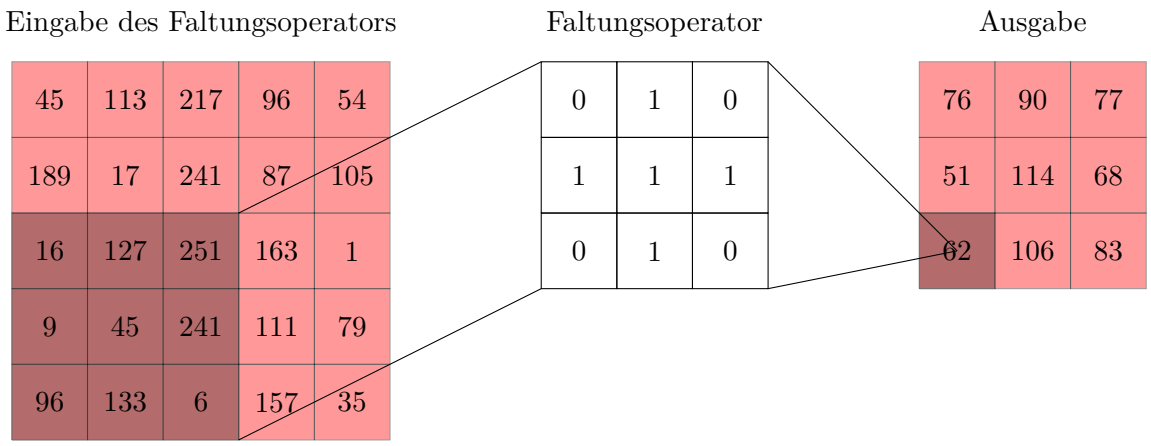


Abbildung 2.9: Beispiel der Anwendung eines Faltungsoperators auf einer zweidimensionalen Eingabe. Die Einträge des Faltungsoperators sind veränderbare Parameter, welche im Laufe des Trainings vom Modell angepasst werden. Die Ausgabe ist eine Merkmalskarte, welche als Eingabe einer neuen Faltungsoperation oder einer Pooling-Schicht dienen kann. In diesem Beispiel werden die Einträge des Filters mit den Einträgen der Eingabe multipliziert, anschließend addiert und durch die Anzahl der Einträge des Filteroperators geteilt.

Breite mittels leerer Werte vergrößert. Merkmale, welche in den Randbereichen der Eingabe auftreten, können somit besser erkannt werden. Der Stride gibt an, wie weit der Kernel je Berechnungsschritt verschoben wird. Bei einer Eingabe der Größe 5×5 besitzt das zweidimensionale Eingabefeld eine Größe von 25. Wird nun ein Kernel der Größe 3×3 mit einer Schrittgröße von 1 und ohne Padding über die Eingabe geschoben, resultiert daraus eine Merkmalskarte mit der Größe 3×3. Eine grafische Darstellung dieses Prozesses ist in Abbildung 2.9 dargestellt. Hier wird eine Ebene als Eingabe betrachtet. Das Prinzip kann vereinfacht werden, indem ein Bild, welches ausschließlich aus Graustufen besteht, betrachtet wird. Dann liegt eine zweidimensionale Eingabe B für einen zweidimensionalen Kernel K vor. Für einen solchen Fall können wir folgende Gleichung aufstellen:

$$S(i,j) = (B * K)(i,j) = \sum_m \sum_n B(m,n) K(i-m, j-n) \tag{2.21}$$

Dabei beschreibt $S(i,j)$ den Wert nach der Faltung von B mittels K über (i,j). Da die Faltung eine kommutative Operation ist, können die Argumente der Funktionen getauscht werden und daraus resultiert folgende Formel:

$$S(i,j) = (K * B)(i,j) = \sum_m \sum_n K(i-m, j-n) B(m,n) \tag{2.22}$$

Die Parameter innerhalb des Kernels werden während des Lernprozesses angepasst. Das sind die Werte, welche das neuronale Netz erlernen soll. Dabei handelt es sich um die einzelnen Gewichte des Kernels. Deshalb ist Formel 2.22 einfacher in der Praxis zu implementieren, da es deutlich weniger gültige Werte für m und n gibt, als in der Formel 2.21. Die beiden Formeln sind deshalb äquivalent, da der verwendete Kernel quadratisch ist und dieser in der zweiten Gleichung horizontal und vertikal gespiegelt wurde.
Im Vergleich zu Feedforward-Netzwerken, in denen häufig jedes Neuron einer Schicht mit

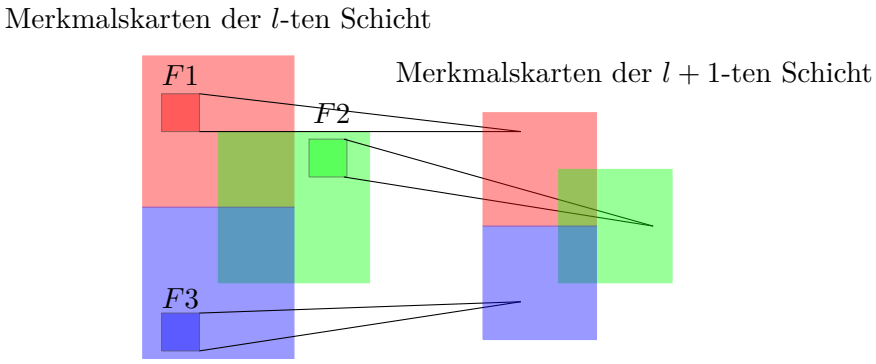


Abbildung 2.10: Darstellung einer einzelnen Poolingschicht. Als Eingabe dienen die vorher erzeugten Merkmalskarten aus der Faltungsoperation und der anschließend angewendeten Aktivierungsfunktion. $F1$,$F2$ und $F3$ stellen die einzelnen Kernel für die jeweiligen Merkmalskarten dar. Jeder Kernel erzeugt seine eigene Merkmalskarte als Ausgabe.

jedem Neuron der folgenden Schicht verknüpft (vollständig vernetzt) ist, interagieren bei CNNs ausschließlich benachbarte Neuronen miteinander. Das liegt daran, da die verwendeten Kernel deutlich kleiner als die tatsächliche Eingabegröße sind. Kernel extrahieren bestimmte Merkmale aus der Eingabe und diese Merkmale können selbst aus sehr großen Eingaben mittels kleiner Kernel bestimmt werden. Der Vorteil dieser Abbildungen ist, dass deutlich weniger Parameter im Vergleich zu tiefen Feedforward-Netzwerken, welche eine große Anzahl an Neuronen je Schicht beinhalten, gespeichert werden müssen. Das reduziert nicht nur den Speicherplatzbedarf eines CNNs gegenüber eines vollständig vernetzten Feedforward-Netzwerks, sondern auch die Berechnungszeit. Wenn wir ein Netzwerk mit x Eingaben und y Ausgaben betrachten, dann benötigt ein vollständig vernetztes Modell die Laufzeit $\mathcal{O}(x \cdot y)$. Diese resultiert aus der Matrizenmultiplikation der Eingabe mit der Gewichtsmatrix.
In einem CNN kann die Anzahl der Verbindungen zwischen den Neuronen auf einen Parameter z beschränkt werden, für den $z < y$ gilt. In diesem Fall benötigt das Modell eine Laufzeit von $\mathcal{O}(x \cdot z)$. Mittels Faltung können also effizient Merkmale aus einer Eingabe extrahiert werden. Auf die Ergebnisse einer Faltung werden in der Praxis häufig Aktivierungsfunktionen angewendet, um zu regulieren, welche Ausgaben für das Ergebnis relevant sind.

2.2.2 Pooling

Nach der Faltung und der Anwendung einer Aktivierungsfunktion wird auf dem berechneten Ergebnis meist eine Poolingoperation ausgeführt. Durch die Poolingoperation wird die Ausgabe auf das Wesentliche reduziert und erneut verkleinert [GBC16]. Dabei werden ähnlich wie bei der Faltung benachbarte Bereiche betrachtet und ein kleiner quadratischer Filter wird schrittweise über die erzeugte Merkmalskarte geschoben. Daraus resultieren kleinere Merkmalskarten. Eine vereinfachte Abbildung dieses Prozesses ist in Grafik 2.10 dargestellt.
In der Praxis wird als Poolingoperation häufig Max-Pooling verwendet (siehe Abbildung 2.11). Bei dieser der Poolingoperation wird ein kleinerer Filter als bei der Faltungsoperation verwendet. Aus dem vom Filter betrachteten Bereich wird bei Max-Pool das Maximum

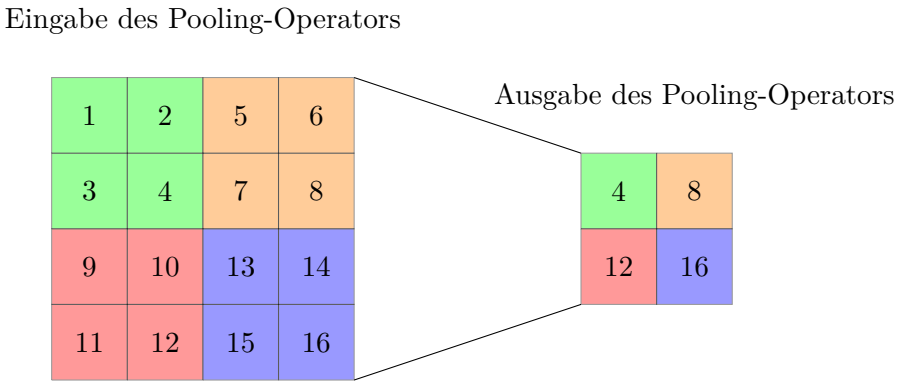


Abbildung 2.11: Anwendung einer Max-Pool-Operation mit einem 2×2 Filter und einer Schrittweite von 2 auf einer Eingabe-Merkmalskarte. Es wird kein Padding verwendet. Der Filter wählt aus dem betrachteten Bereich das Maximum aus.

gewählt. Dadurch werden unwichtige Informationen verworfen und die Zwischenergebnisse gefiltert. Ein weiterer Vorteil der Poolingoperation ist, dass die Zwischenergebnisse invariant bezüglich kleinerer Änderungen in den Eingabedaten werden [Zhe+16]. Diese Invarianz ist nützlich, falls das verwendete Modell erkennen möchte, ob bestimmte Merkmale in den Eingabedaten vorhanden sind. Dann ist die exakte Position dieser Merkmale für die Ausgabe des Modells nicht relevant, sondern nur ob das Merkmal vorhanden ist oder nicht.

2.2.3 Vollständig vernetzte Schicht

Der Vorgang aus Faltung, Aktivierungsfunktion und Pooling kann beliebig oft wiederholt werden. Jede Schicht steigert sowohl die Komplexität des Netzwerks als die der extrahierten Merkmale. Ist die Extraktion der Merkmale abgeschlossen, folgen auf die letzte Poolingoperation eine vollständig vernetzte Schicht und die Ausgabeschicht. Die vollständig vernetzte Schicht wird vor allem bei Klassifizierungsproblemen eingesetzt, also wenn das Netz die Eingabedaten einer bestimmten Klasse zuordnen soll. Die Anzahl der Neuronen in der vollständig vernetzten Schicht ist davon abhängig, wie viele Merkmalskarten erzeugt wurden und wie groß die einzelnen Merkmalskarten sind. Die bisher erzeugte Ausgabe wird ausgerollt, sodass jedes Element der bisherigen Ausgabe einem Neuron in der vollständig vernetzten Schicht zugeordnet wird. Die Ausgabeschicht enthält so viele Neuronen, wie es mögliche Klassen gibt. Grafik 2.12 zeigt den Aufbau einer vollständig vernetzten Schicht und ihre Anbindung an das restliche Netzwerk.

2.3 Geometrisches Deep Learning

Als Geometrisches Deep Learning (GDL) werden Methoden bezeichnet, welche tiefe neuronale Modelle für nicht-euklidische Bereiche generalisieren [Bro+16]. Als nicht-euklidische Bereiche zählen beispielsweise Graphen und Mannigfaltigkeiten. In dieser Arbeit betrachten wir hauptsächlich Geometrisches Deep Learning mit Bezug auf Graphen. Deshalb wird der Zusammenhang zwischen Geometrischem Deep Learning und Mannigfaltigkeiten nicht weiter behandelt.

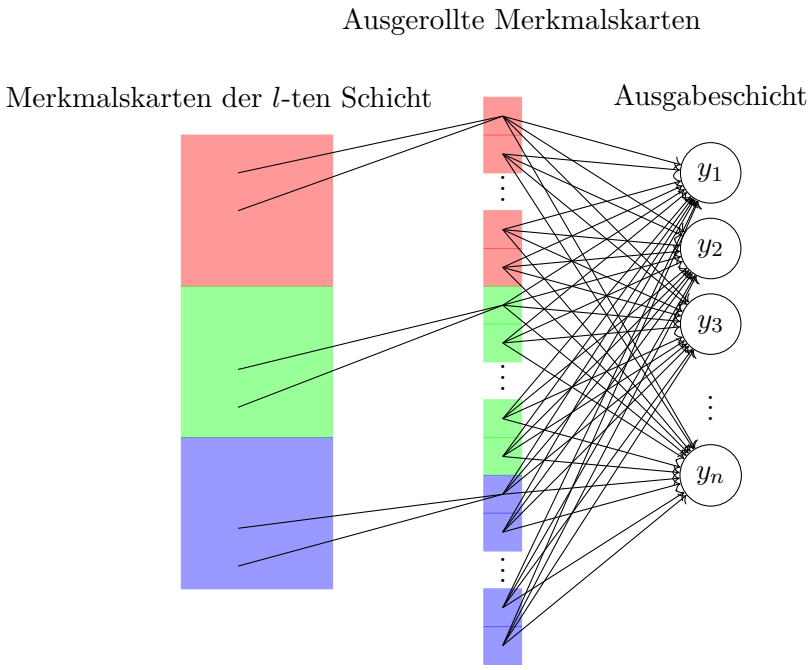


Abbildung 2.12: Der Aufbau einer vollständig vernetzten Schicht am Ende eines CNNs. Mittels dieser Schicht werden die zuletzt erzeugten Merkmalskarten ausgerollt (geglättet) und dienen als Eingabe für die Ausgabeschicht.

Es gibt zwei Klassen Geometrischer Deep Learning Probleme [Bro+16]. Das Ziel der ersten Klasse ist die Charakterisierung der Struktur der Eingabedaten. Das Ziel der zweiten Klasse ist die Analyse der Funktionen, welche auf einem nicht-euklidischen Bereich definiert sind. In dieser Arbeit wird die zweite Problemklasse betrachtet. Es existiert ein fest definierter Bereich (Netzwerk) und Daten bezüglich des Graphen sollen vorhergesagt werden. Das Problem des Geometrischen Deep Learnings ist, den Faltungsoperator auf nicht-euklidische Daten anzuwenden.
Ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ besteht aus einer Knotenmenge $\mathcal{V}$ mit $|\mathcal{V}| = n$ und einer Kantenmenge $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ mit $|\mathcal{E}| = m$. Der Parameter n gibt die Anzahl der Knoten innerhalb des Graphen an und m beschreibt die Anzahl der Kanten. Die Kanten repräsentieren die Konnektivität zwischen den Knoten. Diese können gerichtete oder ungerichtet sein. Dementsprechend ist der gesamte Graph gerichtet oder ungerichtet. Für die Übersichtlichkeit werden in diesem Unterkapitel ausschließlich ungerichtete Graphen betrachtet. Das bedeutet, wenn zwischen zwei Knoten i und j eine Kante existiert, also $(i, j) \in \mathcal{E}$, dann gilt zusätzlich $(j, i) \in \mathcal{E}$.
Jeder Knoten $i \in \mathcal{V}$ bekommt ein Gewicht bzw. ein Knotenmerkmal a_i zugeordnet. Für alle a_i gilt: $a_i > 0$. Die Matrix über die Knotengewichte ist $\mathbf{A} = \operatorname{diag}(a_1, \ldots, a_n)$. Jede Kante $(i, j) \in \mathcal{E}$ erhält ein Kantengewicht $w_{i,j}$ für das gilt: $w_{ij} \geq 0$. Die Gewichtsmatrix wird durch $\mathbf{W} = (w_{ij})$ dargestellt und enthält die Kantengewichte für alle Knotenpaare $(i, j) \in \mathcal{E}$. Für $(i, j) \notin \mathcal{E}$ gilt $w_{i,j} = 0$.
Zusätzlich wird im Verlauf des Kapitels noch die Matrix $\mathbf{D} = \operatorname{diag}\left(\sum_{j:j\neq i} w_{ij}\right)$ benötigt. Die Einträge dieser Matrix stellen die Knotengrade jedes Knotens dar. Es werden nun zwei reelle Funktionen auf der Knotenmenge und der Kantenmenge des Graphen definiert, welche im Verlauf dieses Kapitel noch benötigt werden. Die Funktion $f : \mathcal{V} \to \mathbb{R}$

bildet von der Knotenmenge in die reellen Zahlen ab. Die Funktion $F : \mathcal{E} \to \mathbb{R}$ bildet von der Kantenmenge in die reellen Zahlen ab.
Seien $L^2(\mathcal{V})$ und $L^2(\mathcal{E})$ zwei Hilberträume bzgl. des Skalarprodukts, definiert auf der Knoten- und der Kantenmenge [Bro+16].
Hilberträume gehen zurück auf den deutschen Mathematiker David Hilbert. Ein Hilbertraum beschreibt einen Vektorraum über dem Körper der reellen oder auch über den komplexen Zahlen. Dieser Raum wird zusätzlich mit einem Skalarprodukt versehen und ist vollständig bezüglich der vom Skalarprodukt induzierten Norm [D18].

$$\langle f, g \rangle_{L^2(\mathcal{V})} = \sum_{i \in \mathcal{V}} a_i f_i g_i \tag{2.23}$$

$$\langle F, G \rangle_{L^2(\mathcal{E})} = \sum_{(i,j) \in \mathcal{E}} w_{ij} F_{ij} G_{ij} \tag{2.24}$$

Für die Funktionen f und F gilt: $f \in L^2(\mathcal{V})$ und $F \in L^2(\mathcal{E})$. Wir nehmen zusätzlich an, dass $F_{ij} = -Fji$ gilt. Zusätzlich wird der Gradient des Graphen benötigt. Der Gradient eines Graphen ist ein Operator, welcher Funktionen aus dem Hilbertraum der Knotenmenge auf Funktionen des Hilbertraums der Kantenmenge abbildet. Es gilt also: $\nabla : L^2(\mathcal{V}) \to L^2(\mathcal{E})$. Daraus folgt:

$$(\nabla f)_{ij} = f_i - f_j \tag{2.25}$$

Die Gleichung 2.25 erfüllt auch unsere Bedingung $(\nabla f)_{ij} = -(\nabla f)_{ji}$.
Die Divergenz des Graphen ist ein Operator, welcher entgegengesetzt des Gradienten funktioniert. Er bildet Elemente aus dem Hilbertraum der Kantenmenge auf Elemente des Hilbertraums der Knotenmenge ab. Der Divergenzoperator ist wie folgt definiert:

$$(\operatorname{div} F)_i = \frac{1}{a_i} \sum_{j:(i,j) \in \mathcal{E}} w_{ij} F_{ij} \tag{2.26}$$

Der Gradient und die Divergenz sind bezüglich ihrer inneren Produkte 2.23 und 2.24 komplementär zueinander.

$$\langle F, \nabla f \rangle_{L^2(\mathcal{E})} = \langle \nabla^* F, f \rangle_{L^2(\mathcal{V})} = \langle -\operatorname{div} F, f \rangle_{L^2(\mathcal{V})} \tag{2.27}$$

Dabei bezeichnet ∇^* den transponierten-konjugierten bzw. den adjungierten Gradienten.
Nun wird zusätzlich der Laplace-Operator des Graphen benötigt. Ein diskretes Beispiel für einen Laplace-Operator ist die Laplace-Matrix eines Graphen. Sie beschreibt, welche Beziehung Knoten und Kanten eines Graphen zueinander haben. Aufgrund des definierten Gradienten und der definierten Divergenz kann der Laplace-Operator $\Delta : L^2(\mathcal{V}) \to L^2(\mathcal{V})$ wie folgt definiert werden:

$$\Delta = -\operatorname{div} \nabla \tag{2.28}$$

Mithilfe der Formel für den Gradienten 2.25 und der Formel für die Divergenz 2.26 kann der Laplace-Operator wie folgt aufgeschrieben werden:

$$(\Delta f)_i = \frac{1}{a_i} \sum_{(i,j) \in \mathcal{E}} w_{ij} \left(f_i - f_j \right) \tag{2.29}$$

Unter Berücksichtigung der Matrizen $\mathbf{W}$,$\mathbf{A}$ und $\mathbf{D}$ kann die Darstellung der Anwendung des Laplace-Operators auf einer Funktion $f \in L^2(\mathcal{V})$ als Spaltvektor $\mathbf{f} = (f_1, \ldots, f_n)^\top$ dargestellt werden:

$$\Delta \mathbf{f} = \mathbf{A}^{-1}(\mathbf{D} - \mathbf{W})\mathbf{f} \tag{2.30}$$

Um die Faltung auf nicht-euklidischen Daten anwenden zu können, muss nun der Faltungsoperator, wie er in Gleichung 2.19 definiert ist, angepasst werden. Leider kann die dort verwendete Operation $(x - a)$ nicht einfach auf Graphen übertragen werden. Deshalb muss nach einer Möglichkeit gesucht werden, den Faltungsoperator zu generalisieren. Für diese Generalisierung kann das Faltungstheorem verwendet werden. Dieses besagt, dass die Faltung zweier Funktionen als Produkt der jeweiligen Fouriertransformierten dargestellt werden kann.

$$(f \star g)(x) = \sum_{i \geq 0} \langle f, \phi_i \rangle_{L^2(X)} \langle g, \phi_i \rangle_{L^2(X)} \phi_i(x) \tag{2.31}$$

Eine Möglichkeit für die Generalisierung stammt von Bruna et al. [Bru+14]. Für diese Generalisierung werden die Eigenwerte des Laplace-Operators verwendet. Diese werden auch als Spektrum bezeichnet. Aufbauend auf der Arbeit von Bruna et. Al [Bru+14] entwickelten Defferard et al. [DBV16] eine Filtermethode, welche deutlich effizienter arbeitet. Es werden keine expliziten Berechnungen der Eigenvektoren des Laplace-Operators mehr benötigt. Stattdessen werden Chebyshev-Polynome eingesetzt. Eine weitere Vereinfachung der Methode von Defferard et al. wurde von Kipf und Welling [KW16] vorgenommen. Bei dieser Methode verwenden die Filter 1-Hop-Nachbarschaften des Graphen.
Mittels der Generalisierungen der Faltungsoperationen für Graphen als Eingabedaten, können für vielfältige Aufgaben Graph Convolutional Neural Networks eingesetzt werden.

2.4 Graph Convolutional Neural Networks

Graph Convolutional Neural Networks (Graph-CNNs) bieten die Möglichkeit, die Eigenschaften von CNNs auf geometrische Eingabedaten, wie beispielsweise Graphen, zu übertragen. Der Begriff „Graph-CNN“ wird als Sammelbegriff für alle Convolutional Neural Networks verwendet, welche geometrische Eingabedaten verarbeiten können. Entsprechend der Domäne, auf welcher der Filter des CNN-Modells arbeitet, existieren unterschiedliche Graph-CNNs. Hauptsächlich wird zwischen spektralen und räumlichen Methoden unterschieden. Die spektralen Methoden basieren auf der spektralen Graphentheorie [Chu97]. Ein Beispiel dafür sind die Spectral-CNNs [Bru+14]. Diese interpretieren die Eigenvektoren des Laplace-Operators als Fourierbasis. Als Erweiterungen dieses Ansatzes gelten ChebNet [DBV16] und GNNs [Sca+09]. Diese Modelle besitzen effizientere Filtervarianten. ChebNet verwendet Chebyshev-Polynome als Approximation des Filters. In diesen Modellen wird ausgenutzt, dass sich ein Polynom des Laplace-Operators genauso verhält, wie ein Polynom seiner Eigenwerte. Dies erlaubt die Darstellung der Filter mittels Polynomerweiterung.
Bei räumlichen Methoden wird die Faltung direkt auf der Graphstruktur durchgeführt, indem die Informationen der Nachbarknoten gesammelt werden [Wu+19]. Dafür wird auch die räumliche Beziehung zwischen den Knoten betrachtet. Diese können beispielsweise durch Polarkoordinaten oder durch kartesische Koordinaten dargestellt werden. In dieser Arbeit verwenden wir Spline-based Convolutional Neural Networks (Spline-CNNs) [Fey+17]. Die Spline-CNNs sind ein Modell aus dem räumlichen Bereich der Graph-CNNs. Sie verwenden einen neuartigen Faltungsoperator, welcher auf B-Splines basiert.

Alle räumlichen Methoden beziehen ihre räumlichen Informationen aus der Konnektivität der Knoten, den Kantengewichten und den Knotenmerkmalen der geometrischen Eingabedaten. Diese Eigenschaft gilt nicht für eingebettete Graphen, welche zusätzliche räumliche Informationen durch ihre relativen Knotenpositionen beinhalten. Diese zusätzlichen Informationen können die Spline-CNNs erkennen und für ihre Berechnung verwenden.
Neben den Graph-CNNs gibt es noch drei weitere Klassen von Graph Neural Networks (GNNs) [Wu+19]: Graph Autoencoder, Recurrent Graph Neural Networks und Spatial-temporal Graph Neural Networks. Innerhalb dieser Arbeit werden wir ein Modell entwickeln, welches sowohl ein Graph-CNN basiertes Modell als auch eine Framework verwendet, welches den Spatial-Temporal Graph Neural Networks zuzuordnen ist.

2.5 B-Spline Basisfunktionen

Basis-Spline Kurven, auch B-Spline Kurven genannt, werden durch B-Spline Basisfunktionen beschrieben, welche stückweise aus Polynomen zusammengesetzt sind. Die Grafik 2.13 zeigt eine solche B-Spline Kurve. Die Grafik wurde mithilfe der Python-Bibliothek geomdl[1] erzeugt [BK19]. Mittels solcher Funktionen können komplexe Formen approximiert werden. Durch die von deBoor, Cox und Mansfield entwickelte Rekursionsformel für die Basisfunktionen, können diese effizient berechnet werden [PT96]. Sei $U = \{u_0, ...u_m\}$ eine Menge von nicht-absteigenden reellen Zahlen mit $u_i \leq u_{i+1}, i = 0, ..., m-1$. U wird als Knotenvektor bezeichnet und seine einzelnen Elemente u_i sind Knoten. Die i-te B-Spline Basisfunktion vom Grad p kann als $N_{i,p}(u)$ wie in den Formeln 2.32 und 2.33 definiert werden. Die Formel in 2.32 beschreibt den Basisfall, falls der Grad p Null ist.

$$N_{i,0} = \begin{cases} 1 & \text{wenn } u_i \leq u < u_{i+1} \\ 0 & \text{sonst} \end{cases} \tag{2.32}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.33}$$

Die Funktion $N_{i,0}(u)$ ist ausschließlich auf dem halboffenen Intervall $u \in [u_i, u_{i+1})$ nicht Null. Außerdem gilt für $p > 0$, dass $N_{i,p}$ eine Linearkombination zweier Basisfunktionen mit dem Grad $p-1$ ist. Die Grafik 2.14 zeigt, wie eine B-Spline Basisfunktion mit Grad p zusammensetzt ist. Es werden zur Berechnung einer Basisfunktion mit dem Grad p zwei Basisfunktionen mit dem Grad $p-1$ benötigt. In der Abbildung 2.14 wird beispielsweise die Basisfunktion $N_{0,1}$, welche den Grad $p = 1$ besitzt, aus den Basisfunktionen $N_{0,0}$ und $N_{1,0}$ zusammengesetzt. Beide Basisfunktionen besitzen den Grad 0. Das halboffene Intervall $[u_0, u_{i+1})$ wird als die i-te Knotenspanne bezeichnet. Für den Fall, dass die Knoten nicht verschieden voneinander sind, hat die Knotenspanne die Länge 0.
B-Spline Basisfunktionen besitzen einige vorteilhafte Eigenschaften. Eine davon ist die lokale Stützeigenschaft. Der Funktionswert $N_{i,p}(u) = 0$ falls u nicht in dem halboffenen Intervall $[u_i, u_{i+p+1})$ liegt. Das liegt mitunter an dem Aufbau der B-Spline Basisfunktionen. Als Beispiel dafür betrachten wir die Basisfunktion $N_{1,3}$. Diese benötigt zur Berechnung die Basisfunktionen $N_{1,0}$, $N_{2,0}$, $N_{3,0}$ und $N_{4,0}$ (siehe Abbildung 2.14). Eine weitere Eigenschaft der Basisfunktionen ist, dass sie nicht negativ sind. Die Basisfunktion $N_{i,p}(u)$ ist für alle i und p größer Null. Dies kann mittels Induktion über den Grad p nachgewiesen werden. Die Ableitung einer B-Spline Basisfunktion $N_{i,p}$ ist durch die Gleichung 2.34 definiert. Auch die Definition der Ableitung kann mittels Induktion nachgewiesen werden.

[1] `https://pypi.org/project/geomdl/`

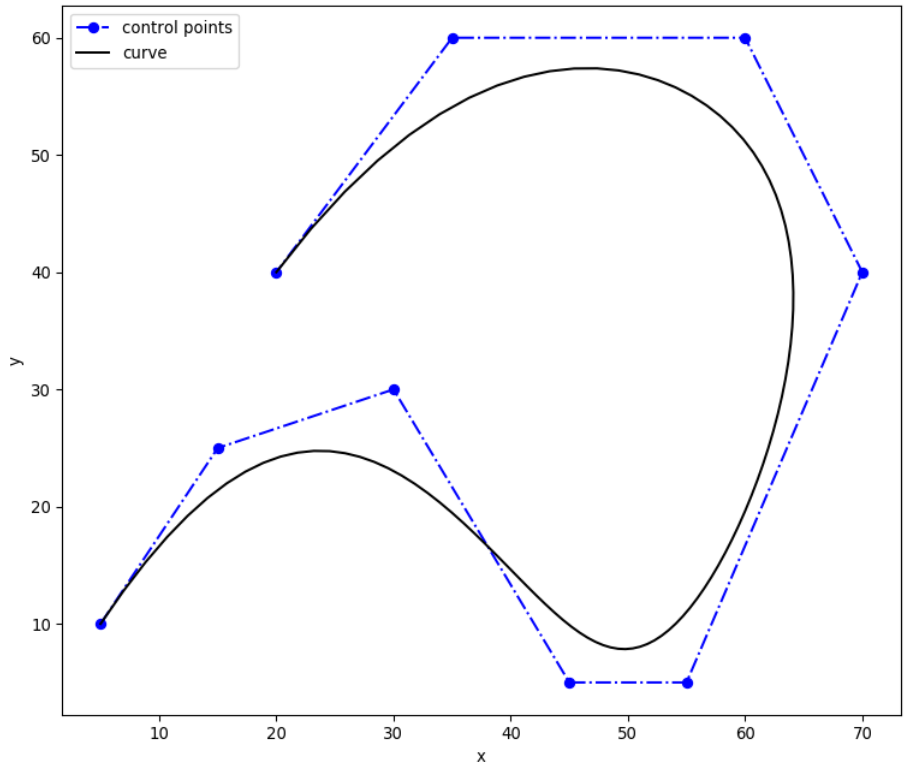


Abbildung 2.13: Beispiel einer B-Spline Kurve mit 9 Kontrollpunkten

$$N'_{i,p} = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.34}$$

Auf Basis der Formel 2.34 kann die k-te Ableitung einer Basisfunktion $N_{i,p}$ wie folgt dargestellt werden:

$$N^k_{i,p}(u) = p \left(\frac{N^{(k-1)}_{i,p-1}}{u_{i+p} - u_i} - \frac{N^{(k-1)}_{i+1,p-1}}{u_{i+p+1} - u_{i+1}} \right) \tag{2.35}$$

Die Abbildungen 2.15 und 2.16 zeigen eine B-Spline Kurve mit 7 Kontrollpunkten und die zugehörigen Basisfunktionen mit einem Grad $p = 3$. Die dargestellte Kurve ist stückweise aus den zugehörigen Basisfunktionen zusammengesetzt.

2.6 B-Spline Kurven

Eine B-Spline Kurve vom Grad p ist definiert durch [PT96]:

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \tag{2.36}$$

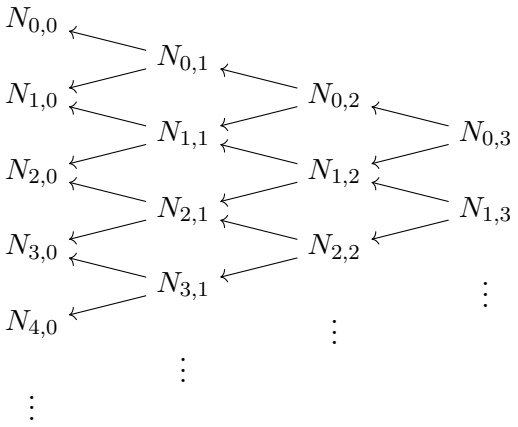


Abbildung 2.14: Rekursiver Aufbau einer B-Spline Basisfunktion. Für eine Basisfunktion vom Grad p werden immer zwei Basisfunktionen mit Grad $p-1$ benötigt.

Dabei gilt $a \leq u \leq b$. Die Menge der P_i beschreibt die Kontrollpunkte und $N_{i,p}(u)$ die B-Spline Basisfunktionen mit Grad p. Die Kontrollpunkte und die Basisfunktionen sind auf dem nicht-periodischen Knotenvektor $U = \{a, ..., a, u_{p+1}, ..., u_{m-p-1}, b, ..., b\}$ definiert. Der Knotenvektor U enthält $m+1$ Knoten. Wenn nicht anders beschrieben, ist $a = 0$ und $b = 1$. Die Menge der Kontrollpunkte wird auch als Kontrollpolygon bezeichnet.
Für die Berechnung einer B-Spline Kurve sind drei Schritte notwendig [PT96]. Zunächst muss die Knotenspanne gefunden werden, welche u enthält. Anschließend werden die Basisfunktionen berechnet, welche nicht Null sind. Als letzter Schritt werden die Werte der Basisfunktionen mit den zugehörigen Kontrollpunkten multipliziert.
Aus den Eigenschaften der B-Spline Basisfunktionen aus Kapitel 2.5 resultieren ebenfalls einige nützliche Eigenschaften der B-Spline Kurven. Da die Basisfunktionen stückweise aus Polynomen zusammengesetzt sind, gilt dies auch für die B-Spline Kurven. Der Grad p, die Anzahl der Knoten m und die Anzahl der Kontrollpunkte resultieren aus der Gleichung:

$$m = n + p + 1 \tag{2.37}$$

Der Anfang $\mathbf{C}(0)$ und das Ende $\mathbf{C}(1)$ der B-Spline Kurve entsprechen den Kontrollpunkten P_0 und P_n. Wird der Kontrollpunkt P_i verschoben, wird die Form der Kurve $\mathbf{C}(u)$ ausschließlich in dem halboffenen Intervall $[u_i, u_{i+p+1})$ verändert. Das liegt daran, da die zugehörige Basisfunktion mit dem Kontrollpunkt multipliziert wird und die Basisfunktion für alle Werte außerhalb dieses Intervalls 0 ist. Das Kontrollpolygon einer B-Spline Kurve ist eine Annäherung an die tatsächliche Kurve. Diese Approximation kann durch Hinzufügen neuer Knoten immer weiter verbessert werden. Je geringer also der Grad einer B-Spline Kurve ist, desto näher liegt die Approximation des Kontrollpolygons an der tatsächlichen Kurve.
Die Ableitung einer B-Spline Kurve ist analog zu den Ableitungen der Basisfunktionen definiert (siehe Gleichung 2.38).

$$\mathbf{C}^{(k)}(u) = \sum_{i=0}^{n} N_{i,p}^{k}(u) P_i \tag{2.38}$$

Für den Fall, dass u nicht verändert wird, kann die Ableitung $\mathbf{C}^{(k)}(u)$ mithilfe der Berechnungen der k-ten Ableitungen der Basisfunktionen berechnet werden.

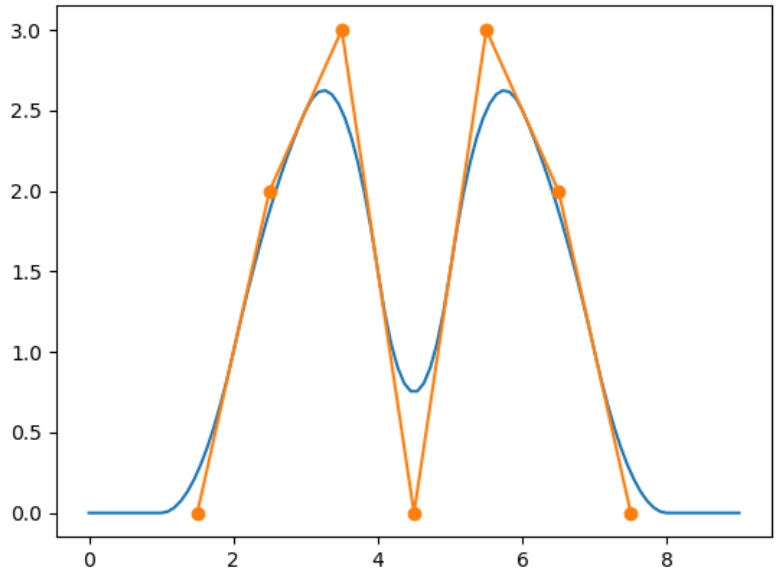


Abbildung 2.15: Die Abbildung zeigt eine B-Spline Kurve mit 7 Kontrollpunkten. Die Kontrollpunkte sind in Orange dargestellt und die B-Spline Kurve ist blau.

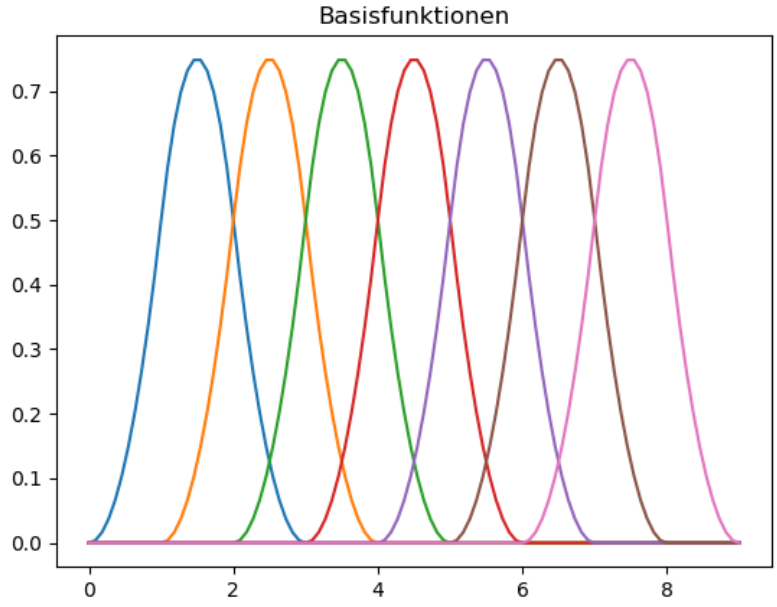


Abbildung 2.16: Die Abbildung zeigt die zu Abbildung 2.15 gehörenden B-Spline Basisfunktionen mit einem Grad $p = 3$.

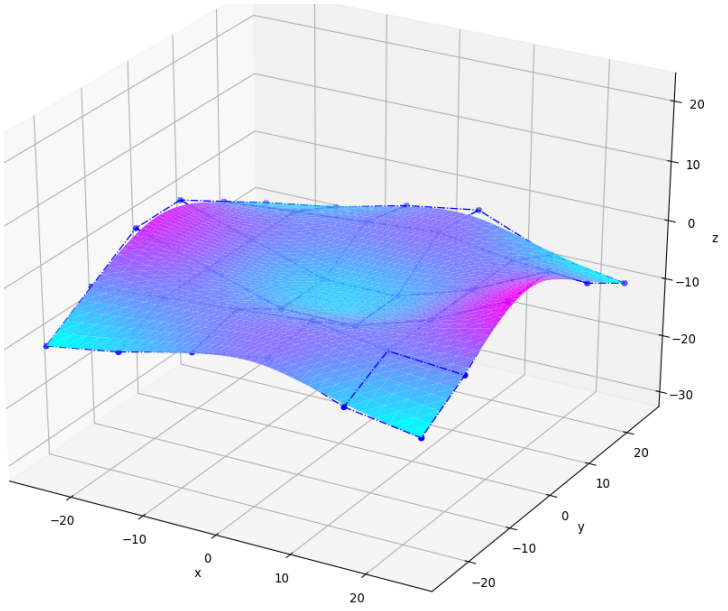


Abbildung 2.17: Die Abbildung zeigt eine B-Spline Oberfläche mit ihren zugehörigen Kontrollpunkten. Die blauen Punkte stellen die Kontrollpunkte dar.

2.7 B-Spline Oberflächen

Eine B-Spline Oberfläche ist durch ein bidirektionales Netz von Kontrollpunkten, zwei Knotenvektoren und das Produkt der B-Spline Basisfunktionen (siehe Formel 2.39) definiert [PT96].

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P_{i,j} \tag{2.39}$$

Die beiden Knotenvektoren U und V besitzen $r+1$ bzw. $s+1$ Knoten. Für r und s gilt:

$$r = n + p + 1 \qquad s = m + q + 1 \tag{2.40}$$

Dabei entsprechen p und q dem Grad der entsprechenden Basisfunktion und $n+1$ und $m+1$ sind die Anzahl der jeweiligen Kontrollpunkte. Um einen Punkt auf einer B-Spline Oberfläche für (u,v) zu berechnen sind fünf Schritte notwendig [PT96]. Zuerst muss eine Knotenspanne gefunden werden für die gilt $u \in [u_i, u_{i+1})$. Anschließend wird die Menge der Basisfunktionen $\{N_{i-p,p}(u), \ldots, N_{i,p}(u)\}$, welche nicht Null sind, berechnet. Darauffolgend wird eine Knotenspanne gesucht, welche v enthält und es wird analog die Menge der Basisfunktionen berechnet, welche nicht Null sind. Zuletzt werden die Werte der gewählten Basisfunktionen mit ihren zugehörigen Kontrollpunkten multipliziert. Die Abbildung 2.17 zeigt eine B-Spline Oberfläche mit ihren zugehörigen Kontrollpunkten. Für B-Spline Oberflächen gelten die gleichen Eigenschaften wie für die Basisfunktionen. Für uns sind die Nichtnegativität (siehe 2.41) und die Teilung der Einheit (siehe 2.42) wichtig.

$$N_{i,p}(u) N_{j,q}(v) \geq 0, \ \forall i,j,p,q,u,v \tag{2.41}$$

$$\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) = 1, \ \forall (u,v) \in [0,1] \times [0,1] \tag{2.42}$$

3 Spline-Spatio-Temporal Graph Convolutional Neural Networks

In diesem Kapitel stellen wir das im Zuge dieser Arbeit entwickelte Modell der Spline-Spatio-Temporal Graph Convolutional Neural Networks (Spline-STGCNs) vor. Sie verwenden Spatio-Temporal Graph Convolutional Neural Networks (siehe Abschnitt 3.2), welche für die Vorhersage von Verkehrsdaten geeignet sind und eine besondere Art von Convolutional Neural Networks, die Spline-CNNs, welche sehr gut für den Umgang mit räumlich strukturierten Eingabedaten geeignet sind (siehe Abschnitt 3.1).
Im Abschnitt 3.3 wird der Aufbau und die Funktionsweise der Spline-STGCNs erläutert und welche Eigenschaften wir uns von dem entwickelten Modell erhoffen.

3.1 Spline-based Convolutional Neural Networks

Die Spline-CNNs wurden von Fey et al. [Fey+17] entwickelt. Spline-CNNs sind für die Arbeit auf geometrischen Eingaben geeignet. Sie werden verwendet, um CNN-Methoden auf Graphen und Mannigfaltigkeiten anzuwenden. Die Spline-CNNs gehören zur Klasse der Graph Convolutional Neural Networks. Das Besondere an den Spline-CNNs ist der neuartige Filteroperator. Dieser verwendet B-Splines, was dazu führt, dass die Berechnungszeit unabhängig von der verwendeten Filtergröße ist. Der Filter ist ein trainierbarer, räumlicher und kontinuierlicher Faltungsoperator. Aufgrund der Eigenschaften der B-Spline Basisfunktionen kann der Filter Eingaben mit beliebiger Dimension verarbeiten.
Wir verwenden die Spline-CNNs als Faltungsschicht in unserem Modell, da sie in den Experimenten von Fey et al. [Fey+17] hervorragende Ergebnisse bezüglich verschiedenen geometrischen Problemstellungen gezeigt haben. Die Spline-CNNs wurden für zwei verschiedene Graph-Klassifizierungsprobleme getestet. Aufgrund ihres neuartigen Faltungsoperators und ihre Eignung für den Umgang mit räumlichen Eingabedaten, ist es naheliegend sie in die Bildung eines neuen Modells für die Kurzzeitprognose von räumlich verteilten Zeitreihen mit einzubeziehen.
Wir erhoffen uns durch die Verwendung der Spline-CNNs in unserem Modell, die Fähigkeit zusätzliche Informationen für die zeitlichen Merkmale aus der räumlichen Struktur des Sensornetzwerkes zu erfassen. Diese zusätzlichen räumlichen Informationen sollen unserem Modell helfen, indirekte Abhängigkeiten in den Eingabedaten zu finden und somit die Wahrscheinlichkeitsverteilung der zugrundeliegenden Daten besser zu lernen.

3.1.1 Notation

Die Eingabe für den Faltungsoperator ist ein gerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{U})$. Die Variable $\mathcal{V} = \{1, ..., N\}$ beschreibt die Menge der Knoten des Graphen. Die Menge der Kanten wird durch $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ dargestellt. Besonders wichtig ist die Matrix $\mathbf{U} \in [0, 1]^{N \times N \times d}$. Sie enthält die d-dimensionalen Pseudokoordinaten $\mathbf{u}(i, j) \in [0, 1]^d$ für jede gerichtete Kante $(i, j) \in \mathcal{E}$. Diese Pseudokoordinaten werden benötigt, um zusätzliche Informationen aus den relativen Positionen der Knoten zu erhalten.

In diesem Modell kann die Matrix $\mathbf{U}$ als Adjazenzmatrix des Graphen betrachtet werden. Der Unterschied zu einer klassischen Adjazenzmatrix, welche die Information beinhaltet, ob eine Kante zwischen zwei Knoten in einem Graphen vorhanden ist oder nicht, ist, die Dimension der Einträge. Die Einträge $\mathbf{u}$ sind d-dimensional und werden normalisiert. Das bedeutet $\mathbf{u}(i,j) \in [0,1]^d$ wenn die Kante (i,j) existiert und ansonsten 0. Die Menge der Nachbarknoten eines Knotens $i \in \mathcal{V}$ wird durch $\mathcal{N}(i)$ dargestellt. Jeder Knoten des Graphen enthält Werte. Diese werden als Knotenmerkmal bezeichnet. Dies können beispielsweise Farbwerte eines Pixels oder Messwerte eines Sensors sein. Der Vektor $\mathbf{f} : \mathcal{V} \to \mathbb{R}^{M_{\text{in}}}$ bezeichnet die Eingabemerkmale für jeden Knoten $i \in \mathcal{V}$. Für diesen Vektor gilt $\mathbf{f}(i) \in \mathbb{R}^{M_{\text{in}}}$. Die Eingabemerkmalskarte ist die Menge $\{\mathbf{f}_l(i) | i \in \mathcal{V}\}$ mit $1 \leq l \leq M_{in}$. Die Variable M_{in} bezeichnet die Menge der Eingabemerkmale. Die räumlichen Beziehungen, der in dem Vektor gespeicherten Knotenmerkmale der geometrischen Eingabe, sind durch die Pseudokoordinaten in $\mathbf{U}$ definiert. Das bedeutet, dass der Vektor $\mathbf{f}(i)$ den Inhalt der Knoten darstellt, also welche Art von Informationen erfasst werden und die Einträge $\mathbf{u}(i,j)$ stellen dar, wie die Informationen miteinander verbunden sind. Für Graphen kann $\mathbf{U}$ beispielsweise Kantengewichte oder Koordinaten (kartesisch, polar oder sphärisch) beinhalten. Im Allgemeinen ist die Art der Informationen in $\mathbf{U}$ nicht beschränkt, solange sie Auskunft über die relative Position der Knoten beinhaltet und die Werte in einem festgelegten Intervall liegen. Jeder Eintrag $\mathbf{u}(i,j)$ wird auf einen skalaren Wert abgebildet. Dieser wird als Gewicht für die Extraktion der Merkmale verwendet. Für den Faltungsoperator werden B-Spline Basisfunktionen benötigt. Mit $\left(\left(N_{1,i}^m\right)_{1\leq i\leq k_1}, \ldots, \left(N_{d,i}^m\right)_{1\leq i\leq k_d}\right)$ werden d B-Spline Basisfunktionen mit dem Grad m bezeichnet. Diese basieren auf äquidistanten Knotenvektoren. Der Knotenvektor $\mathbf{k} = (k_1, \ldots, k_d)$ beschreibt die Größe des d-dimensionalen Filters. Dieser Vektor ist leicht abweichend von der Definition des Knotenvektors im Abschnitt 2.5 definiert. Das liegt daran, dass $\mathbf{U}$ in dem Fall der Spline-CNNs die Adjazenzmatrix mit den normalisierten Kantengewichten beschreibt.

3.1.2 Faltungsoperator

Zuerst wird eine kontinuierliche Filterfunktion angewendet. Diese verwendet B-Spline Basisfunktionen, welche eine feste Anzahl an Kontrollwerten besitzen. Für die Filterfunktion wird die lokale Stützeigenschaft von B-Spline Basisfunktionen [PT96] ausgenutzt. Diese Stützeigenschaft besagt, dass Basisfunktionen nur auf einer begrenzten Menge von Unterintervallen nicht 0 sind. Das bedeutet im Umkehrschluss, dass die Basisfunktionen ihre Eingabe für ein unbekanntes Intervall auf 0 abbilden. Diese Eigenschaft ist für eine effiziente Berechnung und für die Skalierbarkeit des Modells wichtig.
Die Menge $\left(\left(N_{1,i}^m\right)_{1\leq i\leq k_1}, \ldots, \left(N_{d,i}^m\right)_{1\leq i\leq k_d}\right)$ beschreibt d offene B-Spline-Basen vom Grad m. Das kartesische Produkt der B-Spline Basisfunktionen wird durch $\mathcal{P} = \left(N_{1,i}^m\right)_i \times \cdots \times \left(N_{d,i}^m\right)_i$ und seine Elemente mit $\mathbf{p} \in \mathcal{P}$ dargestellt. Die trainierbaren Parameter des Filters sind $w_{\mathbf{p},l} \in \mathbf{W}$ für jedes $\mathbf{p}$ und jedes l. Die Variable l indiziert die M_{in} Eingabemerkmale. Aus dieser Definition von $\mathbf{W}$ folgt, dass insgesamt $K = M_{\text{in}} \cdot \prod_{i=1}^{d} k_i$ trainierbare Parameter existieren.
Die kontinuierlichen Filter werden durch die Funktionen $g_l : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$ in Gleichung 3.1 dargestellt.

$$g_l(\mathbf{u}) = \sum_{\mathbf{p}\in\mathcal{P}} w_{\mathbf{p},l} \cdot B_{\mathbf{p}}(\mathbf{u}) \tag{3.1}$$

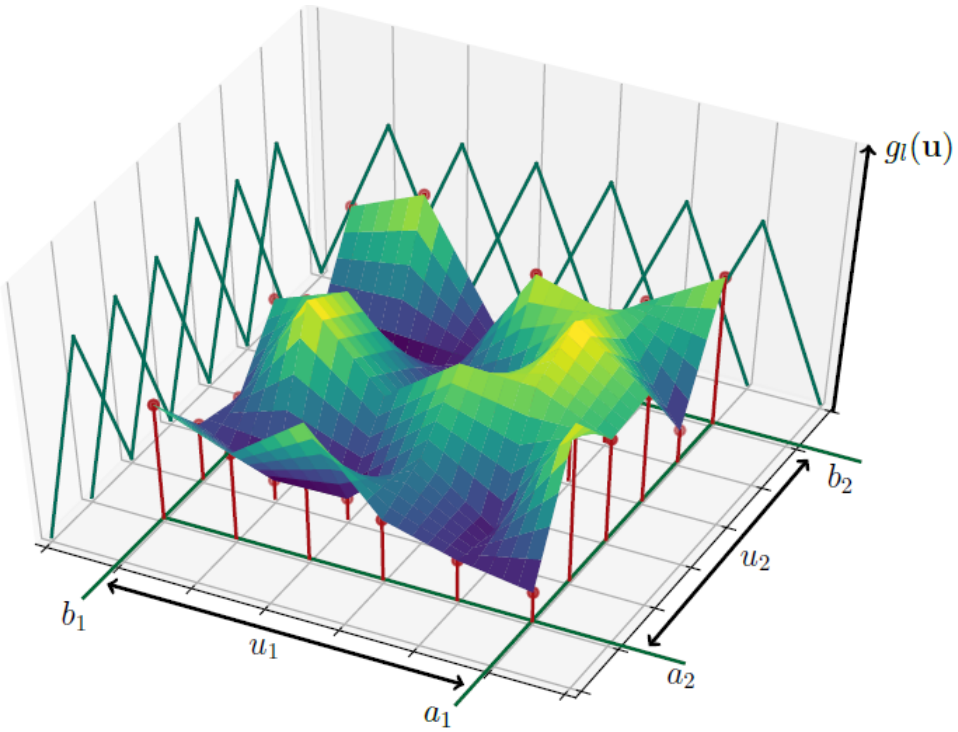


Abbildung 3.1: Beispiel eines B-Spline Filters mit dem Grad $m = 1$ und der Dimensionalität $d = 2$. Die Höhe der roten Punkte können als Kontrollpunkte der Oberfläche interpretiert werden. Sie beschreibt die trainierbaren Parameter für eine einzelne Eingabemerkmalskarte [Fey+17].

Dabei beschreibt $B_{\mathbf{p}}$ das Produkt der Basisfunktionen in $\mathbf{p}$ und ist wie folgt definiert:

$$B_{\mathbf{p}}(\mathbf{u}) = \prod_{i=1}^{d} N_{i,p_i}^{m}(u_i) \tag{3.2}$$

Die definierten Gewichte $w_{\mathbf{p},l}$ können dabei als Steuerungswerte betrachtet werden. Diese steuern die Höhe einer $(d+1)$-dimensionalen B-Spline Oberfläche. Damit kann abhängig von $\mathbf{u}(i,j)$ ein Gewicht für jeden Nachbarpunkt j bestimmt werden. Die Grafik 3.1 zeigt ein Beispiel für einen Filter mit B-Spline Basisfunktionen mit dem Grad $m = 1$ und der Dimensionalität $d = 2$ [Fey+17]. Im Vergleich zu anderen B-Spline Methoden werden in den Spline-CNNs ausschließlich eindimensionale Kontrollpunkte verwendet. Zusätzlich werden die Filterfunktionen $g_l : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$ anstelle von Kurven approximiert. Der Definitionsbereich von g_l ist ein Intervall, in welchem die Eigenschaft der Teilung der Einheit (engl. *partition of unity*) der B-Spline Basisfunktionen gilt [PT96]. Die Teilung der Einheit ist eine Eigenschaft, welche besagt, dass für einen beliebigen Knotenvektor $[k_i, k_{i+1})$ mit $1 \leq i \leq d$ $\sum_{j=i-m}^{i} N_j^m(k) = 1$ für alle $k \in [k_i, k_{i+1})$ gilt [PT96].
Die Variablen a_i und b_i sind abhängig vom Grad m und der Filtergröße. Die Einträge $\mathbf{u}(i,j) \in \mathbf{U}$ werden so skaliert, dass sie in genau dem benötigten Intervall liegen. Nun, da die Filterfunktionen $\mathbf{g} = (g_1, \ldots, g_{M_{\text{in}}})$ und die Eingabeknotenmerkmale $\mathbf{f}$ erklärt sind, kann der Faltungsoperator $\star$ für einen Knoten i wie folgt definiert werden:

$$(\mathbf{f} \star \mathbf{g})(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{l=1}^{M_{in}} \sum_{j \in \mathcal{N}(i)} f_l(j) \cdot g_l(\mathbf{u}(i,j)) \tag{3.3}$$

Der Faltungsoperator wird M_{out} Mal mit verschiedenen trainierbaren Parametern auf dieselben Eingabedaten angewendet, um M_{out} Ausgabemerkmalskarten zu erhalten. Für jede Kombination von Eingabe- und Ausgabemerkmalskarten wird eine eigene Menge von Gewichten trainiert. Die Ausgabe von $g_l(\mathbf{u})$ ist ausschließlich von $M_{in} \cdot s$ der insgesamt $M_{in} \cdot K$ trainierbaren Parametern abhängig. Das liegt an der lokalen Stützeigenschaft der B-Splines, dass $B_{\mathbf{p}} \neq 0$ nur für $s := (m+1)^d$ der K verschiedenen Vektoren $\mathbf{p} \in \mathcal{P}$ gilt. Zusätzlich sind die Variablen s, d, und m in der Regel sehr klein. $\mathcal{P}(\mathbf{u}(i,j))$ beschreibt alle Vektoren $\mathbf{p}$ mit $B_{\mathrm{p}} \neq 0$ für jedes Knotenpaar (i,j). Diese Vektoren können in konstanter Zeit berechnet werden, wenn m und d gegeben sind. Anhand dieser Annahmen kann die innere Summe des Faltungsoperators aus Gleichung 3.3 auch dargestellt werden als:

$$(f_l \star g_l)(i) = \sum_{\substack{j \in \mathcal{N}(i) \\ \mathbf{p} \in \mathcal{P}(\mathbf{u}(i,j))}} f_l(j) \cdot w_{\mathbf{p},l} \cdot B_{\mathbf{p}}(\mathbf{u}(i,j)) \tag{3.4}$$

Da $B_{\mathbf{p}}(\mathbf{u}(i,j))$ unabhängig von l ist, muss $B_{\mathbf{p}}(\mathbf{u}(i,j))$ nur einmal für alle Eingabemerkmale berechnet werden.
Bisher wurde ausschließlich die Menge der Nachbarknoten $\mathcal{N}(i)$ eines Knotens i im Faltungsoperator betrachtet, jedoch nicht der Knoten i selbst. Das ist ein Unterschied zu klassischen CNNs, welche den betrachteten Knoten selbst und seine Nachbarn im Faltungsoperator berücksichtigen. Wenn kartesische Koordinaten verwendet werden, kann $\mathcal{N}(i)$ so definiert werden, dass der Knoten i mit enthalten ist. In diesem Fall ist jeder Knoten i ein Nachbar von sich selbst.
In dieser Arbeit werden ausschließlich Graphen betrachtet, welche Sensornetzwerke repräsentieren. Das bedeutet, jeder Sensor stellt einen Knoten in dem Graphen dar. In einer, dem Graphen zugehörigen, Adjazenzmatrix hätte jeder Knoten auch eine Selbstkante, falls diese Eigenschaft gewünscht ist. Die Graphen, welche wir im Zuge unserer Experimente betrachten, besitzen keine Selbstkanten. Der B-Spline basierten Faltungsoperator ist somit eine Generalisierung des klassischen Faltungsoperators mit ungerader Filtergröße für Convolutional Neural Networks.

3.2 Spatio-Temporal Graph Convolutional Neural Networks

Spatio-Temporal Graph Convolutional Neural Networks (STGCNs) sind ein Deep Learning Framework für die Vorhersage von Zeitreihen [YYZ17]. Dieses Framework gehört zur Klasse der Spatial-Temporal Graph Neural Networks. Das Ziel dieser Art von Graph Neural Networks ist es, versteckte Muster aus den Knotenmerkmalen des Graphen, welche räumliche und zeitliche Daten beinhalten, zu lernen.
Wir haben uns aus mehreren Gründen für eine Verwendung des STGCN-Frameworks als Teil unseres Modell für die Kurzzeitprognose räumlich verteilter Zeitreihen entschieden. Die STGCNs sind zum einen ein sehr neues und bisher nicht viel getestetes Modell, welches bei den bisherigen Experimenten jedoch gute Ergebnisse im Vergleich zu anderen Modellen gezeigt hat. Zum anderen bietet es durch seinen strukturellen Aufbau die Möglichkeit, die Spline-CNNs für die zeitliche Faltungsschicht zu verwenden. Zusätzlich ist das Framework nicht nur in der Lage Verkehrsdaten zu modellieren, sondern es kann in Zukunft auch auf anderen räumlich-zeitlichen Problemstellungen angewendet werden.

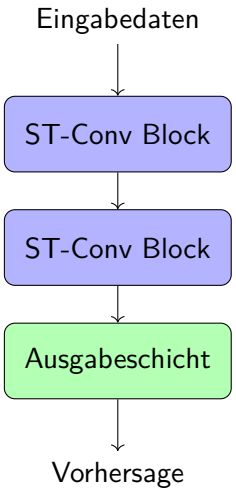


Abbildung 3.2: Die Abbildung zeigt den äußeren Aufbau der STGCNs. Die Eingabedaten dienen als Eingabe in einen ST-Conv Block. Die Ausgabe dieses Faltungsblocks dient als Eingabe eines weiteren ST-Conv Blocks. Anschließend wird diese Ausgabe von einer Ausgabeschicht verarbeitet. Diese enthält einen einzelnen zeitlichen Faltungsblock und eine vollständig vernetzte Schicht.

3.2.1 Aufbau

Ein STGCN besteht aus zwei räumlich-zeitlichen Faltungseinheiten und einer vollständig vernetzten zeitlichen Ausgabeschicht. Die räumlich-zeitlichen Faltungseinheiten werden als ST-Conv Block bezeichnet. Dieser Aufbau wird in 3.2 dargestellt. Die Abbildung 3.3 zeigt die Darstellung der Sandwich-Struktur der zeitlichen und räumlichen Faltungsschichten. Der Aufbau der räumlich-zeitlichen Blöcke folgt dieser Sandwich-Struktur. Einer räumlichen Faltungsschicht geht eine zeitliche Faltungsschicht voraus und ihr folgt eine solche zeitliche Faltungsschicht. Es wird zunächst betrachtet, wie die räumlichen Merkmale aus der Eingabe extrahiert werden.
Ein Verkehrsnetzwerk besitzt grundlegend den Aufbau eines Graphen. Deshalb ist es naheliegend, ein Verkehrsnetzwerk auch mathematisch als Graphen zu betrachten. Viele andere Methoden teilen diesen Graphen in Segmente oder ordnen ihn in einer gleichmäßigen Gitterstruktur an, um ihn zu verarbeiten. Durch das Aufteilen oder die Umstrukturierung gehen Informationen über das Netzwerk verloren. Es werden Kanten entfernt oder Abstände von Knoten verändert. Dadurch geht zum einen teilweise verloren, wie die einzelnen Knoten miteinander verbunden sind und zum anderen werden nicht offensichtliche Informationen entfernt. Um dies zu vermeiden und diese Informationen zu erhalten, werden in den STGCNs Graph-CNNs für die Faltung eingesetzt.
Dadurch kann die Faltung direkt auf der als Graph strukturierten Eingabe erfolgen und das Modell kann aus den zusätzlichen Informationen Muster und Merkmale besser erkennen. Innerhalb dieses Modells wird ein Faltungsoperator $*_{\mathcal{G}}$ auf Basis der Fouriertransformation verwendet. Dieser basiert auf dem Konzept der räumlichen Faltungsoperatoren. Allgemein wird ein Filter Θ mit einem Signal $\mathbf{x} \in \mathbb{R}^n$ multipliziert (siehe Gleichung 3.5).

$$\Theta *_{\mathcal{G}} \mathbf{x} = \Theta(L)\mathbf{x} = \Theta\left(U\Lambda U^T\right)\mathbf{x} = U\Theta(\Lambda)U^T\mathbf{x} \tag{3.5}$$

Die Fourierbasis des Graphen wird als $U \in \mathbb{R}^{n \times n}$ bezeichnet. Sie bildet die Matrix der Eigenvektoren der normalisierten Laplace-Matrix L.

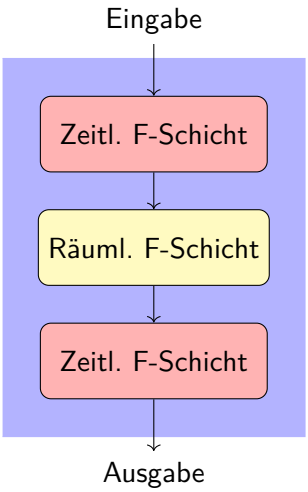


Abbildung 3.3: Diese Abbildung zeigt die innere Sandwich-Struktur eines einzelnen ST-Conv Blocks. Auf eine zeitliche Faltungsschicht folgt eine räumliche Faltungsschicht und abschließend wieder eine zeitliche Faltungsschicht.

Für die Laplace-Matrix gilt: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = U \Lambda U^T \in \mathbb{R}^{n \times n}$. Die Matrix I ist die Identitätsmatrix und $D \in \mathbb{R}^{n \times n}$ ist die Diagonalmatrix mit den entsprechenden Knotengraden und $D_{ii} = \Sigma_j W_{ij}$. Die Diagonalmatrix, welche die Eigenwerte der Laplace-Matrix beinhaltet, wird als $\Lambda \in \mathbb{R}^{n \times n}$ dargestellt. Der Filter Θ über Λ ist ebenfalls eine Diagonalmatrix.
Anhand dieser Definitionen ist die Faltung des Signals $\mathbf{x}$ mittels eines Filters Θ eine Multiplikation zwischen dem Filter Θ und einer Fouriertransformation des Graphen $U^T \mathbf{x}$ [Shu+12]. Die Berechnung mittels der Gleichung 3.5 ist mit $\mathcal{O}\left(n^2\right)$ benötigten Multiplikation sehr aufwendig. Dank zweier Approximationsmethoden, kann die Berechnungskomplexität jedoch reduziert werden.
Die erste Annäherungsmethode ist die Approximation mittels Chebyshev-Polynomen. Bei diesem Ansatz können die Parameter des Filters Θ reduziert werden, indem dieser auf ein Polynom von Λ beschränkt wird. Das bedeutet: $\Theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$ mit $\theta \in \mathbb{R}^K$ als Vektor mit polynomiellen Koeffizienten. Die Kernelgröße wird durch K beschrieben. In diesem Fall beschreibt K den maximalen Radius ausgehend vom mittleren Knoten der aktuell betrachteten Faltung. Weitergehend wird das Chebyshev-Polynom $T_k(\mathbf{x})$ verwendet, um die Filter zu approximieren. Dabei wird der Filter Θ als gekürzte Erweiterung der Größe $K - 1$ dargestellt (siehe Gleichung 3.6)

$$\Theta(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \tag{3.6}$$

Die Diagonalmatrix mit den Eigenwerten der Laplace-Matrix wird dabei abgeändert zu $\tilde{\Lambda} = 2\Lambda / \lambda_{\max} - I_n$. Der größte Eigenwert aus L wird als λ_{max} bezeichnet. Mittels dieser Umformung kann nun die Faltung eines Signals $\mathbf{x}$ mit einem Filter Θ wie folgt formuliert werden:

$$\Theta *_{\mathcal{G}} \mathbf{x} = \Theta(L)\mathbf{x} \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})\mathbf{x} \tag{3.7}$$

Dabei wird das Chebyshev-Polynom der Ordnung k über der skalierten Laplace-Matrix $\tilde{L} = 2L/\lambda_{\max} - I_n$ ausgewertet. Werden nun K Faltungen mittels der Formel 3.7 durchgeführt, resultiert daraus ein Rechenaufwand von $\mathcal{O}(K|\mathcal{E}|)$ [DBV16].
Die zweite Annäherungsmethode ist die Approximation der ersten Ordnung. Diese geht auf Kipf und Welling [KW16] zurück. Bei dieser Methode werden mehrere Faltungsschichten mittels der Approximation der ersten Ordnung der Laplace-Matrix gestapelt. Dadurch kann eine deutlich tiefere Architektur erreicht werden. Zusätzlich können räumliche Merkmale besser extrahiert werden, da dieses Verfahren nicht durch die explizite Parametrisierung durch die Polynome beschränkt ist. Aufgrund der Skalierung und der Normalisierung, welche innerhalb von neuronalen Netzen angewendet werden, kann angenommen werden, dass für den größten Eigenwert der Laplace-Matrix $\lambda_{\max} \approx 2$ gilt. Mittels dieser Annahme kann die Gleichung 3.7 wie folgt vereinfacht werden:

$$\begin{aligned} \Theta *_{\mathcal{G}} \mathbf{x} &\approx \theta_0 \mathbf{x} + \theta_1 \left(\frac{2}{\lambda_{\max}} L - I_n \right) \mathbf{x} \\ &\approx \theta_0 \mathbf{x} - \theta_1 \left(D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \right) \mathbf{x} \end{aligned} \tag{3.8}$$

Bei den beiden Parametern θ_0, θ_1 handelt es sich um geteilte Parameter des Filters. Deshalb können θ_0, θ_1 durch einen einzelnen Parameter θ ersetzt werden, für den $\theta = \theta_0 = -\theta_1$ gilt.
Die Gewichtsmatrix W und die Diagonalmatrix D werden durch $\tilde{W} = W + I_n$ und $\tilde{D}_{ii} = \Sigma_j \tilde{W}_{ij}$ renormalisiert.
Durch diese Anpassungen kann die Faltung auf dem Graphen aus der Gleichung 3.8 vereinfacht werden zu:

$$\begin{aligned} \Theta *_{\mathcal{G}} \mathbf{x} &= \theta \left(I_n + D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \right) \mathbf{x} \\ &= \theta \left(\tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}} \right) \mathbf{x} \end{aligned} \tag{3.9}$$

Anhand dieser Approximationen für ein Signal $\mathbf{x}$ kann der Faltungsoperator $*_{\mathcal{G}}$ für mehrdimensionale Eingabedaten angepasst werden. Es wird nun ein Signal $\mathbf{X} \in \mathbb{R}^{n \times C_i}$ mit C_i Kanälen betrachtet. Daraus folgt eine Generalisierung des Faltungsoperators

$$y_j = \sum_{i=1}^{C_i} \Theta_{i,j}(L) \mathbf{x}_i \in \mathbb{R}^n, 1 \leq j \leq C_o \tag{3.10}$$

mit $C_i \times C_o$ Vektoren der Chebyshev-Koeffizienten $\Theta_{i,j} \in \mathbb{R}^K$. Die Größe von C_i und C_o kann auch als Größe der Eingabe- und Ausgabemerkmalskarten betrachtet werden. Daraus können wir die Faltung für einen Graphen mit zweidimensionalen Variablen als $\Theta *_{\mathcal{G}} \mathbf{X}$ mit $\Theta \in \mathbb{R}^{K \times C_i \times C_o}$ ableiten.
Um diese Beobachtung nun auf Verkehrsdaten zu übertragen, muss ersichtlich sein, in welcher Form die Daten vorliegen. Ein Verkehrsnetzwerk kann, wie bereits beschrieben, als Graph, bestehend aus n Knoten und m Kanten, dargestellt werden. Die Knoten stellen dabei die Messstationen dar, an denen Werte gemessen werden. Die Kanten mit ihren Gewichten repräsentieren, wie die Knoten untereinander verbunden sind. Die Eingabe für ein Modell, welches Verkehrsvorhersagen erzeugt, besteht aus M solcher Graphen. Jeder Graph beschreibt die Beobachtungen des gesamten Netzwerks zu einem bestimmten Zeitpunkt t. Jede dieser Einzelbeobachtungen v_t kann als Matrix betrachtet werden. Die i-te Spalte der Matrix entspricht dem C_i-dimensionalen Wert von v_t am i-ten Knoten im

Graphen $\mathcal{G}_t$. Damit kann die Matrix geschrieben werden als $\mathbf{X} \in \mathbb{R}^{n \times C_i}$. Bei einem betrachteten Zeitschritt als Eingabe gilt $C_i = 1$.
Für jeden Zeitschritt $t \in M$ wird die gleiche Faltungsoperation mit demselben Filter Θ auf $\mathbf{X}_t \in \mathbb{R}^{n \times C_i}$ parallel ausgeführt. Deshalb kann der Faltungsoperator auch auf dreidimensionalen Eingabedaten angewendet werden. Dann wird der Faltungsoperator bezeichnet durch $\Theta *_{\mathcal{G}} \mathcal{X}$ mit $\mathcal{X} \in \mathbb{R}^{M \times n \times C_i}$.
Als nächstes wird die Extraktion von zeitlichen Merkmalen betrachtet. In vielen Modellen mit Bezug zu zeitlichen Merkmalen werden Rekurrente Neuronale Netze (RNNs) verwendet. Diese sind jedoch komplex, das Finden der richtigen Hyperparameter erweist sich oftmals als schwierig und sie benötigen einen hohen Berechnungsaufwand. CNNs hingegen sind im Vergleich leichter trainierbar, die Trainingszeit ist oft geringer und ihre Struktur ist deutlich simpler. Auf Basis der von Gehring entwickelten Methode [Geh+17] werden in den STGCNs ausschließlich Faltungsstrukturen verwendet, welche die gesamte Zeitachse und nicht nur Teile davon betrachten, um zeitliche Merkmale und das dynamische Verhalten von Verkehrsfluss zu extrahieren. Die zeitliche Faltungsschicht enthält eine eindimensionale Faltung gefolgt von Gated Linear Units (GLUs) [Dau+16]. Die GLUs übernehmen die Aufgabe der Aktivierungsfunktion und verwenden intern eine elementweise Multiplikation mit der Sigmoidfunktion. Sie haben den Vorteil, dass sie das Problem des verschwindenden Gradienten reduzieren.
Das verschwindende Gradientenproblem tritt dann auf, wenn beim Optimierungsverfahren der Gradient gebildet wird und dieser einen Wert kleiner als 1 annimmt. Der Gradient wird, wie in Kapitel 2.1.3 beschrieben, mithilfe der Kettenregel aus dem Produkt der partiellen Ableitungen gebildet. Damit wird der Gradient im Laufe der Zeit immer kleiner.
Für jeden Knoten $v \in \mathcal{V}$ des Graphen $\mathcal{G}$ untersucht der Faltungsoperator K_t Nachbarn der Eingabeelemente. Somit wir die Länge der Sequenz jedes Mal um $K_t - 1$ gekürzt. Aufgrund dieses Mechanismus kann die Eingabe der zeitlichen Faltung für jeden Knoten als Sequenz der Länge M mit C_i Channels als $Y \in \mathbb{R}^{M \times C_i}$ betrachtet werden [YYZ17]. Der Filter $\Gamma \in \mathbb{R}^{K_t \times C_i \times 2C_o}$ ist so konstruiert, dass die Eingabe Y auf ein einzelnes Ausgabeelement $[PQ] \in \mathbb{R}^{(M-K_t+1) \times (2C_o)}$ abgebildet wird. P und Q werden in der Hälfte geteilt. Sie besitzen dieselbe Anzahl Channels. Daraus folgend kann die zeitliche Faltung mit den GLUs dargestellt werden als:

$$\Gamma *_{\mathcal{T}} Y = P \odot \sigma(Q) \in \mathbb{R}^{(M-K_t+1) \times C_o} \tag{3.11}$$

Dabei bezeichnen P und Q die Eingaben der GLUs und der Operator $\odot$ das elementweise Hadamard-Produkt. Das Hadamard-Produkt ist eine elementweise Multiplikation zweier Matrizen, welche dieselbe Größe besitzen. Die Sigmoidfunktion $\sigma(Q)$ aus den GLUs steuert, welche Eingabe P für die betrachteten Zustände wichtig sind, um Merkmale und dynamische Veränderungen in der Zeitreihe zu erkennen. Die zeitliche Faltung kann auch für dreidimensionale Variablen generalisiert werden. Dafür wird der gleiche Filter Γ auf jeden Knoten $\mathcal{Y}_i \in \mathbb{R}^{M \times C_i}$ (z.B. Messstationen) in $\mathcal{G}$ angewendet. Daraus resultiert für die dreidimensionale Faltung: $\Gamma *_{\mathcal{T}} \mathcal{Y}$ mit $\mathcal{Y} \in \mathbb{R}^{M \times n \times C_i}$.
Nach der Beschreibung der Faltungsoperationen in der räumlichen und der zeitlichen Faltungsschicht werden nun diese beiden Schichten verwendet, um einen räumlich-zeitlichen Faltungsblock zu bilden. Mit diesem Block ist es dem Modell möglich sowohl räumliche Merkmale als auch zeitliche Merkmale aus den Eingabedaten zu erkennen.
In der praktischen Anwendung können mehrere dieser Blöcke hintereinander verwendet werden. Die räumliche Schicht bildet die Brücke zwischen den zwei zeitlichen Schichten. Innerhalb jedes Faltungsblocks wird eine Normalisierung verwendet, um Overfitting zu

vermeiden. Die Eingabe und die Ausgabe eines Faltungsblocks sind jeweils dreidimensionale Tensoren. Für die Eingabe $v^l \in \mathbb{R}^{M \times n \times C^l}$ des l-ten Faltungsblocks wird die Ausgabe $v^{l+1} \in \mathbb{R}^{(M-2(K_t-1)) \times n \times C^{l+1}}$ wie folgt berechnet:

$$v^{l+1} = \Gamma_1^l *_{\mathcal{T}} \text{ReLU}\left(\Theta^l *_{\mathcal{G}} \left(\Gamma_0^l * {}_{\mathcal{T}} v^l\right)\right) \tag{3.12}$$

Dabei beschreiben Γ_0^l und Γ_1^l jeweils den Filter der oberen und der unteren zeitlichen Faltungsschicht innerhalb des l-ten Blocks. Der Filter der räumlichen Faltung des Graphen ist Θ^l. Die Funktion ReLU$(\cdot)$ beschreibt die Aktivierungsfunktion, welche auf die Ausgabe der räumlichen Faltungsschicht mit der Ausgabe der ersten zeitlichen Faltungsschicht angewendet wird.
In diesem Modell werden für die Extraktion von Merkmalen aus den Verkehrsdaten und für die Vorhersage von Verkehrsdaten zwei der beschriebenen ST-Conv Blöcke verwendet. Nach dem Durchlauf der Daten durch diese zwei Blöcke wird das Ergebnis als Eingabe für eine einzelne zeitliche Faltungsschicht verwendet. Dessen Ergebnis dient als Eingabe für eine vollständig vernetzte Schicht, welche letztlich die Ausgabeschicht des Modells darstellt. Diese Komposition der letzten beiden Schichten wird benötigt, um die Ausgabe des letzten Faltungsblocks für die Vorhersage eines einzelnen Zeitschritts zu verwenden. Als Ausgabe erzeugt das Modell $\mathbf{Z} \in \mathbb{R}^{n \times c}$. Daraus können wir die Geschwindigkeitsvorhersage für n Knoten des Graphen errechnen. Dafür wird eine lineare Transformation über c Kanälen als $\hat{v} = \mathbf{Z}\mathbf{w} + b$ angewendet. In diesem Fall ist $\mathbf{w} \in \mathbb{R}^c$ ein Vektor, welcher die einzelnen Gewichte enthält und b ist der Bias.
Innerhalb des STGCN-Modells wird der L2-Fehler verwendet, um die Performanz des Modells zu testen. Daraus resultiert folgende Verlustfunktion für die Vorhersage des Verkehrs innerhalb des STGCN-Modells:

$$L\left(\hat{v}; W_\theta\right) = \sum_i \left\| \hat{v}\left(v_{t-M+1}, \ldots, v_t, W_\theta\right) - v_{t+1} \right\|^2 \tag{3.13}$$

Die trainierbaren Parameter innerhalb des Modells sind W_θ. Die tatsächliche Vorhersage, also die Grundwahrheit, ist in der Formel 3.13 durch v_{t+1} beschrieben. Die Vorhersage, welche das Modell erzeugt, wird durch $\hat{v}(\cdot)$ dargestellt.

3.3 Spline-STGCNs

In dieser Arbeit beschäftigen wir uns mit den Spline-Spatio-Temporal Graph Convolutional Neural Networks (Spline-STGCNs). Wir wollen prüfen, inwiefern die Spline-STGCNs zur Kurzzeitprognose von Verkehrsdaten geeignet sind. Innerhalb unseres Modells ersetzen wir die in Abschnitt 3.2 verwendeten zeitlichen Faltungsschichten durch Spline-CNNs (siehe Abschnitt 3.1). Das STGCN-Framework ist sehr gut für den Umgang mit räumlich verteilten Zeitreihen geeignet. Durch den Aufbau und durch die Verwendung der räumlich-zeitlichen Faltungsblöcke beinhaltet das Framework sowohl die Möglichkeit der räumlichen als auch der zeitlichen Faltung.
Da die Spline-CNNs gute Ergebnisse für Klassifizierungsprobleme auf geometrischen Eingaben liefern (siehe [Fey+17]), liegt nahe zu testen, inwiefern sie sich in einem geeigneten Framework, für andere geometrische Problemstellungen eignen und ob sie überhaupt in der Lage sind andere geometrische Probleme zu lösen. Die Spline-CNNs können sehr gut zusätzliche räumliche Informationen aus der Konnektivität der Knoten zueinander erfassen und extrahieren.

Wie in Kapitel 1.1 beschrieben, ist die Prognose von Verkehrsdaten ein sehr interessantes, jedoch auch komplexes Thema. Es gibt viele unterschiedliche Lösungsansätze für dieses Problem. Wir erhoffen uns, mit den Spline-STGCNs ein neues Modell zu entwickeln, welches in der Lage ist sowohl die komplexen räumlichen Informationen eines Verkehrsnetzwerks zu nutzen als auch dynamische Merkmale aus der zeitlichen Komponente des Verkehrsflusses zu lernen und eine gute Genauigkeit bei den Vorhersagen zu erzielen. Wir wollen testen, inwiefern unser Modell als neuer Lösungsansatz für die Kurzzeitprognose räumlich verteilter Zeitreihen geeignet ist und es in Experimenten mit anderen State-of-the-art Modellen vergleichen.
Wir erhoffen uns ein Modell zu entwickeln, welches die Vorteile beider verwendeten Modelle vereint und somit besser in der Lage ist, räumliche Eingabedaten zu verarbeiten und aus der zugrundeliegenden Struktur der Eingabedaten zu lernen. Aufgrund der Experimentergebnisse der Spline-CNNs haben wir uns entschieden, sie für drei zeitliche Faltungsschichten innerhalb eines Spline-ST-Conv Blocks zu verwenden. Eine Spline-CNN Faltungsschicht bekommt ein Objekt mit den Knotenmerkmale, der Adjazenzmatrix in COO-Form und die Kantenattribute als Pseudokoordinaten übergeben. Darauf wird ein entsprechender Filter angewendet und eine Ausgabe erzeugt.

3.3.1 Aufbau

Abbildung 3.4 zeigt eine Übersicht über den internen Aufbau unseres Modells. Ein einzelner Spline-ST-Conv Block besteht aus einem zeitlichen Faltungsblock mit drei Schichten Spline-CNNs, gefolgt von einer räumlichen Faltung mithilfe der Kantengewichte des eingegebenen Graphen und anschließend folgt wiederum ein zeitlicher Faltungsblock mit drei Spline-CNN Schichten. Die durchgeführten Operationen in diesem Block werden in Abbildung 3.5 verdeutlicht. Die Ausgabe eines solchen zeitlichen Faltungsblocks wird in 3.5 jeweils durch die Ausgaben out_1 und out_3 dargestellt. Die Parameter unseres Modells können aus der Grafik 3.4 abgelesen werden. Die Eingabechannels C_{in} und die Ausgabechannels C_{out} beschreiben die Anzahl der Ein- und Ausgabemerkmale je Schicht. Die Menge der Eingabemerkmale ist abhängig vom gewählten Kontext und von der Anzahl der Knotenmerkmale. Werden für einen Knoten mehr als ein Wert je Zeitpunkt erfasst, wird diese Anzahl mit der Größe des Kontextes multipliziert. Die räumlichen Channels C_s beschreiben die Eingabemerkmale für die räumliche Faltungsschicht. Jeder Spline-ST-Conv Block bekommt zusätzlich noch die Anzahl der Knoten ($\#nodes$) des Graphen übergeben. Diese wird für diverse Umformungen innerhalb des Modells verwendet. Auf die zwei Spline-ST-Conv Blöcke folgt noch ein zeitlicher Faltungsblock und anschließend eine vollständig vernetzte Schicht für die Ausgabe.
Von diesem Modell erhoffen wir uns, dass es mit einer guten Genauigkeit, abhängig vom übergebenen Kontext, in der Lage ist Kurzzeitprognosen für räumlich verteilte Zeitreihen zu tätigen. Wir werden unser Modell auf zwei Verkehrsdatensätzen (siehe Abschnitt 4.2 und Abschnitt 4.3) testen und es mit anderen Modellen vergleichen. Als zeitlichen Kontext verwenden wir für unser Modell die übergebenen vergangenen Zeitschritte, auf dessen Basis unser Netzwerk die Prognose erzeugt [Wöl13]. Der Kontext ist abhängig vom verwendeten Datensatz und vom gewünschten Vorhersagehorizont.

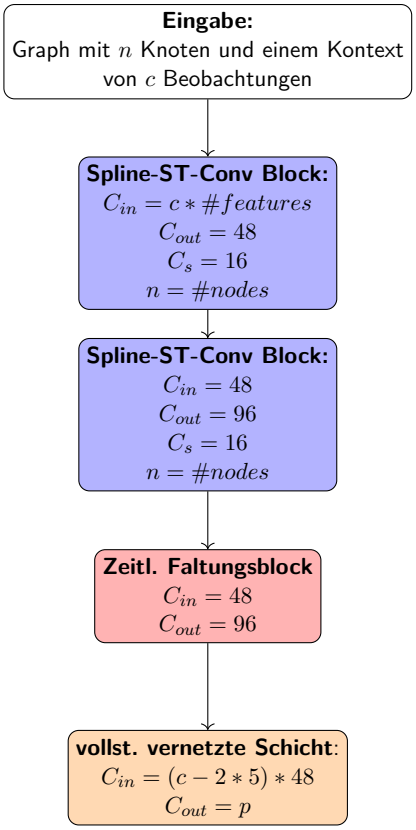


Abbildung 3.4: Übersicht über den internen Aufbau der Spline-STGCNs für einen Graphen als Eingabe mit einem Kontext c und einem Vorhersagehorizont p. Die Eingabe wird von einem Spline-ST-Conv Block verarbeitet und dessen Ausgabe dient als Eingabe eines neuen Spline-ST-Conv Blocks. Anschließend durchläuft diese Ausgabe einen letzten zeitlichen Faltungsblock. Eine vollständig vernetzte Schicht erzeugt daraus die Vorhersage für den Vorhersagehorizont p.

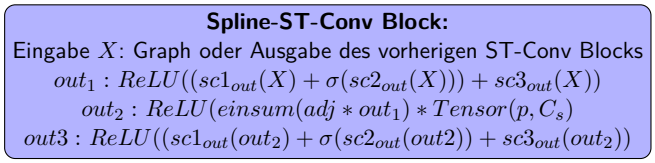


Abbildung 3.5: Die Grafik zeigt die Abfolge der Operationen innerhalb eines Spline-ST-Conv Blocks. Die Variablen $sc1_{out}, sc2_{out}$ und $sc3_{out}$ bezeichnen jeweils die Ausgabe einer Spline-CNN Schicht innerhalb des zeitlichen Faltungsblocks. Die Matrix mit den Kantengewichten wird durch adj bezeichnet. $Tensor(p, C_s)$ bezeichnet einen zweidimensionalen Tensor mit der Größe (p, C_s).

4 Experimente

In diesem Kapitel beschreiben wir die Experimente, welche wir mit den Spline-STGCNs durchgeführt haben. Um zu prüfen, inwiefern die Ergebnisse der Experimente einzuordnen sind, werden die Spline-STGCNs mit den STGCNs [YYZ17], dem Historischen Durchschnitt [PDS12], dem ARIMA-Modell mit Kalman-Filter [Geo+16], dem FC-LSTM [SVL14], den DCRNNs [Li+17] und dem ST-UNet [YYZ19] verglichen. Eine genauere Beschreibung der Vergleichsmodelle, mit Ausnahme der STGCNs (siehe Abschnitt 3.2), ist in dem Unterkapitel 4.1 zu finden.
Aufgrund der Vorteile, welche sich vermutlich aus der Verwendung der Spline-CNNs in unserem Modell ergeben, erhoffen wir uns für die Kurzzeitprognose von räumlich verteilten Zeitreihen eine höhere Genauigkeit der Vorhersagen im Vergleich zu den anderen Modellen. Da die Spline-CNNs speziell für räumliche Eingabedaten, wie beispielsweise Graphen, entwickelt wurden und sehr gut geeignet sind, zusätzliche Informationen aus dem räumlichen Zusammenhang zwischen einzelnen Knoten zu erfassen und zu verarbeiten, erwarten wir, dass unser entwickeltes Modell besser als andere Modelle in der Lage ist räumliche Daten zu verarbeiten und daraus zu lernen. Durch die Verwendung der Spline-CNNs in der zeitlichen Faltungsebene ist die Möglichkeit gegeben, dass das Modell versteckte Muster aus den räumlichen Informationen erkennt und in den Zusammenhang mit den zeitlichen Informationen setzt.
Die Verwendung der STGCNs als Framework zur Prognose von Verkehrsdaten bietet daher eine sehr gute Grundlage, um unser Modell für die Eignung zur Kurzzeitprognose von räumlich verteilten Zeitreihen zu testen. Die Modelle werden zunächst auf dem METR-LA-Datensatz getestet (siehe Abschnitt 4.2). Dieser Datensatz ist aufgrund der höheren Komplexität gut geeignet, um unser Modell für die Kurzzeitprognose räumlich verteilter Zeitreihen zu testen. Jede Messung wird als einzelner Sensorgraph dargestellt.
Zusätzlich testen wir das Modell auf dem PeMSD7(M)-Datensatz (siehe Abschnitt 4.3). Dieser Datensatz beinhaltet ebenfalls Verkehrsdaten, jedoch ist die Menge der Daten um etwa die Hälfte reduziert. Das bedeutet, das Modell muss die Wahrscheinlichkeitsverteilung anhand einer geringeren Menge von Daten lernen. Die Struktur der Daten in diesem Datensatz ist jedoch deutlich weniger komplex. Das liegt zum einen an dem Unterschied der Lage der einzelnen Verkehrssensoren und zum anderen an der Wahl der erfassten Wochentage. Während in dem METR-LA-Datensatz die Sensoren innerhalb der Stadt Los Angeles verteilt sind und den komplexen Stadtverkehr erfassen, sind die Sensoren im PeMSD7(M)-Datensatz auch außerhalb der Stadt, in der Nähe der kalifornischen Hauptstadt Sacramento, verteilt. In dem PeMSD7(M)-Datensatz werden ausschließlich die Wochentage von Montag bis Freitag erfasst. In dem METR-LA-Datensatz werden auch die Tage Samstag und Sonntag berücksichtigt. Daraus folgt zusätzlich zu der Sensorverteilung eine komplexere Struktur der Daten.
Unser Modell und die verwendeten Datensätze wurden mit PyTorch[1] in der Version 1.1.0 implementiert. Zusätzlich wurde für die Implementierung der Spline-CNNs die Bibliothek

[1] `https://pytorch.org`

PyTorch Geometric[2] verwendet. Alle Experimente wurden auf einem Windows-System mit einem Intel Core i7-7700K mit 4 Kernen und einer maximalen Taktfrequenz von 4.50 GHz, einer Nvidia Geforce GTX 1080 mit 8 Gigabyte Grafikspeicher und 16 Gigabyte Arbeitsspeicher durchgeführt.
Die Experimentergebnisse der Vergleichsmodelle stammen aus dem Paper [YYZ19].

4.1 Vergleichsmodelle

In diesem Abschnitt des Kapitels gehen wir kurz auf die verwendeten Vergleichsmodelle ein und beschreiben kurz ihren Aufbau und ihre Funktionsweise.

4.1.1 Historischer Durchschnitt

Der historische Durchschnitt ist ein Vorhersagemodell für Zeitreihen [PDS12]. Er basiert auf der Korrelation zwischen den Sensoren und dem Zeitpunkt der jeweiligen Messung. Das bedeutet, dass das Verkehrsaufkommen in einem bestimmten Straßensegment zu einem Zeitpunkt d an einem Tag w anhand der in der Vergangenheit gemessenen Werte geschätzt werden kann. Soll beispielsweise der Verkehr in einem Straßensegment an einem Freitag um 12 Uhr bestimmt werden, kann dieser mittels eines Durchschnitts der um 12 Uhr in diesem Straßensegment gemessenen Werte vergangener Freitage geschätzt werden. Um den historischen Durchschnitt formal zu beschreiben, wird ein gegebenes Zeitintervall t benötigt. Dieses ist variabel und kann von sehr kleinen Werten, wie beispielsweise einer Minute bis hin zu mehreren Stunden reichen. In einem Straßennetzwerk gibt es n Sensoren. Die Messung eines Sensors i mit $1 \leq i \leq n$ in dem Zeitintervall t wird mit $v_i[t]$ bezeichnet. Die Menge der von den Sensoren gemessenen Werte wird durch die Menge $V = \{v_i(j), i = 1, \ldots, n; j = 1, \ldots, t\}$ dargestellt. Das Ziel ist es, für die Vorhersage die Menge $V = \{v_i(j), j = t+1, t+2, \ldots t+p\}$ zu finden. Der Vorhersagehorizont wird mit der Variable p bezeichnet. Diese Variable gibt an, für welchen Zeitraum die Vorhersage berechnet werden soll. Setzen wir beispielsweise $p = 1$, soll eine Vorhersage für den Zeitpunkt $t + 1$ getroffen werden. Somit wird für jeden Sensor i die Vorhersage für das gewünschte Vorhersagefenster berechnet. Für die Berechnung der Vorhersage wird folgende Formel verwendet:

$$v\left(t_{d,w} + p\right) = \frac{1}{|V(d,w)|} \sum_{s \in V(d,w)} v(s) \tag{4.1}$$

Dabei beschreibt $V(d, w)$ die Teilmenge der vergangenen Beobachtungen, welche zum gleichen Zeitpunkt t am gleichen Tag d gemessen wurden. Damit erfasst d die Korrelation der Beobachtungen, welche zum gleichen Zeitpunkt des Tages stattfinden und w erfasst die Korrelation von Beobachtungen, welche am gleichen Wochentag stattfinden. Um zu vermeiden, dass jahreszeitlich auftretende Schwankungen die Berechnung negativ beeinflussen, werden jeweils nur Daten für die jeweilige Berechnung betrachtet, welche in der gleichen Jahreszeit erfasst wurden.

4.1.2 ARIMA mit Kalman-Filter

Bei dem Auto-Regressive Integrated Moving Average (ARIMA) handelt es sich um ein Model, welches bevorzugt für die statistische Analyse von Zeitreihen eingesetzt wird [Geo+16].

[2] `https://github.com/rusty1s/pytorch_geometric`

ARIMA selbst ist nur bedingt für die Prognose von Verkehrsdaten geeignet, da es ein lineares Modell ist. Es gibt jedoch Erweiterungen des ARIMA-Modells, welche für die Vorhersage von Zeitreihen verwendet werden. Ein Beispiel dafür ist ein ARIMA-Modell, welches ARIMA und Kalman-Filter verwendet [LTL12]. Ursprünglich wurde dieses Modell dazu entwickelt, um Vorhersagen für Windgeschwindigkeiten zu erzeugen. Es hat sich jedoch herausgestellt, dass dieses Modell auch für die Prognose anderer räumlicher Zeitreihen geeignet ist. Das ARIMA-Modell wird dazu verwendet, um das Kalman-Filter Verfahren zu initialisieren.

4.1.3 FC-LSTM

Das FC-LSTM-Modell basiert auf der Verwendung von zwei vollständig vernetzten LSTMs [Geh+17]. Es wird für Prozesse verwendet, bei denen aus einer Sequenz eine andere Sequenz erzeugt werden soll. Ein Anwendungsfall für solche Prozesse ist beispielsweise die Prognose von Verkehrsdaten anhand bereits erfasster Daten. Bei diesem Modell sind alle verdeckten Units zwischen zwei Schichten vollständig miteinander verknüpft. Ein LSTM wird verwendet, um die Beobachtungen schrittweise einzulesen. Daraus entsteht eine Vektorrepräsentation der Daten mit einer festen Dimension. Anschließend wird ein zweites LSTM verwendet, welches die Ausgabe aus dieser Vektorrepräsentation erzeugt.
Die LSTMs sind besonders gut für das Lernen auf Daten, welche eine lange zeitliche Abhängigkeit besitzen, geeignet. Das Ziel eines LSTMs ist die Schätzung einer bedingten Wahrscheinlichkeit $p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T)$. Dabei ist $(x_1, \ldots, x_T)$ die Eingabesequenz und $(y_1, \ldots, y_{T'})$ zugehörige die Ausgabesequenz. Die Längen der beiden Sequenzen werden durch T und T' repräsentiert. Diese beiden Sequenzen können unterschiedlich lang sein.

4.1.4 DCRNN

Das Diffusion Convolutional Recurrent Neural Network (DCRNN) ist ein Framework für räumlich-zeitliche Vorhersagen [Li+17]. Das Framework ist in der Lage sowohl räumliche als auch zeitliche Abhängigkeiten in den Eingabedaten zu erfassen. Für die Erfassung der räumlichen Abhängigkeiten nutzt das DCRNN bidirektionale Random Walks. Die zeitlichen Abhängigkeiten werden mittels einer Encoder-Decoder Architektur erfasst. Das DCRNN verwendet eine Diffusions-Faltungsschicht, um die räumlichen Abhängigkeiten zu erfassen. Für die Erfassung der zeitlichen Abhängigkeiten verwendet das Framework eine Erweiterung Rekurrenter Neuronaler Netzwerke (RNNs). An dieser Stelle wird eine Variation der Gated Recurrent Units (GRUs) eingesetzt. Dabei handelt es sich um eine sehr effiziente Form der RNNs. Innerhalb der GRUs wird Matrixmultiplikation eingesetzt. In diesem Framework wird die Matrixmultiplikation durch die Diffusions-Faltung ersetzt. Daraus resultieren die Diffusion Convolutional Gated Recurrent Units (DCGRUs). Für den Encoder und den Decoder werden jeweils diese DCGRUs verwendet. Während des Trainings werden die Trainingsdaten vom Encoder eingelesen. Die letzten Zustände der DCGRUs werden verwendet, um den Decoder zu initialisieren. Der Decoder erzeugt dann die Vorhersage auf Basis der gegebenen Beobachtungen.
Das Framework erzielt auf den getesteten METR-LA und PEMS-BAY sehr gute Ergebnisse im Vergleich zu den anderen, in [Li+17] getesteten, Modellen. Es zeigt eine durchschnittliche Verbesserung der Performanz von 14 Prozent.

4.1.5 ST-UNet

Das ST-UNet ist ein räumlich-zeitliches U-Netzwerk für die Vorhersage von Zeitreihen, welche als Graphen dargestellt werden können [YYZ19]. Dieses Modell nutzt die zusätzlichen Informationen, welche durch die Graphstruktur der Daten vorhanden sind. Dabei kann das ST-UNet, ähnlich wie Spline-STGCNs, durch diese zusätzlichen Informationen, die räumlichen Abhängigkeiten in den Daten besser erfassen, als beispielsweise tiefe neuronale Netzwerke oder RNNs.
Verkehrsbeobachtungen zu einem beliebigen Zeitpunkt werden durch einen Merkmalsvektor $\mathbf{x} \in \mathbb{R}^D$ repräsentiert, wobei in jedem der n Sensoren D Messwerte gespeichert sind. Das bedeutet, die Daten, welche aus dem gesamten Sensornetzwerk stammen, können als Merkmalsmatrix $\mathbf{X} = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^{n \times D}$ dargestellt werden. Für jeden Messzeitpunkt wird eine solche Matrix angelegt. Dadurch resultiert im Laufe der Zeit eine Folge von Matrizen $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_t$, welche jeweils einen Messzeitpunkt darstellen.
In dem Modell wird jede dieser Matrizen als räumlich-zeitlicher Graph betrachtet. Jeder dieser Graphen kann in zwei Darstellungen zerlegt werden. Die als Graph strukturierten Daten stellen die räumliche Ebene dar und die Zeitreihe die zeitliche Ebene. Das Modell nutzt Poolingoperationen auf den räumlich strukturierten Daten. Der Aufbau des Modells folgt einer U-Form. Es beinhaltet zwei Teile, welche symmetrisch angewendet werden. Zuerst wendet das Modell eine Faltungsoperation auf dem Graphen an, um Informationen über die Nachbarn eines jeden Knoten zu erhalten. Anschließend wird eine Poolingschicht verwendet, um gefaltete Informationen in eine räumlich-zeitliche Auflösung umzuwandeln. Darauffolgend wird eine Unpoolingschicht verwendet, um Merkmale, welche durch das Downsampling in einer reduzierten Form vorliegen, wieder auf ihre ursprüngliche Größe zu bringen. Im selben Schritt werden hochrangige Merkmale aus dem Downsampling-Schritt an diese Informationen angehängt. Letztlich wird eine weitere Faltungsschicht angewendet, um die Informationen für die erzeugte Vorhersage weiterzuleiten.
Die Ergebnisse dieses Modells sind genauso gut oder minimal besser, als die der DCRNNs aus dem Abschnitt 4.1.4.

4.2 METR-LA

Das ersten Experimente führen wir auf dem METR-LA Datensatz durch. Die Daten aus diesem Datensatz stammen von der Los Angeles Metropolitan Transportation Authority (LA-Metro). Es werden Daten aus dem Zeitraum vom 01.03.2012 bis zum 30.06.2012 von 207 Verkehrssensoren betrachtet [Li+17].
Für jeden Sensor sind zwei Werte gespeichert. Der erste Wert beschreibt die gemessene Geschwindigkeit in Meilen pro Stunde (mph) und der zweite Wert repräsentiert den Zeitstempel, wann der erste Wert gemessen wurde. Der METR-LA-Datensatz ist für viele Modelle eine größere Herausforderung, als beispielsweise PEMS-Datensätze. Das liegt daran, dass die Verkehrsverhältnisse innerhalb der Stadt Los Angeles komplexer sind. Der Verkehrsfluss ändert sich laufend und Ereignisse, wie beispielsweise Baustellen oder Unfälle treten häufiger auf. Der Unterschied im Verkehrsfluss zwischen Berufsverkehr und dem Verkehr außerhalb der Stoßzeiten sind sehr groß. Zusätzlich werden in diesem Datensatz alle Wochentage und nicht nur die Werktage betrachtet. Die Grafik 4.1 zeigt die Verteilung der 207 Verkehrssensoren [Li+17].

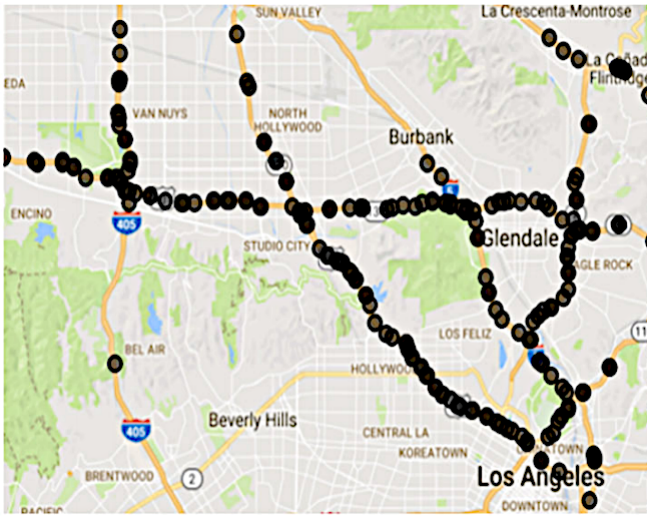


Abbildung 4.1: Die Verteilung der 207 Verkehrssensoren des METR-LA-Datensatzes [Li+17]

4.2.1 Vorverarbeitung

Es werden die ersten 60 Prozent der Daten als Trainingsdaten verwendet. Die restlichen 40 Prozent werden gleichermaßen in Validierungs- und Testdaten aufgeteilt. Das betrachtete Zeitfenster je Messung beträgt fünf Minuten. Die Werte werden mittels Z-Score Normalisierung (siehe Gleichung 4.2) normalisiert [PS15].

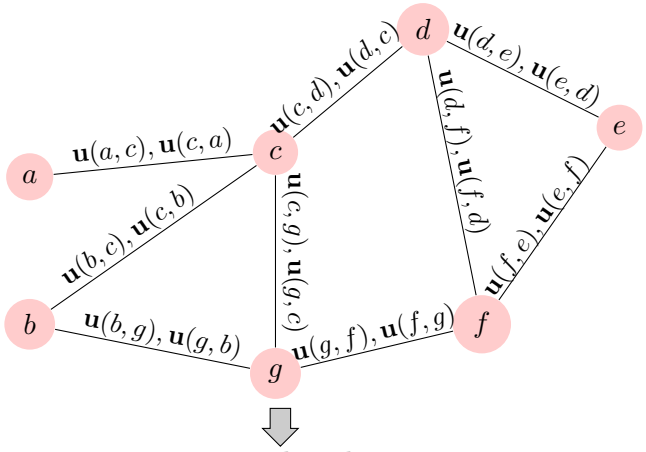
$$v_i' = \frac{v_i - \bar{E}}{std(E)} \tag{4.2}$$

Dabei beschreibt $std(E)$ die Standardabweichung der gesamten E-ten Zeile und v_i ist der Wert der Beobachtung in Zeile E und Spalte i. $\bar{E}$ ist der Mittelwert über alle Werte v_i.
Um den Sensorgraphen zu erzeugen, wird die paarweise Distanz zwischen den Sensoren berechnet. Die Distanz bezieht sich auf den Abstand, welchen die Sensoren im Straßennetzwerk zueinander besitzen und nicht mittels Luftlinie. Diese paarweise Distanz zwischen zwei Sensoren v_i und v_j wird mithilfe einer schwellwertbehafteten Gaussischen Gewichtsfunktion (siehe Gleichung 4.3) berechnet [Shu+12].

$$W_{i,j} = \begin{cases} \exp\left(-\frac{[dist(v_i,v_j)]^2}{2\theta^2}\right) & \text{wenn } \operatorname{dist}(i,j) \leq \kappa \\ 0 & \text{sonst} \end{cases} \tag{4.3}$$

Das bedeutet, wenn die Distanz zwischen den zwei Sensoren v_i und v_j kleiner als ein Schwellwert κ ist, wird das Kantengewicht mittels dieser Formel berechnet. Ansonsten ist das Kantengewicht 0. Durch den Schwellwert kann reguliert werden, wie viele Kanten des Graphen berücksichtigt werden. Der Parameter θ beschreibt die Standardabweichung der Distanzen.
Damit die Faltungsschichten der Spline-CNNs innerhalb der Spline-STGCNs die Daten aus dem Datensatz verarbeiten können, müssen die Daten entsprechend angepasst werden. Dafür wandeln wir die Adjazenzmatrix des Graphen in das COO-Format um. Das bedeutet, dass eine Kante des Graphen durch ihren Start- und Endknoten definiert ist. Anschließend werden aus der normalisierten Adjazenzmatrix die Kantengewichte erzeugt. Diese verwenden wir als Pseudokoordinaten für die Spline-CNNs. Jeder Knoten enthält

Graph mit Kantengewichten



Knotenmerkmalsmatrix

$$\begin{pmatrix} v_t(a)\ v_{t-1}(a)\ v_{t-2}(a) \dots v_{t-c}(a) \\ v_t(b)\ v_{t-1}(b)\ v_{t-2}(b) \dots v_{t-c}(b) \\ v_t(c)\ v_{t-1}(c)\ v_{t-2}(c) \dots v_{t-c}(c) \\ v_t(d)\ v_{t-1}(d)\ v_{t-2}(d) \dots v_{t-c}(d) \\ v_t(e)\ v_{t-1}(e)\ v_{t-2}(e) \dots v_{t-c}(e) \\ v_t(f)\ v_{t-1}(f)\ v_{t-2}(f) \dots v_{t-c}(f) \\ v_t(g)\ v_{t-1}(g)\ v_{t-2}(g) \dots v_{t-c}(g) \end{pmatrix}$$

Adjazenzmatrix

$$\begin{bmatrix} a\ b\ b\ c\ c\ c\ c\ d\ d\ d\ e\ e\ f\ f\ f\ g\ g\ g \\ c\ c\ g\ a\ b\ d\ g\ c\ e\ f\ d\ f\ d\ e\ g\ b\ c\ f \end{bmatrix}$$

Pseudokoordinaten

$$\begin{pmatrix} 0 & 0 & \mathbf{u}(a,c) & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{u}(b,c) & 0 & 0 & 0 & \mathbf{u}(b,g) \\ \mathbf{u}(c,a) & \mathbf{u}(c,b) & 0 & \mathbf{u}(c,d) & 0 & 0 & \mathbf{u}(c,g) \\ 0 & 0 & \mathbf{u}(d,c) & 0 & \mathbf{u}(d,e) & \mathbf{u}(d,f) & 0 \\ 0 & 0 & 0 & \mathbf{u}(e,d) & 0 & \mathbf{u}(e,f) & 0 \\ 0 & 0 & 0 & \mathbf{u}(f,d) & \mathbf{u}(f,e) & 0 & \mathbf{u}(f,g) \\ 0 & \mathbf{u}(g,b) & \mathbf{u}(g,c) & 0 & 0 & \mathbf{u}(g,f) & 0 \end{pmatrix}$$

Abbildung 4.2: Die Abbildung zeigt in welche Komponenten ein Graph aufgeteilt wird, wenn er als Eingabe für die Spline-CNNs dienen soll. Jeder Knoten n enthält eine vom Kontext abhängige Anzahl an Beobachtungen $\{v_t(n), \dots, v_{t-c}(n)\}$. Diese werden in der Knotenmerkmalsmatrix gespeichert. Die Konnektivität der Knoten wird in der Adjazenzmatrix im COO-Format gespeichert. Jede Kante wird dort durch ihren Start- und Endknoten definiert. Die obere Zeile repräsentiert die Startknoten und die untere Zeile die Endknoten der Kanten. Als Pseudokoordinaten werden die Kantengewichte verwendet. Der Eintrag $\mathbf{u}(i,j) \in [0,1]^d$ für eine Kante (i,j) beschreibt, wie die Knoten miteinander verbunden sind. Der Parameter d beschreibt die Dimensionalität der Pseudokoordinaten.

eine vom Kontext abhängige Anzahl an Beobachtungen. Diese werden in einer Knotenmerkmalsmatrix gespeichert. Der Umwandlungsprozess eines Graphen, damit er als Eingabe für die Spline-CNNs verwendet werden kann, ist in der Abbildung 4.2 dargestellt.

Batchgröße	Epochen	Lernrate	Optimierung	Kontext	Fehlerfunktion
50	100	0.001	Adam	12	MSE

Tabelle 4.1: Modellparameter für die ersten 10 Experimente zur Parameterevaluierung

Mittels der Konnektivität des Graphen im COO-Format, den Knotenmerkmalen und den Kantengewichten können nun aus den vorliegenden Daten Data-Objekte erzeugt werden, welche wir anschließend zu Batches zusammenfügen. Ein Data-Objekt beinhaltet ein Abbild des gesamten Verkehrsnetzwerkes mit der gewünschten Anzahl der zu betrachtenden Zeitscheiben als Kontext in jedem Knoten. Aus den Data-Objekten wird entsprechend der Batchgröße eine Liste mit Batch-Objekten erzeugt. Ein Batch-Objekt wird aus einer Liste von Data-Objekten erzeugt.

4.2.2 Parameterevaluierung

In diesem Unterkapitel gehen wir darauf ein, warum die Parameter für die späteren Experimente des Modells und damit auch für den Vergleich mit anderen Modellen so gewählt wurden. Dabei gehen wir zunächst auf die Wahl der Lernrate und die Wahl der Batchgröße ein. Anschließend testen wir das Modell für unterschiedliche Kontextgrößen.
Für die Minimierung des Mittleren Quadratischen Fehlers des Modells wird die Adam-Optimierung [KB15] verwendet. Dieser Optimierungsalgorithmus wird in dem Abschnitt 2.1.4 genauer erklärt. Als Werte für β_1 und β_2 verwenden wir die Standardwerte der Adam-Optimierung $\beta_1 = 0.9$ und $\beta_2 = 0.999$. Die Anzahl der Trainingsepochen beträgt 100 für jedes in diesem Unterkapitel durchgeführte Experiment. Es wurden testweise Experimente mit mehr Trainingsepochen durchgeführt. Aus diesen Ergebnissen ist jedoch hervorgegangen, dass bei den meisten geglückten Experimenten der MSE, und der Trainings- und Validierungsfehler spätestens zwischen der Epoche 80 und Epoche 90 konvergieren.

Lernrate und Batchgröße

In diesem Unterkapitel beschäftigen wir uns mit einer geeigneten Lernrate und einer geeigneten Batchgröße der Spline-STGCNs für den METR-LA-Datensatz. Wir testen zunächst eine für die Adam-Optimierung gängige Lernrate von $\epsilon = 0.001$ [KB15]. Gängige Werte für die Lernrate eines neuronalen Netzwerks mit standardisierten Eingabewerten aus dem Intervall $(0, 1)$ sind kleiner als 1 und größer als 10^{-6} [Ben12].
Die zunächst gewählte Batchgröße von 50 geht aus der Arbeit [YYZ17] hervor. Als Standardkontextgröße erhalten die Spline-STGCNs in diesen Experimenten die letzten zwölf Verkehrsbeobachtungen (60 Minuten) als Eingabe. Wir wählen zunächst einen Vorhersagehorizont von $p = 15$. Dies entspricht einem Zeitraum von 15 Minuten. Als Fehlerfunktion wird der Mittlere Quadratische Fehler (MSE) verwendet [Bot18]. Das bedeutet, die Parameter des Modells werden während des Trainings insofern angepasst, dass der MSE immer weiter reduziert wird. Um die Performanz des Modells zu messen, wird in diesen Experimenten der Mittlere Absolute Fehler (MAE) verwendet.
Wir haben zehn Experimente mit den Spline-STGCNs und den in der Tabelle 4.1 genannten Parametern durchgeführt. Der interne Aufbau des Modells entspricht dabei dem aus Unterkapitel 3.
Die Trainingsdauer beträgt für 100 Epochen etwa 35700 Sekunden. Das sind ungefähr 10 Stunden. Diese ist zusammengesetzt aus einer Trainingszeit je Iteration von 0.87 Se-

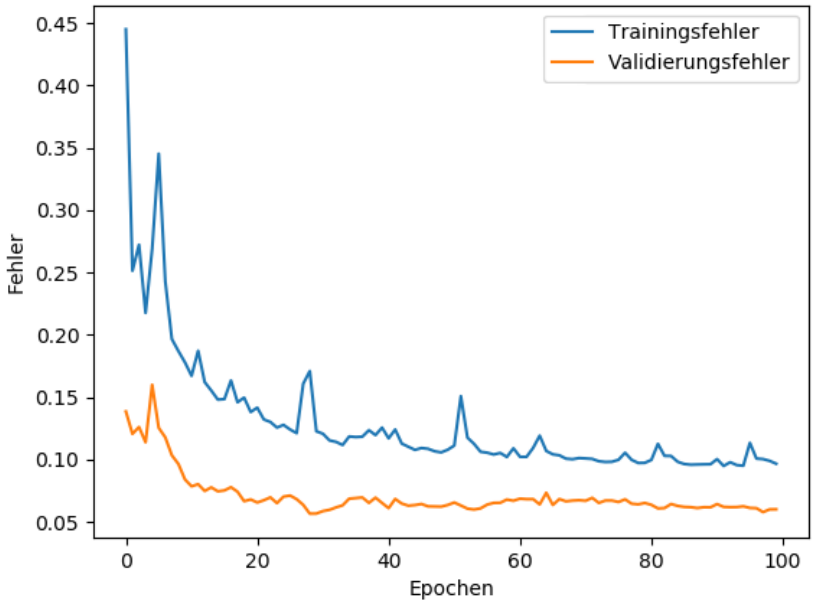


Abbildung 4.3: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.001$ und einem Kontext von $c = 12$.

kunden, 410 Iterationen je Epoche und 100 Trainingsepochen. Draus ergibt resultiert eine mittlere Trainingszeit je Epoche von 357 Sekunden. In der Grafik 4.3 ist der Mittelwert des Trainings- und Validierungsfehlers der einzelnen Experimente abgebildet. Wir sehen, dass zu Beginn des Trainings sowohl der Trainingsfehler als auch der Validierungsfehler jeweils sehr hoch ist. Das liegt daran, dass aufgrund der kurzen Trainingszeit, das Modell noch nicht die Möglichkeit hatte, die erlernbaren Parameter anzupassen und somit die Zielfunktion korrekt anzunähern. Im Laufe des Trainings werden sowohl der Trainings- als auch der Validierungsfehler immer geringer. Das ist ein Zeichen dafür, dass das Modell die erlernbaren Gewichte richtig anpasst. Da der Validierungsfehler geringer als der Trainingsfehler ist, können wir sowohl eine Überanpassung als auch eine Unteranpassung des Modells ausschließen. Der Validierungsfehler konvergiert ab Epoche 40 gegen 0.06 und der Trainingsfehler gegen 0.12. Die Grafik 4.4 zeigt den Mittelwert des Mittleren Absoluten Fehlers über die 10 Experimente, welche mit den genannten Parametern und den Spline-STGCNs durchgeführt wurden. Auch hier zeigt sich ein ähnliches Bild, wie bei dem Trainings- und Validierungsfehler in Abbildung 4.3. Jedoch wird mittels dieser Grafik deutlich, dass der MAE bei den Experimenten sehr sprunghaft ist. Er wird zwar mit der Zeit immer kleiner, bis er bei einem Wert von ungefähr 3.0 konvergiert, jedoch sind in der Kurve Sprünge erkennbar.
Aufgrund dieser Erkenntnis wurde das Modell erneut mit einer geringeren Lernrate von $\epsilon = 0.0001$ trainiert. Durch die geringere Lernrate werden die Parameter des Modell in kleineren Schritten angepasst. Auch hier wurden erneut 10 Experimente durchgeführt. Alle anderen Parameter bleiben unverändert. Sowohl in der Grafik 4.5 als auch in der Abbildung 4.6 ist erkennbar, dass die Kurven deutlich weicher und keine großen Sprünge mehr vorhanden sind. Daraus können wir schließen, dass die verwendete Optimierungsme-

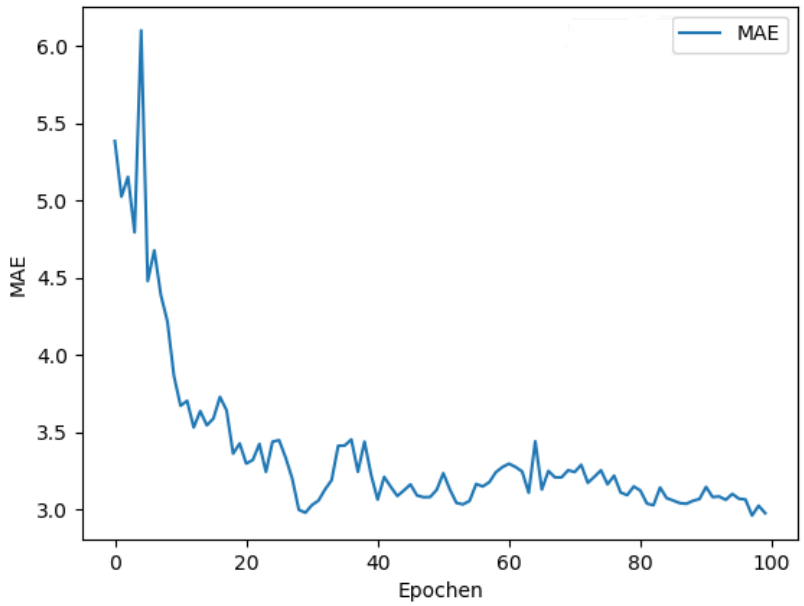


Abbildung 4.4: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.001$ und einem Kontext von $c = 12$.

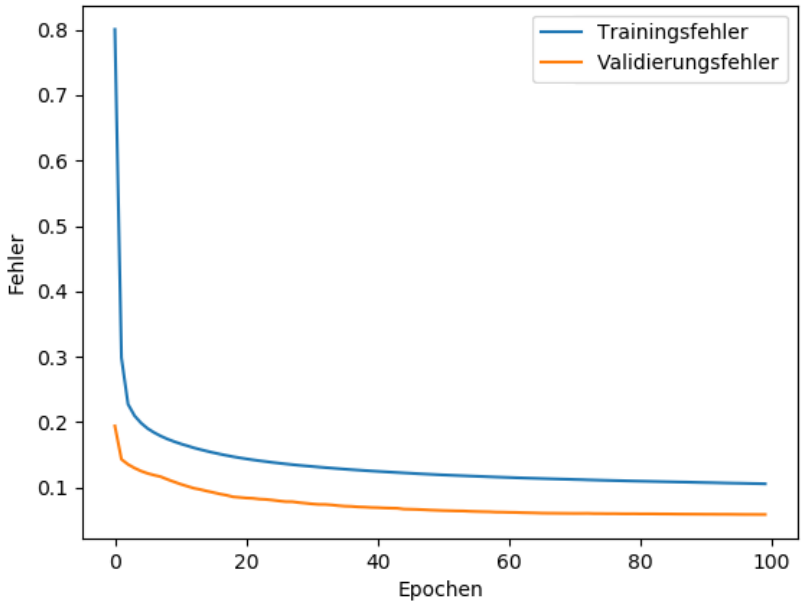


Abbildung 4.5: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

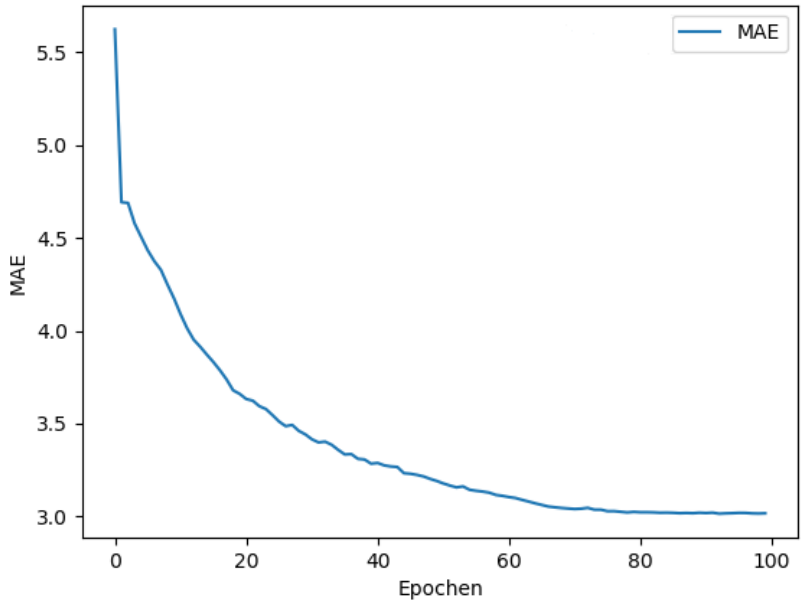


Abbildung 4.6: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

thode mit einer Lernrate von $\epsilon = 0.0001$ besser funktioniert und den Fehler innerhalb des Modells besser minimieren kann. Die Ergebnisse ändern sich trotz der geänderten Lernrate kaum. Der Hauptunterschied ist, dass das Modell ab Epoche 80 deutlicher konvergiert. Der Trainingsfehler nähert sich bei diesem Experimentaufbau 0.12, der Validierungsfehler 0.06 und der Mittlere Absolute Fehler 2.8 an. Die Trainingszeit verändert sich mit der Änderung der Lernrate kaum. Die Gesamttrainingszeit beträgt 38950 Sekunden. Das sind ungefähr 10.8 Stunden. Die leichte Erhöhung der Dauer je Iteration (0.95 Sekunden) kann auf die Verringerung der Lernrate zurückgeführt werden.
Nach der Anpassung der Lernrate wird das Modell mit verschiedenen Batchgrößen getestet. Gängige Batchgrößen reichen von kleinen Werten wie beispielsweise 1 bis hin zu einigen hundert Beispielen pro Batch. Welche Batchgröße geeignet ist, ist abhängig von dem zu lösenden Problem [Ben12]. Die Spline-STGCNs werden für die Batchgrößen 32, 50, 64, 100, 128 und 256 getestet. Wie sich das Modell mit einer Batchgröße von 50 verhält, wird aus den bereits durchgeführten Experimenten ersichtlich, deshalb werden in den kommenden Absätzen die verbleibenden Batchgrößen getestet. Die Experimente mit einer Batchgröße von 50 dienen als Referenzwert.
Für jede Batchgröße wurden aus Zeitgründen drei Experimente mit ansonsten gleichen Parametern durchgeführt, um sicherzustellen, dass schlechte Ergebnisse nicht auf ein missglücktes Experiment zurückzuführen sind. Die Aufteilung der Daten in Trainings-, Validierungs- und Testdaten bleibt ebenfalls unverändert, damit die Ergebnisse nicht auf eine zufällige, für das Modell günstige, Konstellation der Daten zurückzuführen sind.
Bei einer Batchgröße von 32, 100 Trainingsepochen und einer Lernrate von $\epsilon = 0.0001$ konvergiert der Trainingsfehler der Spline-STGCNs gegen 0.12, der Validierungsfehler gegen 0.07 und der MAE gegen 3.5 (siehe Abbildungen 4.7 und 4.8).

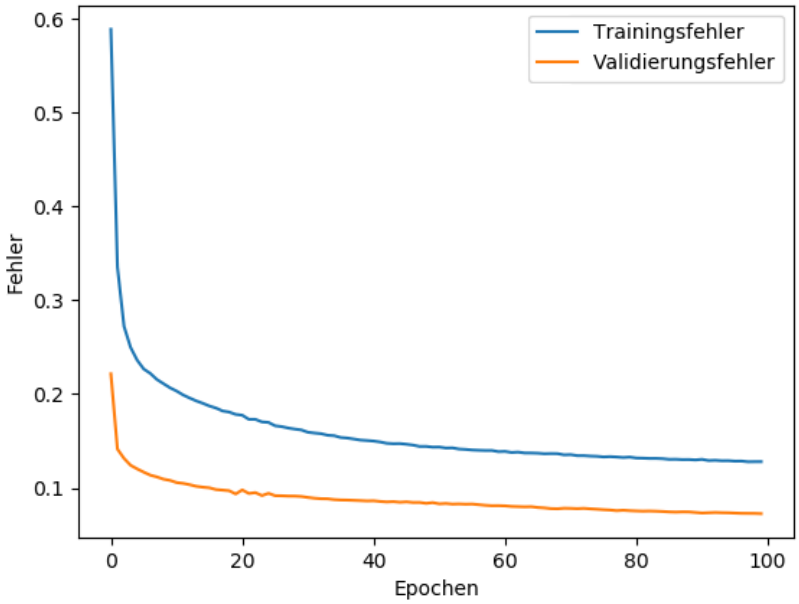


Abbildung 4.7: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 32 und einer Lernrate von $\epsilon = 0.0001$.

Anhand dieses Experimentaufbaus wird ersichtlich, dass das Modell ungefähr die gleiche Performanz erreicht, jedoch auf Grund der geringeren Batchgröße länger für ein ähnliches Ergebnis benötigt. Aufgrund der geringen Anzahl an Trainingsbeispielen je Batch muss das Modell mehr Iterationen je Epoche durchführen. Das hat einen erhöhten Zeitaufwand zur Folge.
Als Nächstes testen wir die Spline-STGCNs mit einer Batchgröße von 64. Der Validierungsfehler und der Mittlere Absolute Fehler werden zu Beginn des Trainings größer und springen dann zwischen großen Werten hin und her. Der Trainingsfehler hingegen nimmt über die Trainingszeit ab (siehe Abbildung 4.9 und 4.10). Der Trainingsfehler konvergiert gegen 0.19. Der Validierungsfehler pendelt zwischen 4.3 und 7.9 und der MAE zwischen 43.2 und 44.9. Das ist ein Zeichen dafür, dass das Modell nicht in der Lage ist, die richtigen Parameter aus den Trainingsdaten zu lernen, sondern es lernt ausschließlich die Beispiele aus den Trainingsdaten zu erzeugen. Dieser Effekt deutet auf eine Überanpassung des Modells für diese Batchgröße hin. Somit ist diese Batchgröße für unser Modell mit diesem Datensatz nicht geeignet.
Anschließend wird die Batchgröße der Spline-STGCNs auf 100 erhöht. Da das Modell gute Ergebnisse für eine Batchgröße von 50 erzielt hat, ist es naheliegend, das Modell für ein Vielfaches der funktionierenden Batchgröße zu testen. Damit wollen wir prüfen, bis zu welcher Batchgröße das Modell gute Ergebnisse erzeugt und für welche Batchgröße das Modell besser die Wahrscheinlichkeitsverteilung der zugrundeliegenden Daten zu erlernt.
Die restlichen Parameter des Modells bleiben für die Experimente unverändert. Es wurden auch in diesem Testfall wieder drei Experimente mit den gleichen Parametern durchgeführt. Der Trainingsfehler liegt zum Ende des Trainings bei 0.08, der Validierungsfehler bei 0.07 und der Mittlere Absolute Fehler bei 3.35. Die Abbildungen 4.11 und 4.12 stellen

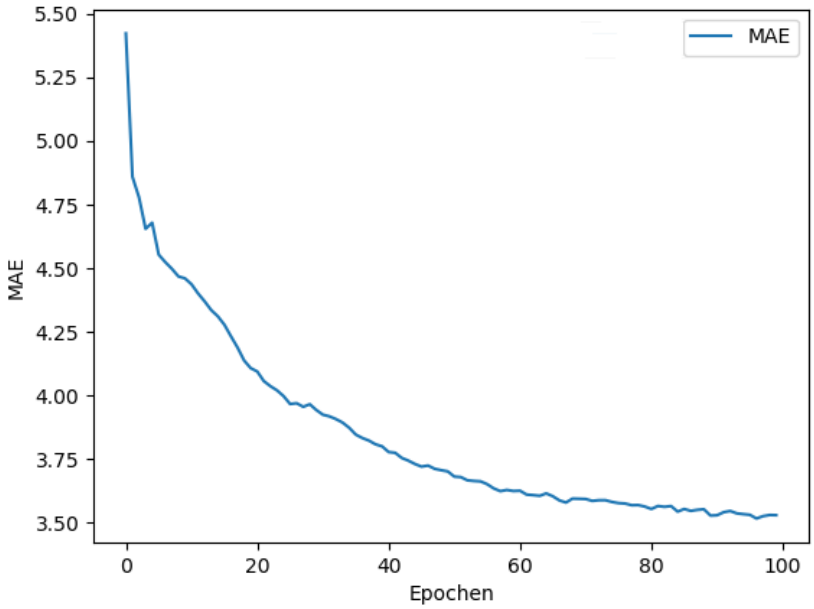


Abbildung 4.8: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 32, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

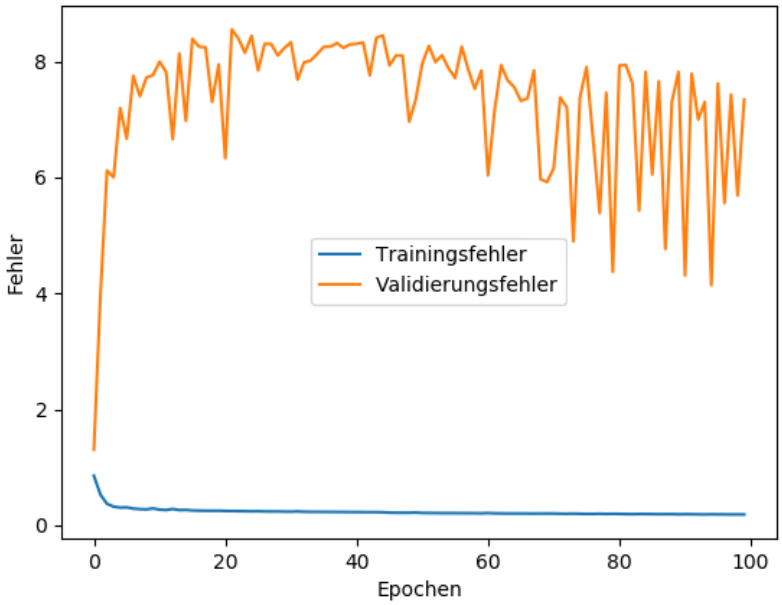


Abbildung 4.9: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 64, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

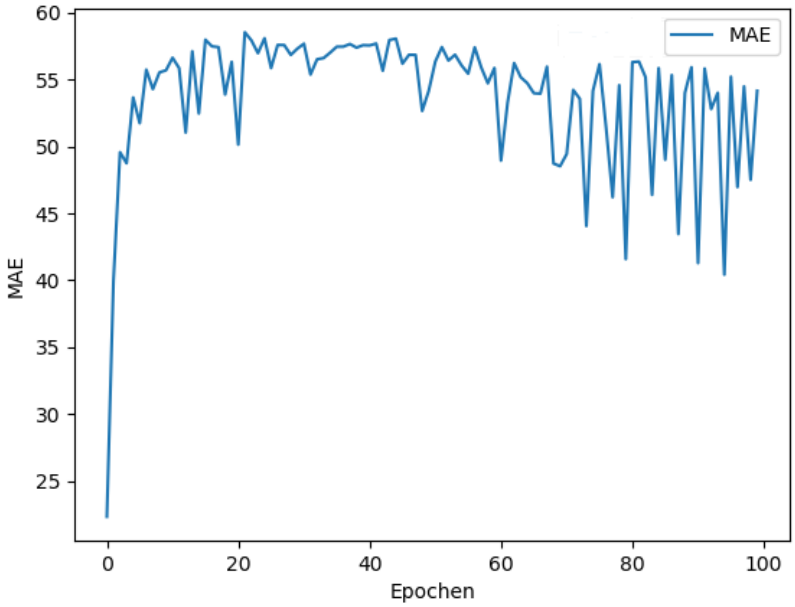


Abbildung 4.10: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 64, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

die Ergebnisse grafisch dar. Interessant an diesen Ergebnissen ist der Verlauf des Mittleren Absoluten Fehlers. Obwohl die Kurve in Abbildung 4.12 den Mittelwert aus den drei Experimenten darstellt, zeigt der Kurvenverlauf besonders in der zweiten Hälfte des Trainings immer wieder Ausreißer nach unten. Ein Grund dafür könnte eine nicht passende Lernrate für die Batchgröße sein. Da die Lernrate angibt, in welcher Schrittgröße die Parameter des Modells angepasst werden. Wenn eine Batchgröße von 100 gewünscht ist, müssten die anderen Parameter des Modells entsprechend angepasst werden. Eine Reduzierung der Lernrate könnte dem Modell an dieser Stelle helfen, den Fehler zu verringern. Um zu testen, ob das Modell mit einer großen Batchgröße besser arbeitet, wird für das nächste Experiment die Batchgröße auf 128 erhöht. Die restlichen Parameter bleiben weiterhin unverändert. Auch in diesem Fall werden für die Reproduzierbarkeit der Ergebnisse erneut drei Experimente mit denselben Parametern durchgeführt. Die Abbildungen 4.13 und 4.14 zeigen ähnliche Ergebnisse wie bei den Experimenten mit einer Batchgröße von 64. Zwar zeigen sowohl der Validierungsfehler als auch der Mittlere Absolute Fehler weniger Sprünge auf, jedoch steigen sie mit der Trainingszeit immer weiter an. Auch an dieser Stelle weisen die Ergebnisse auf eine Überanpassung des Modells hin. Aufgrund der vorherigen Ergebnisse geht die Frage hervor, ob das Modell für eine Batchgröße entsprechend einer Zweierpotenz größer als 2^5 nicht geeignet ist. Deshalb wurde noch ein weiter Experimentaufbau bezüglich der Batchgröße durchgeführt. Die Spline-STGCNs erhalten diesmal Trainingsdaten mit einer Batchgröße von 256 als Eingabe. Wie in den anderen Experimenten auch, bleiben die restlichen Modellparameter unverändert und das Experiment wird drei Mal wiederholt. Der Trainingsfehler konvergiert gegen 0.24, der Validierungsfehler gegen 0.2 und der Mittlere Absolute Fehler gegen 4.55 (siehe Abbildungen 4.15 und 4.16). Das Modell funktioniert mit dieser Batchgröße schlechter als mit einer Batchgröße

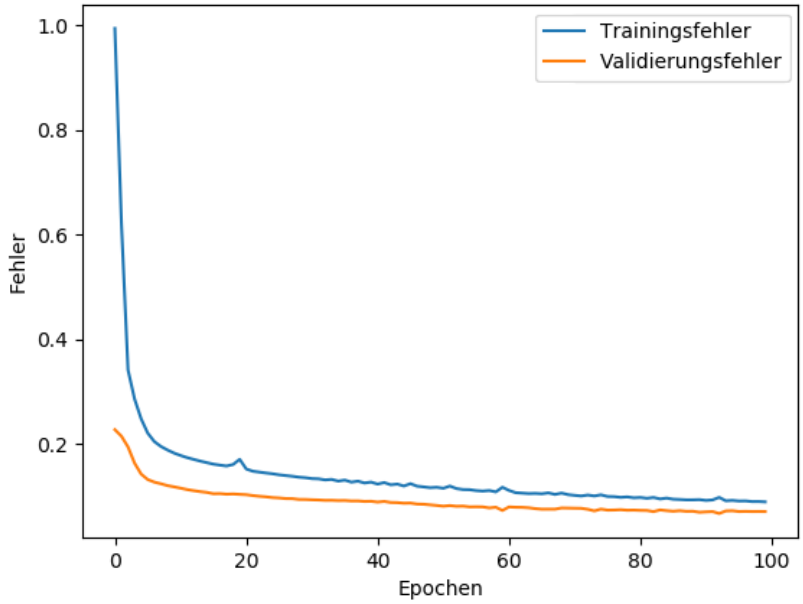


Abbildung 4.11: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 100, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

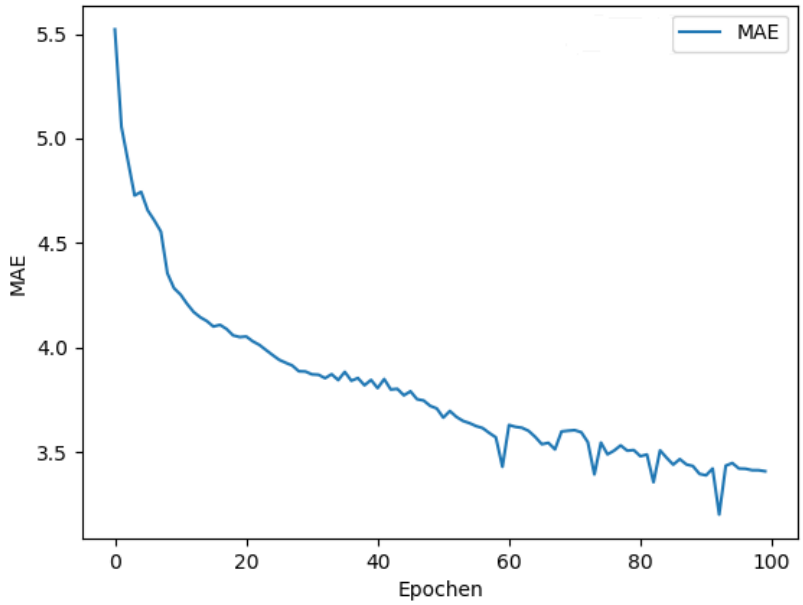


Abbildung 4.12: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 100, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

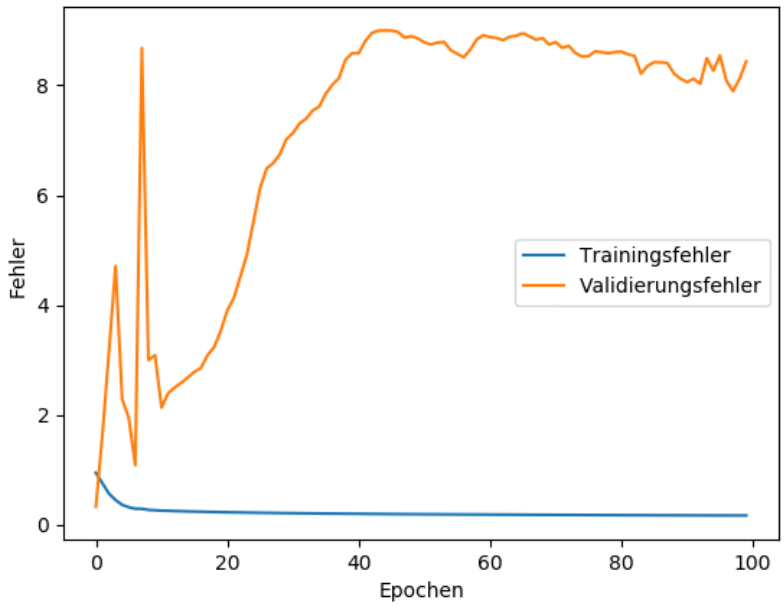


Abbildung 4.13: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 128, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

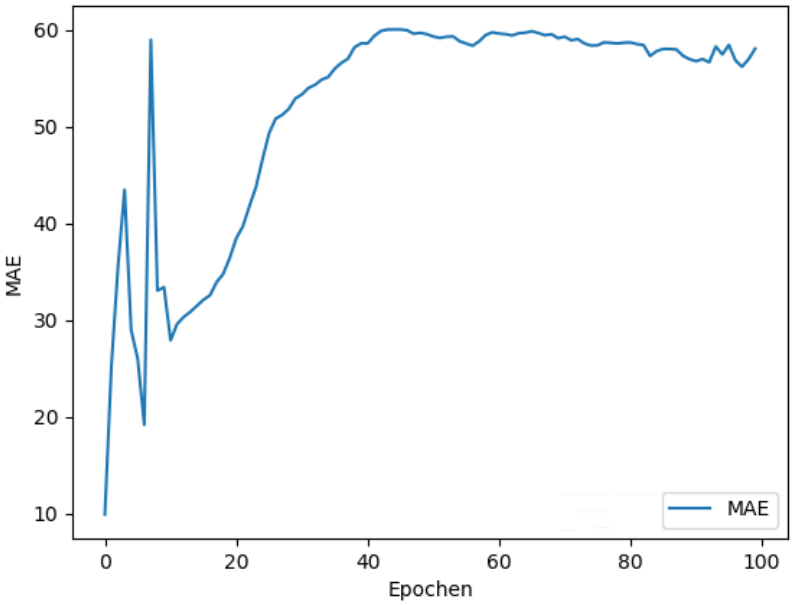


Abbildung 4.14: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 128, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

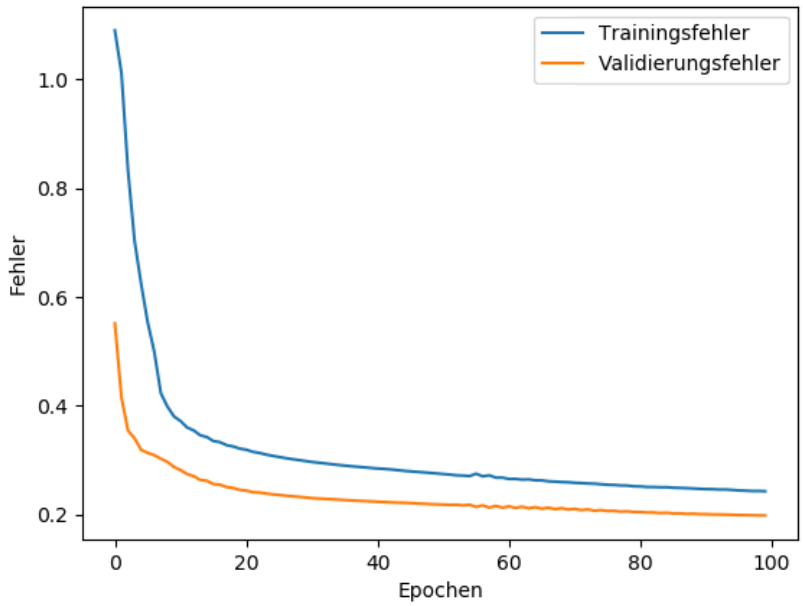


Abbildung 4.15: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 256, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

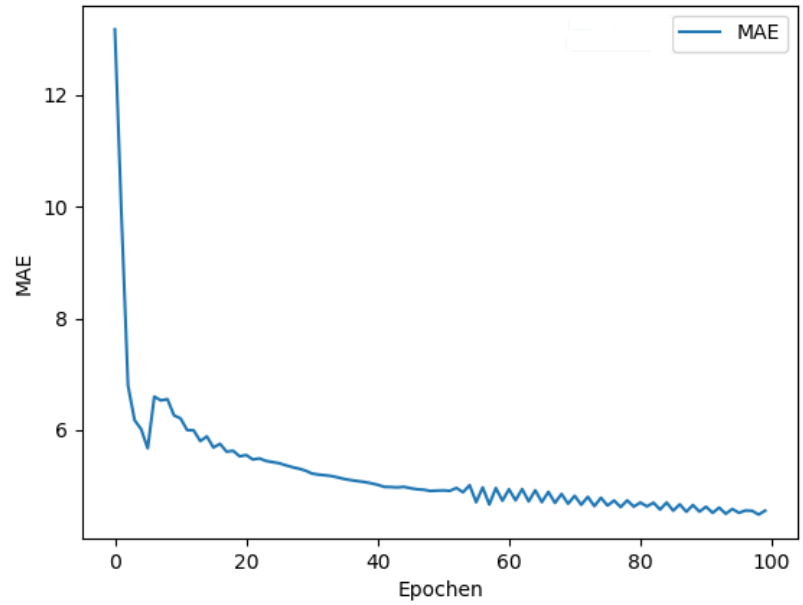


Abbildung 4.16: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 256, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

Batchgröße	Epochen	Lernrate	Optimierung	Kontext	Fehlerfunktion
50	100	0.0001	Adam	12	MSE

Tabelle 4.2: Modellparameter für die weiteren Experimente auf dem METR-LA-Datensatz

von 50. Die Ergebnisse sind jedoch nicht so schlecht, wie bei den Experimenten mit den Batchgrößen 64 und 128. Das bedeutet, dass das Modell auch für Batchgrößen, welche einer Zweierpotenz entsprechen, funktioniert. Der Mittlere Absolute Fehler springt auch bei diesem Experiment ab der Hälfte der Trainingszeit hin und her, ähnlich wie bei dem Experimentaufbau mit einer Batchgröße von 100 (siehe Abbildung 4.12). Auch an dieser Stelle könnte ein Grund dafür eine nicht passende Lernrate sein. Aber etwa Epoche 55 sind Sprünge im MAE erkennbar. Dies wäre eine geeignete Stelle, um die Lernrate zu reduzieren oder andere Parameter des Modells anzupassen, um zu prüfen, ob das Modell den Fehler für die Batchgröße weiter reduzieren kann.
Aufgrund der Testergebnisse haben wir uns für die weiteren Experimente auf dem METR-LA-Datensatz für eine Lernrate von $\epsilon = 0.0001$ und einer Batchgröße von 50 entschieden. In der Tabelle 4.2 sind die Modellparameter für die folgenden METR-LA Experimente dargestellt.

Kontextgröße

Die Kontextgröße c der Spline-STGCNs beschreibt die Anzahl der betrachteten Zeitfenster je Knoten für die Prognose. Als Startwert wählen wir eine Kontextgröße von 12, da wir für diese Kontextgröße Referenzwerte aus den bereits durchgeführten Experimenten besitzen. In diesem Unterkapitel werden nachfolgend noch eine Kontextgröße von 6 (30 Minuten) und 24 (60 Minuten) getestet. Wir wollen dabei prüfen, inwiefern sich die Leistung des Modells mit einem anderen Kontext verändert und welche Kontextgröße für die nachfolgenden Experimente verwendet wird. In diesem Kapitel werden, wie im vorherigen Kapitel auch, alle Experimente drei Mal mit denselben Parametern durchgeführt, um sicherzustellen, dass die Ergebnisse repräsentativ sind.
Als ersten Experimentaufbau testen wir die Spline-STGCNs mit einem Kontext von $c = 12$. Da zwischen den einzelnen Beobachtungen ein Zeitraum von 5 Minuten liegt, entsprechen 12 Beobachtungen einem Zeitraum von 60 Minuten. Die Lernrate und die Batchgröße betragen für die Experimente $\epsilon = 0.0001$ und 50, wie im vorherigen Kapitel getestet. Die Ergebnisse entsprechen dabei den Ergebnissen der bereits durchgeführten Experimente aus dem vorherigen Kapitel (siehe Abbildungen 4.5 und 4.6).
Für den zweiten Experimentaufbau ändern wie die Batchgröße des Modells auf 6. Das bedeutet, dass das Modell nur noch die letzten 30 Minuten der Verkehrsbeobachtung als Eingabe erhält und auf dieser Basis eine Prognose für die folgenden 15 Minuten erzeugen soll. Anhand des Verlaufs des Validierungsfehlers (siehe Abbildung 4.17) und des Mittleren Absoluten Fehlers (siehe Abbildung 4.18) sehen wir, dass das Modell in der Lage ist, den Verkehr der kommenden 15 Minuten auf Basis der letzten 30 Minuten vorherzusagen. Die Prognose ist nicht ganz so genau, wie bei einem Kontext $c = 12$, jedoch annehmbar. Der Trainingsfehler konvergiert gegen 0.18, der Validierungsfehler gegen 0.12 und der Mittlere Absolute Fehler gegen 5.57.
Die letzten Experimente bezüglich des Kontextes, werden mit einem Kontext von 24, also einem Zeitfenster von 120 Minuten durchgeführt. Aufgrund der deutlich höheren Trainingszeit und den gezeigten Ergebnissen, wurden die drei Experimente früher abgebrochen. Der

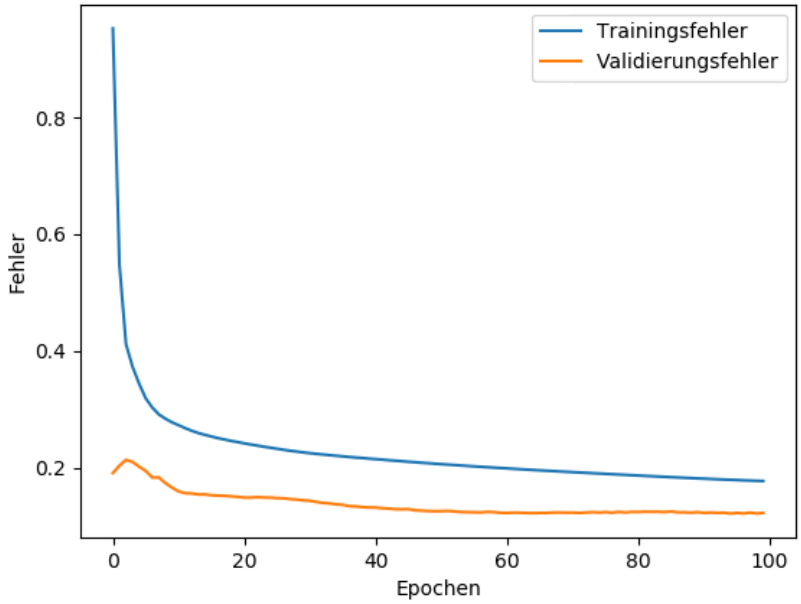


Abbildung 4.17: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 50 und einer Lernrate von $\epsilon = 0.0001$. Das Modell wurde mit einem Kontext von $c = 6$ (30 Minuten) trainiert.

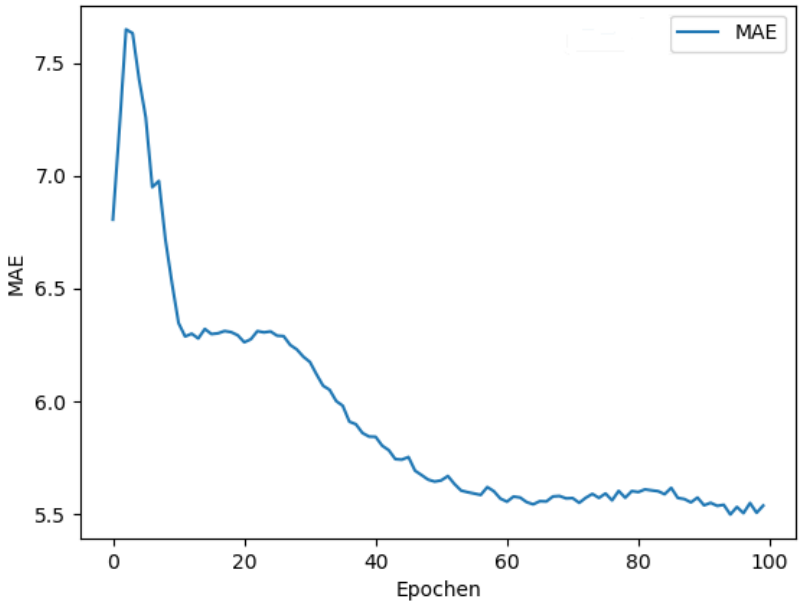


Abbildung 4.18: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 100 Epochen mit einer Batchgröße von 50 und einer Lernrate von $\epsilon = 0.0001$. Das Modell wurde mit einem Kontext von $c = 6$ (30 Minuten) trainiert.

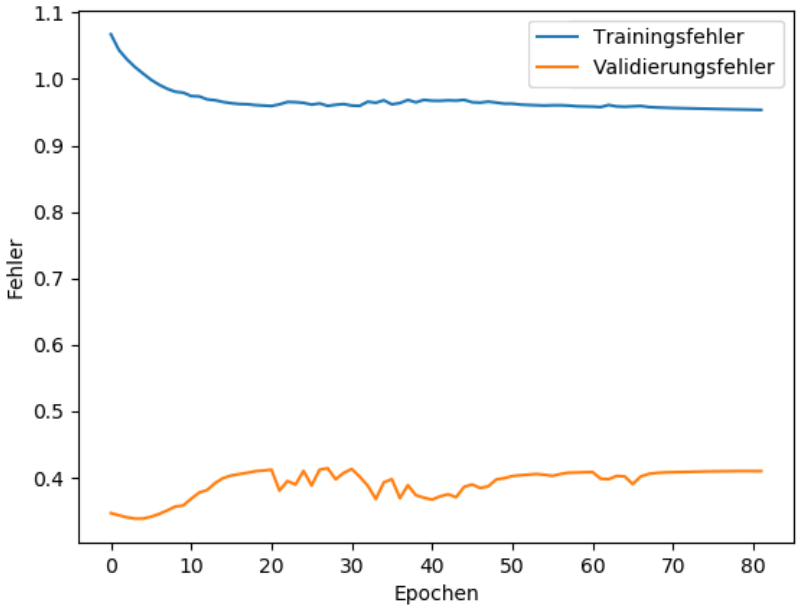


Abbildung 4.19: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs über 82 Epochen mit einer Batchgröße von 50 und einer Lernrate von $\epsilon = 0.0001$. Das Modell wurde mit einem Kontext von $c = 24$ (120 Minuten) trainiert.

Grund für die höhere Trainingszeit liegt in dem deutlich größeren Kontext. Deshalb benötigt das Modell zur Berechnung der Vorhersage einen höheren Rechenaufwand.

Die Abbildung 4.19 zeigt den Verlauf des Trainings- und des Validierungsfehlers. Im Vergleich zu vorherigen Experimenten beginnt das Modell mit vergleichsweise hohen Fehlerwerten, welche im Laufe des Trainings nicht deutlich abnehmen. Der Validierungsfehler steigt sogar über die Dauer an, statt geringer zu werden. Dieses Verhalten zeigt das Modell in allen drei Experimenten. Das hat zur Folge, dass auch der Mittlere Absolute Fehler (siehe Abbildung 4.20) nicht kleiner wird, sondern mit der Zeit wieder ansteigt.
Dieses Verhalten des Modells deutet darauf hin, dass es nicht in der Lage ist, aus den betrachteten Eingabedaten die Verteilung der zugrundeliegenden Daten zu lernen. Ein Grund dafür ist die starke Veränderung des Verkehrs über einen Zeitraum von zwei Stunden an den Messstationen. Bei Übergangszeiten, beispielsweise zwischen Berufsverkehr und nicht Berufsverkehr, sind die gemessenen Unterschiede so groß, dass für das Modell der Trend der kommenden 15 Minuten nicht ersichtlich ist.
Aus diesen Experimenten können wir schließen, dass für unser Modell ein Kontext von $c = 12$ (60 Minuten) geeignet ist, um mit den anderen gewählten Parametern eine geeignete Prognose für einen Vorhersagehorizont von $p = 15$ (15 Minuten) auf dem METR-LA-Datensatz zu erzeugen.

4.2.3 Durchführung

Für die weiteren Experimente auf dem METR-LA-Datensatz wird für die Spline-STGCNs die Adam-Optimierung [KB15] mit einer Lernrate von $\epsilon = 0.0001$ angewendet. Zusätzlich

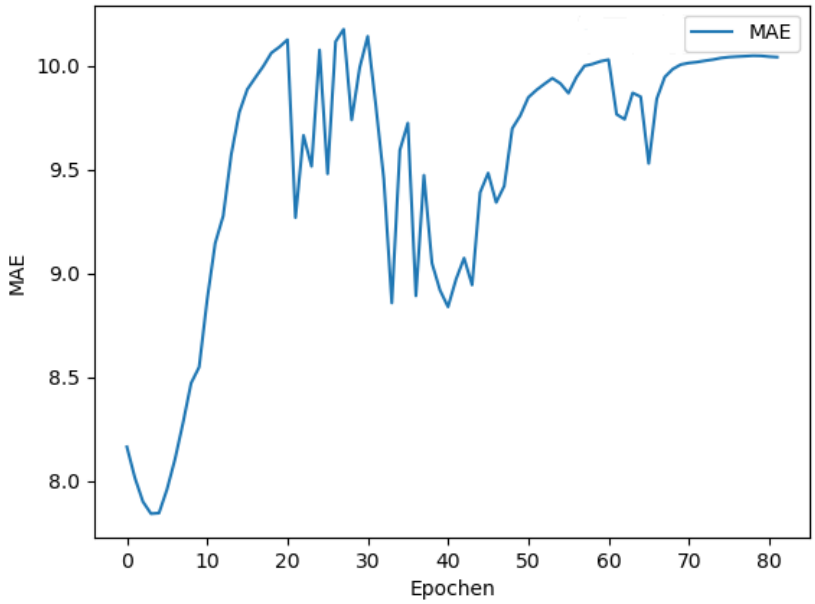


Abbildung 4.20: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs über 82 Epochen mit einer Batchgröße von 50 und einer Lernrate von $\epsilon = 0.0001$. Das Modell wurde mit einem Kontext von $c = 24$ (120 Minuten) trainiert.

wird eine Batchgröße von 50 und ein Kontext von $c = 12$ verwendet. Für die Messung der Performanz unseres Modells und für den Vergleich mit den anderen Modellen, verwenden wir den Mittleren Absoluten Fehler (MAE), die Wurzel der mittleren Fehlerquadratsumme (RMSE) und den Mittleren Absoluten Prozentualen Fehler (MAPE). Wie diese Fehlermaße berechnet werden, wird in Kapitel 2.1.2 beschrieben.
Wir testen die Genauigkeit der Vorhersagen unseres Modells bezüglich drei verschiedener Vorhersagehorizonte: $p = 15$, $p = 30$ und $p = 60$. In dem METR-LA-Datensatz beträgt der Anteil der Messwerte, welche 0 beinhalten, etwa 5 Prozent. Das bedeutet, dass sich der Verkehr an dieser Stelle entweder 5 Minuten nicht bewegt hat oder der Sensor keinen Messwert geliefert hat.
Um den MAPE zu berechnen (siehe Gleichung 2.6) wird durch die aktuelle Beobachtung geteilt. Ist diese 0, würde für die Berechnung durch Null geteilt werden. Da dies nicht erlaubt ist und der Anteil dieser Werte mit etwa 5 Prozent der Validierungsdaten gering ist, setzen wir den MAPE für diese Daten auf 100 Prozent.
Wir verwenden die ersten 60 Prozent des Datensatzes als Trainingsdaten. Die restlichen 40 Prozent der Daten werden gleichermaßen in Validierungs- und Testdaten aufgeteilt. Die Parameter für die Experimente sind in der Tabelle 4.2 dargestellt.
Es wurden 5 Experimente für jeden Vorhersagehorizont mit unserem Modell durchgeführt. Aus Zeitgründen wurden weitergehende Experimente mit einer randomisierten Reihenfolge der Daten auf diesem Datensatz leider nicht mehr durchgeführt.
Die Tabelle 4.3 zeigt die Ergebnisse der Experimente für einen Vorhersagehorizont von $p = 15$, $p = 30$ und $p = 60$. Der Durchschnitt der Ergebnisse wurde mithilfe des arithmetischen Mittels berechnet. Ausreißer der Ergebnisse zwischen den einzelnen Experimenten sind nicht vorhanden. Aus den Quellen der jeweiligen Vergleichsmodelle geht nicht hervor,

p	Fehlermaß	HA	$\text{ARIMA}_{\text{kal}}$	FC-LSTM	DCRNN	STGCN	ST-UNet	Spline-STGCN
15 Minuten	MAE	4.16	3.99	3.44	2.77	2.87	**2.72**	2.85 ± 0.27
	MAPE	13.0%	9.6%	9.6%	7.3%	7.4%	6.9%	$\mathbf{5.4}\% \pm 0.6\%$
	RMSE	7.80	8.21	6.3	5.38	5.54	**5.13**	6.13 ± 0.09
30 Minuten	MAE	4.16	5.15	3.77	3.15	3.48	**3.12**	3.21 ± 0.26
	MAPE	13.0%	12.7%	10.9%	8.8%	9.4%	8.4%	$\mathbf{5.9}\% \pm 0.5\%$
	RMSE	7.80	10.45	7.23	6.45	6.84	**6.16**	6.65 ± 0.11
60 Minuten	MAE	4.16	6.90	4.37	3.60	4.45	3.55	$\mathbf{3.50} \pm 0.29$
	MAPE	13.0%	17.4%	13.2%	10.5%	11.8%	10.0%	$\mathbf{6.44}\% \pm 0.6\%$
	RMSE	7.80	13.23	8.69	7.59	8.41	7.40	$\mathbf{6.89} \pm 0.10$

Tabelle 4.3: Die Tabelle zeigt den Vergleich der Performanz der gewählten Vergleichsmodelle [YYZ19] und der Spline-STGCNs auf dem METR-LA-Datensatz. Mit den Spline-STGCNs wurden für jeden Vorhersagehorizont p jeweils fünf Experimente durchgeführt. Die Experimente wurden auf nicht-randomisierten Daten durchgeführt.

ob die durchgeführten Experimente mit oder ohne Randomisierung der Trainingsdaten durchgeführt wurden.
Für einen Vorhersagehorizont von 15 Minuten ist unser Modell deutlich besser als der Historische Durchschnitt, ARIMA mit Kalman-Filter und FC-LSTM. Bezüglich des MAEs ähnelt das Ergebnis unseres Modells dem der STGCNs. Die Modelle DCRNN und ST-UNet liefern geringfügig bessere Ergebnisse. Der MAPE zeigt für einen Vorhersagehorizont von 15 Minuten ein gutes Ergebnis. Das kann an der Berechnung unseres MAPEs für nicht vorhandene Werte liegen. Die anderen Modelle ignorieren diese Werte für ihr Training. Der MAE und der RMSE sind absolute Fehlermaße. Das heißt, sie messen die absolute Abweichung zwischen der Vorhersage und dem tatsächlichen Wert. Der MAPE ist ein relatives Fehlermaß. Er bezeichnet die relative Abweichung zwischen den Vorhersagen der Modelle und dem tatsächlichen Wert. Da unsere Daten in dem Intervall $[0, 70]$ (renormalisiert) bzw. $[-2.7, 0.8]$ (normalisiert) liegen und unser Modell auf den normalisierten Daten trainiert wird, ist die Intervallbreite verhältnismäßig klein. Deshalb sind absolute Fehlermaße aussagekräftiger als relative Fehlermaße. In dieser Arbeit vergleichen wir trotzdem die MAPEs der verschiedenen Modelle miteinander, da sich die Aussagekraft dieses Wertes mit der Breite des Intervalls der Eingabedaten ändern kann. Somit kann der MAPE für andere Datensätze aussagekräftiger werden. Wir sehen anhand der Ergebnisse, dass der RMSE bei unserem Modell schlechter ist, als bei den anderen Modellen.
Für eine Prognose von 30 Minuten zeigt unserer Modell geringfügig andere Ergebnisse als für die Prognose von 15 Minuten. Sowohl der MAE als auch der RMSE sind vergleichbar mit dem Ergebnis der STGCNs. Die Ergebnisse der DCRNNs und des ST-UNets sind auch für diesen Vorhersagehorizont besser. Der MAPE unseres Modells ist immer noch deutlich geringer verglichen mit den anderen Modellen.
Für eine Prognose von 60 Minuten zeigen die Ergebnisse eine deutliche Veränderung. Wir sehen, dass die Spline-STGCNs bessere Ergebnisse als alle anderen Vergleichsmodelle liefern. Es ist die Tendenz erkennbar, dass unser entwickeltes Modell bessere Ergebnisse für einen größeren Vorhersagehorizont zeigt. Um zu prüfen, ob dies nicht nur vom Datensatz abhängig ist, testen wir unser Modell auf einem weiteren Verkehrsdatensatz.

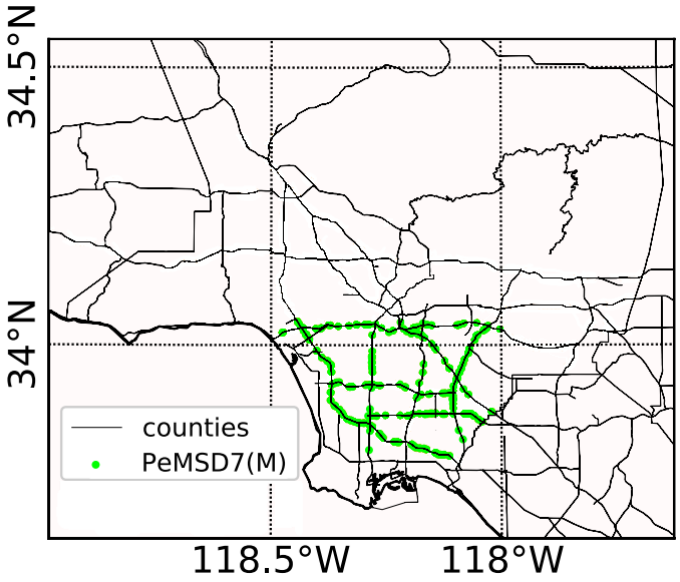


Abbildung 4.21: Verteilung der Verkehrssensoren für den PeMSD7(M)-Datensatz [YYZ17]

4.3 PeMSD7(M)

Für die weitergehenden Experimente wird der Verkehrsdatensatz PeMSD7(M) [YYZ17] verwendet. Die Verkehrsdaten wurden vom Caltrans Performance Measurement System[3] (PeMS) erfasst. Dabei handelt es sich um Verkehrsdaten des kalifornischen Verkehrsministeriums. Das gesamte Netzwerk umfasst etwa 39000 Sensoren, welche den Verkehrsfluss überwachen. Der PeMSD7(M)-Datensatz umfasst 228 Verkehrssensoren aus dem kalifornischen Distrikt 7, nahe der kalifornischen Hauptstadt Sacramento. Der Datensatz ist weniger komplex verglichen mit dem METR-LA-Datensatz, da Distrikt 7 nicht ausschließlich Stadtgebiet umfasst. Die Grafik 4.21 zeigt die Verteilung der 228 Verkehrssensoren für den PeMSD7(M)-Datensatz [YYZ17].
Die Daten stammen aus dem Messzeitraum von Mai und Juni 2012. Es werden nur die Wochentage Montag bis Freitag betrachtet. Für jeden Verkehrssensor wird die gemessene Geschwindigkeit in Meilen pro Stunde (mph) gespeichert. Das Messintervall für die Rohdaten beträgt 30 Sekunden. Aufgrund des halb so großen Messzeitraums im Vergleich zum METR-LA-Datensatz liegen für diesen Datensatz weniger Messwerte vor. Das bedeutet, dass der Trainings-, Validierungs- und der Testdatensatz jeweils um etwa die Hälfte kleiner sind. Die gemessenen Daten liegen in einem Intervall von $[3.0, 82.6]$. Durch die Normalisierung ergibt sich ein Intervall von $[-4.3, 1.8]$.

4.3.1 Vorverarbeitung

Auch bei diesem Datensatz werden, wie in Abschnitt 4.2.1, 60 Prozent der vorliegenden Daten als Trainingsdaten verwenden. Von den restlichen 40 Prozent entfallen 20 Prozent auf die Validierungsdaten und 20 Prozent auf die Testdaten. Die Messdaten liegen unverarbeitet in einem Intervall von 30 Sekunden vor. Für die Experimente werden die Daten für ein Zeitintervall von 5 Minuten aufbereitet. Das bedeutet, zwischen zwei Messwerten

[3] `http://pems.dot.ca.gov/`

liegen jeweils 5 Minuten. Daraus ergeben sich 288 Messungen pro Sensor je Tag.
Um die Adjazenzmatrix des Sensorgraphen zu erzeugen, werden Distanzen zwischen den Stationen verwendet [YYZ17]. Diese sind maßgebend für die Kantengewichte. Die Einträge der Gewichtsmatrix **W** werden mittels folgender Formel berechnet:

$$w_{ij} = \begin{cases} \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right), & i \neq j \text{ und } \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right) \geq \kappa \\ 0 & \text{, sonst.} \end{cases} \tag{4.4}$$

Die Variable w_{ij} beschreibt das Gewicht einer Kante zwischen zwei Sensoren i und j. Für den Fall, dass $\exp\left(-\frac{d_{ij}^2}{\sigma^2}\right)$ größer oder gleich als ein Schwellwert κ ist, wird das Gewicht durch $\exp\left(-\frac{d_{ij}^2}{\sigma^2}\right)$ berechnet. Ansonsten ist das Gewicht 0. Selbstkanten sind durch diese Definition ausgeschlossen. Die Variablen σ und κ sind Schwellwerte. Dadurch kann die Dichte der Gewichtsmatrix, also die Anzahl der vorhandenen Kanten bestimmt werden. Für die Experimente setzen wir σ auf 10 und κ auf 0.5 (siehe [YYZ17]).
Wie schon in dem Experimentaufbau zuvor verwenden wir als Normalisierungsmethode die Z-Score Normalisierung (siehe Gleichung 4.2).

4.3.2 Parameter

Ein Großteil der Parameter für unser Modell werden aus dem Abschnitt 4.2 übernommen. Die einzigen Ausnahmen bilden die Lernrate und die Anzahl der Epochen. Die Wahl derselben Lernrate wie im Abschnitt 4.2 ergibt zwar nummerisches ein ähnliches Ergebnis, jedoch ist die Lernkurve des Modells unstetig (siehe Abbildung 4.22 und Abbildung 4.23). Dies ist an der sprunghaften Kurve des Mittleren Absoluten Fehlers erkennbar. Aufgrund dieser Ergebnisse wird für die Experimente in diesem Kapitel für das Modell eine Lernrate von $\epsilon = 0.00001$ gewählt und die Anzahl der Trainingsepochen auf 200 erhöht, um zu prüfen, wann der Fehler des Modells konvergiert.
Die Ergebnisse der Experimente (siehe Abbildung 4.24 und Abbildung 4.25) zeigen, dass der Fehler des Modells ab etwa 150 Epochen konvergiert. Deshalb setzen wir für die folgenden Experimente die Anzahl der Trainingsepochen auf 150.
Wir verwenden für die Experimente auf dem PeMSD7(M)-Datensatz ebenfalls eine Kontextgröße von $c = 12$, da sich diese im vorherigen Experimentaufbau als geeignet erwiesen hat. Die Verkehrsdaten müssen, wie auch im vorherigen METR-LA-Experiment, in Data-Objekte umgewandelt werden. Ein Data-Objekt beschreibt dabei erneut den Sensorgraphen und jeder Sensor enthält eine vom Kontext abhängige Anzahl vergangener Messungen.

4.3.3 Durchführung

Für die Experimente auf dem PeMSD7(M)-Datensatz wird, wie im letzten Abschnitt beschrieben, eine Lernrate von $\epsilon = 0.00001$ verwendet und die Anzahl der Trainingsepochen wird auf 150 erhöht. Wir verwenden in diesen Experimenten wieder eine Batchgröße von 50.
Um auch bei diesen Experimenten die Performanz der Spline-STGCNs mit der Performanz der anderen Modelle zu vergleichen, verwenden wir wieder den Mittleren Absoluten Fehler (MAE), die Wurzel der mittleren Fehlerquadratsumme (RMSE) und den Mittleren Absoluten Prozentualen Fehler (MAPE). Wir testen die Fähigkeiten des Modells, inwiefern es in der Lage ist, auf der Basis der letzten 60 Minuten, eine Verkehrsprognose für

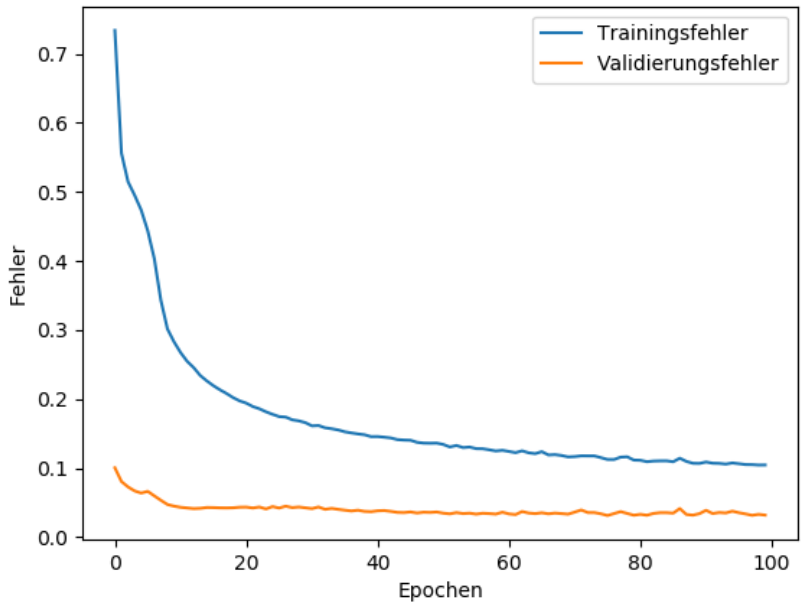


Abbildung 4.22: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs auf dem PeMSD7(M)-Datensatz über 100 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

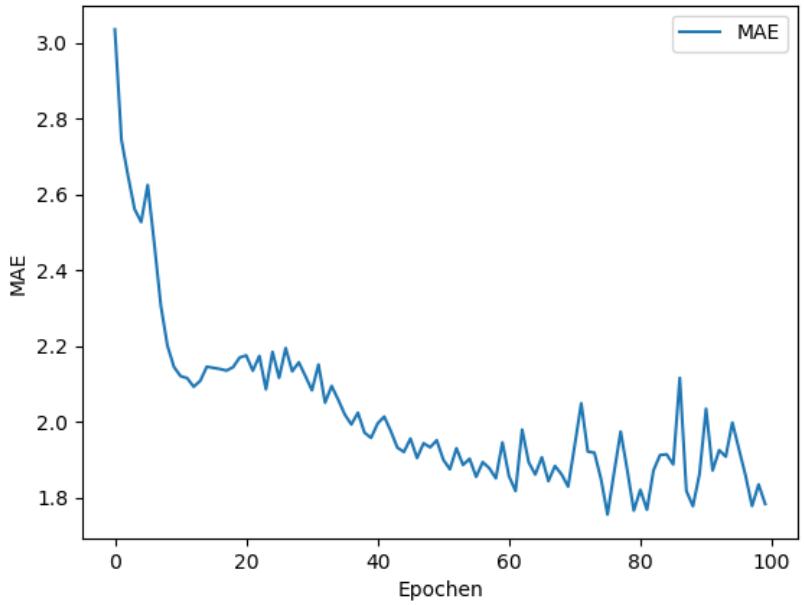


Abbildung 4.23: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs auf dem PeMSD7(M)-Datensatz über 100 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.0001$ und einem Kontext von $c = 12$.

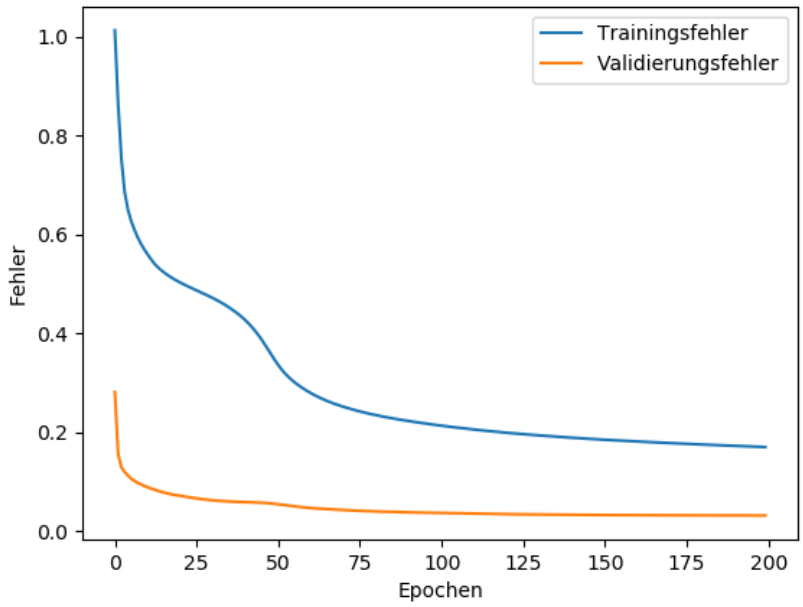


Abbildung 4.24: Mittelwert des Trainings- und Validierungsfehlers der Spline-STGCNs auf dem PeMSD7(M)-Datensatz über 200 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.00001$ und einem Kontext von $c = 12$.

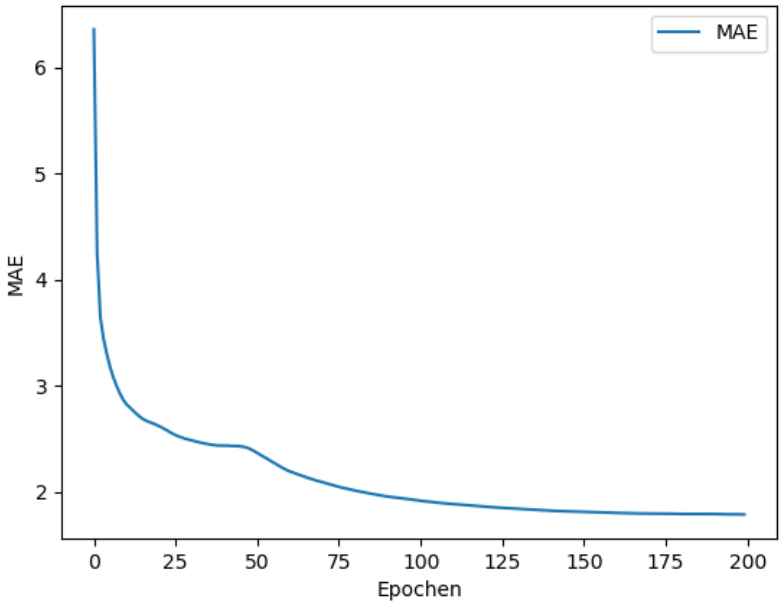


Abbildung 4.25: Mittelwert des Mittleren Absoluten Fehlers der Spline-STGCNs auf dem PeMSD7(M)-Datensatz über 200 Epochen mit einer Batchgröße von 50, einer Lernrate von $\epsilon = 0.00001$ und einem Kontext von $c = 12$.

p	Fehlermaß	HA	$\text{ARIMA}_{\text{kal}}$	FC-LSTM	DCRNN	STGCN	ST-UNet	Spline-STGCN
15 Minuten	MAE	4.01	5.55	3.67	2.25	2.25	**2.15**	2.73 ± 0.44
	MAPE	10.6%	12.9%	9.1%	5.3%	5.3%	**5.1%**	$6.2\% \pm 1.3\%$
	RMSE	7.20	9.00	6.58	4.04	4.04	**4.03**	4.33 ± 0.21
30 Minuten	MAE	4.01	5.86	3.87	2.98	3.03	**2.81**	$\mathbf{2.81} \pm 0.37$
	MAPE	10.6%	13.9%	9.6%	7.4%	7.3%	6.8%	$7.0\% \pm 1.1\%$
	RMSE	7.20	9.13	7.03	5.58	5.70	5.42	$\mathbf{4.75} \pm 0.22$
60 Minuten	MAE	4.01	6.83	4.19	3.83	4.02	**3.38**	3.82 ± 0.32
	MAPE	10.6%	17.3%	10.6%	9.9%	9.9%	**8.3%**	$8.7\% \pm 1.0\%$
	RMSE	7.20	11.48	7.79	7.19	7.64	6.68	$\mathbf{6.31} \pm 0.24$

Tabelle 4.4: Die Tabelle zeigt den Vergleich der Performanz der gewählten Vergleichsmodelle [YYZ19] und der Spline-STGCNs auf dem PeMSD7(M)-Datensatz. Mit den Spline-STGCNs wurden für jeden Vorhersagehorizont p jeweils fünf Experimente auf randomisierten Test- und Validierungsdaten durchgeführt.

die kommenden 15, 30 und 60 Minuten zu erzeugen.
Für nicht vorhandene Messwerte berechnen wir den MAPE analog zum METR-LA-Experiment. Das bedeutet, wir gehen davon aus, dass die Prognose des Modells für einen nicht vorhandenen Wert zu 100% vom Zielwert abweicht. Auf diesem Datensatz führen wir für jeden Vorhersagehorizont 5 Experimente mit randomisiert gewählten Trainings- und Validierungsdaten durch. Die Ergebnisse der Experimente der Spline-STGCNs auf dem PeMSD7(M)-Datensatz und die Vergleichswerte der anderen Modelle, können aus der Tabelle 4.4 abgelesen werden. Wir sehen, dass zwischen den einzelnen Experimenten keine deutlichen Ausreißer vorhanden sind.

Diese Ergebnisse sind auf mehreren Ebenen interessant. Zum einen ist der Anteil der Trainingsdaten zwar derselbe, wie bei dem METR-LA-Datensatz, jedoch ist die absolute Menge der Daten geringer. Das bedeutet, die Modelle haben eine reduzierte Menge an Trainingsdaten zur Verfügung, um die Verteilung der vorliegenden Daten zu lernen. Zum anderen sind die Daten des PeMSD7(M)-Datensatzes deutlich weniger komplex, als im METR-LA-Datensatz. Das liegt an dem Unterschied zwischen der Straßenführung und dem Verkehrsaufkommen in Sacramento und Los Angeles und an der ausschließlichen Betrachtung der Wochentage.
Wir sehen, dass die Vorhersagen aller Modelle auf dem PeMSD7(M)-Datensatz aufgrund der geringeren Komplexität besser sind. Für einen Vorhersagehorizont von $p = 15$ liegt der Mittelwert des MAEs über alle Experimente bei 2.73. Der Mittelwert des MAPEs unseres Modells liegt bei 6.2% und der Mittelwert des RMSEs bei 4.33. Damit liegen die Ergebnisse unseres Modells für den MAE und den RMSE leicht über den Ergebnissen der Vergleichsmodelle. Wir wissen jedoch nicht, wie bereits im Abschnitt 4.2 erwähnt, ob die Experimente mit den Vergleichsmodellen auf randomisierten Daten durchgeführt wurden. Der MAPE unseres Modells ist deutlich schlechter als bei einigen der Vergleichsmodelle. Das kann an der Berechnung unseres MAPEs liegen, dass ein nicht vorhandener Wert mit einem Fehler von 100% in die Berechnung des Fehlers eingeht und der Datensatz weniger komplex ist. Das bedeutet, der relative Fehler der anderen Modelle ist geringer.
Wir betrachten nun die Ergebnisse unseres Modells für die Experimente mit einem Vorhersagehorizont von $p = 30$. Wir sehen, dass sich der Mittlere Absolute Fehler im Vergleich zu einem Vorhersagehorizont von $p = 30$ kaum verändert. Dieser liegt für $p = 30$ im Mittel bei 2.81. Damit ist der MAE unseres Modells für diesen Vorhersagehorizont genauso gut, wie die Ergebnisse des besten Vergleichsmodells, dem ST-UNet. Der Mittelwert des MAPEs beträgt 7.0%. Auch in diesem Experimentaufbau ist davon auszugehen, dass die höheren Werte auf die Berechnung des Fehlermaßes für nicht vorhandene Werte und die Komplexi-

tät des Datensatzes zurückzuführen sind. Der Mittelwert des RMSEs der Spline-STGCNs beträgt 4.75. Dies ist deutliche Verbesserung im Vergleich zu den anderen betrachteten Modellen.
Zuletzt betrachten wir die Ergebnisse unseres Modells für einen Vorhersagehorizont von $p = 60$. Der Mittelwert des Mittleren Absoluten Fehlers beträgt 3.82. Damit sind die Ergebnisse bezüglich des MAEs nicht so gut, wie die des ST-UNets, jedoch mindestens so gut wie die der anderen Vergleichsmodelle. Der Mittelwert des MAPEs liegt für diesen Vorhersagehorizont bei 8.7%. Auch für dieses relative Fehlermaß sind die Ergebnisse unseres Modells mindestens so gut, wie die des zweitbesten Modells, der DCRNNs. Da die relativen Abweichungen der Vorhersagen der anderen Modelle mit wachsendem Vorhersagehorizont größer werden, nähert sich das Ergebnis unseres MAPEs an das der anderen Modelle an. Der Mittelwert des RMSEs unserer Spline-STGCNs liegt bei 6.31. Damit liegen wir mit dem Ergebnis für dieses Fehlermaß ebenfalls, wie bei einem Vorhersagehorizont von $p = 30$, unter den anderen Vergleichsmodellen. Diese Experimentergebnisse stützen die Aussage, dass unser Modell besser für mittel- bzw. langfristige Prognosen als für Kurzzeitprognosen geeignet ist.

4.4 Auswertung der Ergebnisse

Nach der Durchführung der Experimente auf den Datensätzen METR-LA und PeMSD7(M) wollen wir nun die Ergebnisse zusammenfassen und bewerten. Die Tabellen 4.3 und 4.4 zeigen jeweils die Ergebnisse unseres entwickelten Modells im Vergleich zu anderen für die Prognose von Zeitreihen geeigneten Modellen. Bei der Interpretation der Ergebnisse gilt es zu beachten, dass die Experimente auf dem METR-LA-Datensatz auf nicht-randomisierten Daten durchgeführt wurden. Die Experimente auf dem PeMSD7(M)-Datensatz wurden auf randomisierten Daten durchgeführt. Für die Vergleichsmodelle ist nicht sicher, ob sie auf randomisierten Daten getestet wurden.
Wir können anhand der Ergebnisse unserer Experimente und dem Vergleich der Fehlermaße mit den Vergleichsmodellen schließen, dass unsere entwickelten Spline-STGCNs generell für die Prognose räumlich verteilter Zeitreihen geeignet sind. Für einen kurzen Vorhersagehorizont ($p = 15$ und $p = 30$) sehen wir anhand der Ergebnisse auf dem METR-LA-Datensatz, dass die Ergebnisse unseres Modells vergleichbar sind mit denen der Vergleichsmodelle. Für einen längeren Vorhersagehorizont ($p = 60$) sehen wir an diesen Ergebnissen, dass die Vorhersagen unseres Modells genauer sind, im Vergleich zu den anderen getesteten Modellen. Inwiefern die Tendenz dieser Ergebnisse erhalten bleibt, wenn das Modell mit randomisierten Daten trainiert wird, muss in weitergehenden Experimenten getestet werden. Aus den Ergebnissen auf dem METR-LA-Datensatz können wir schließen, dass unser Modell auch für die Kurzzeitprognose geeignet ist, sich jedoch besser für die längerfristige Prognose räumlich verteilter Zeitreihen eignet.
Aus den Ergebnissen der Experimente auf dem PeMSD7(M)-Datensatz können wir schließen, dass unser Modell Prognosen mit weniger Ausreißern erzeugt. Das können wir anhand des niedrigen RMSEs ablesen. Die Berechnung des RMSEs ist abhängig vom MSE (siehe Gleichung 2.4). Durch die Wurzel gehen große Fehlerwerte mit einem hohen Anteil in die Berechnung des RMSEs ein. Das bedeutet, ein Modell welches im Durchschnitt sehr genaue Vorhersagen erzeugt, jedoch einige deutliche Ausreißer beinhaltet, besitzt einen höheren RMSE, als ein Modell, welches im Durchschnitt nicht ganz so präzise Vorhersagen erzeugt, jedoch weniger starke Ausreißer besitzt. Auch aus diesen Ergebnissen ergibt sich die Tendenz, dass die Vorhersagen unseres Modells für einen größeren Vorhersagehorizont

($p = 30$ und $p = 60$) besser sind, als für Kurzzeitprognosen.
Die Ergebnisse der Experimente auf den beiden Datensätzen sind vielversprechend. Deshalb sollten weitergehende Experimente auf randomisierten Daten durchgeführt werden, um zu ermitteln, inwiefern sich die Performanz des Modells verändert. Es ist zu erwarten, dass die Ergebnisse auf dem METR-LA-Datensatz schlechter werden, jedoch wären auch geringfügig schlechtere Ergebnisse immer noch gut im Vergleich zu den anderen gewählten Modellen. Es müsste zusätzlich darauf geachtet werden, inwiefern die Parameter des Modells für randomisierte Experimente auf dem METR-LA-Datensatz angepasst werden müssen. Aus Zeitgründen konnten diese weitergehenden Experimente im Rahmen dieser Arbeit leider nicht mehr durchgeführt werden.
Die durchschnittliche Trainingszeit eines Experiments auf dem METR-LA-Datensatz, mit der zu Beginn in Abschnitt 4 genannten Hardware, betrug 10.8 Stunden. Die durchschnittliche Trainingszeit eines Experiments auf dem PeMSD7(M)-Datensatz betrug 6.8 Stunden. Es wurden im Mittel 2 Gigabyte des zur Verfügung stehenden Grafikspeichers und 5 Gigabyte Arbeitsspeicher verwendet. Die Auslastung des Prozessors lag bei ca. 30 Prozent.

5 Zusammenfassung und Ausblick

In diesem Kapitel fassen wir zusammen, was in dieser Arbeit untersucht wurde. Zusätzlich gehen wir darauf ein, was sich aus den Ergebnissen dieser Arbeit schließen lässt und welche Fragen für die Zukunft offen sind.
Wir haben in dieser Arbeit ein Deep Learning Modell entwickelt, welches in der Lage ist geometrische Eingabedaten zu verarbeiten und Prognosen für räumlich verteilte Zeitreihen zu erzeugen. Das Modell ist dahingehend kontextabhängig, dass sich die Eigenschaften der Spline-STGCNs verändern, wenn ihnen ein anderer Kontext übergeben wird. Der Kontext ist abhängig vom verwendeten Datensatz und vom Vorhersagehorizont. Ein kleinerer Kontext kann abhängig vom Datensatz zu einer geringeren Trainingszeit, jedoch auch zu einer geringeren Vorhersagegenauigkeit führen. Dieses Verhalten kann durchaus gewünscht sein, wenn eine hohe Vorhersagegenauigkeit für die Problemstellung nicht relevant ist.
Wir haben unser Modell auf zwei unterschiedlich komplexen Verkehrsdatensätzen getestet, um zu prüfen, inwiefern die Spline-STGCNs in der Lage sind, Vorhersagen für räumlich verteilte Zeitreihen zu treffen. Aus den Ergebnissen können wir schließen, dass unser Modell in der Lage ist, Verkehrsprognosen mit einer guten Genauigkeit zu erzeugen. Mit steigendem Vorhersagehorizont heben sich die Prognosen unseres Modells von anderen State-of-art Modellen ab. Zusätzlich haben wir gesehen, dass die Prognosen unseres Modells weniger Ausreißer beinhalten, verglichen mit den anderen Modellen. Das zeigen die Werte des RMSEs. Bei der Parameterevaluierung unseres Modells haben wir gesehen, dass die Batchgröße, die Lernrate und die Kontextgröße einen großen Einfluss auf die Vorhersagefähigkeit und Trainingsdauer unseres Modells haben. Diese Parameter sind ebenfalls abhängig vom verwendeten Datensatz und damit von der zugrundeliegenden Wahrscheinlichkeitsverteilung. Bei der Vorverarbeitung der jeweiligen Datensätze haben wir gesehen, dass die Daten für die Verarbeitung innerhalb der Spline-CNNs umgewandelt werden müssen. Da sie nicht einfach nach dem Import verwendet werden können, nimmt dieser Schritt zusätzlich Zeit in Anspruch.
Aus der Entwicklung des Modells und den Ergebnissen der Experimente, bleiben einige Fragen offen, welche für die Zukunft interessant sind. Zunächst sollte vor der weiteren Verwendung der Spline-STGCNs getestet werden, inwiefern sich ihre Performanz verschlechtert, wenn das Training auf weiteren randomisierten Eingabedaten stattfindet. Dies ist besonders für die Ergebnisse auf dem METR-LA-Datensatz wichtig. Eine der darauffolgenden Hauptfragen, welche aus den Experimentergebnissen resultiert, ist, inwiefern die entwickelten Spline-STGCNs für Langzeitprognosen geeignet sind. Anhand der Ergebnisse ist ersichtlich, dass die Vorhersagen unseres entwickelten Modells für einen Vorhersagehorizont von 30 und 60 Minuten am vielversprechendsten sind. Mittels weitergehender Experimente für größere Vorhersagehorizonte und mit anderen Vergleichsmodellen, welche für die Langzeitprognose räumlich verteilter Zeitreihen geeignet sind, kann die Eignungsfähigkeit der Spline-STGCNs für die Langzeitprognose getestet werden. Aus guten Ergebnissen solcher Experimente kann erschlossen werden, ob die Spline-STGCNs beispielsweise für die Prognosen von Wetterdaten, für die Prognose räumlicher Verläufe von Krankheiten bzw. Epidemien oder für die Prognose von Erdbeben geeignet sind. In diesen Anwendungsgebieten sind mittel- und langfristige Prognosen relevanter als Kurzzeitprognosen.

Neben einem größeren Vorhersagehorizont ist das Testen anderer Verhältnisse von Trainingsdaten, Validierungs- und Testdaten interessant. Würde die Menge der Test- und Validierungsdaten in unseren Experimenten um jeweils 10 Prozent verringert werden, stünden dem Modell 20 Prozent mehr Trainingsdaten zur Verfügung. In der anderen Richtung kann das Modell auch mit einem geringeren Anteil an Trainingsdaten getestet werden, um zu prüfen, ob das Modell verhältnismäßig geringe Mengen an Trainingsdaten benötigt, um die zugrundeliegende Wahrscheinlichkeitsverteilung der Daten zu lernen.
Eine weitere Frage ist, wie sich die Performanz unseres Modells mit anderen Graph-Convolutions für die räumliche Faltungsschicht ändert. Es ist durchaus möglich, dass das Modell mit anderen räumlichen Faltungsschichten in der Lage ist, noch bessere Prognosen zu treffen oder seine Vorhersagegenauigkeit für kürzere Vorhersagehorizonte verbessert.
Weitergehend wäre es interessant zu betrachten, ob eine andere Form der Darstellung der Eingabedaten für die Spline-CNNs möglich ist. Vielleicht wäre es damit möglich, den Aufwand der Vorverarbeitung zu reduzieren und damit den zeitlichen Aufwand zu verringern.
Die Performanz der Spline-STGCNs könnte verbessert werden, wenn ein Verfahren für eine effizientere Parameterevaluierung vorgeschaltet wird. Das Finden geeigneter Hyperparameter des Modells ist sehr zeitaufwendig. Ein geeignetes Verfahren, wie beispielsweise der Tree-structured Parzen Estimator (TPE) [Ber+11], könnte die Parameter besser auf das Modell zugeschnitten testen und geeignete Parameter effizienter auswählen.
Der Kontext der Spline-STGCNs bietet eine weitere Möglichkeit für weitergehende Experimente. Die Zeitstempel der entsprechenden Messungen könnten dem Modell als Kontext mit übergeben und mit den Eingabedaten verknüpft werden. Vielleicht bringt dieser zusätzliche Kontext dem Modell die Möglichkeit, die Wahrscheinlichkeitsverteilung der Daten genauer zu lernen und bessere Vorhersagen zu treffen. Dafür müssten die verwendeten Datensätze entsprechend angepasst und um einen Zeitstempel je Messung erweitert werden. Für den METR-LA-Datensatz könnte in diesem Fall der zweite Wert der einzelnen Messung als Kontextvektor zusammengefasst werden. Erweiterungen des Kontexts wären für die Zukunft ebenfalls interessant. Die Spline-STGCNs könnten um einen multivariaten Kontextvektor erweitert werden, welcher beispielsweise neben dem Zeitstempel und den vergangenen Messungen die betrachtete Jahreszeit, Art der betrachteten Straßen, Wetterverhältnisse oder Urlaubszeit berücksichtigen könnte. Es wäre interessant zu prüfen, inwiefern ein solcher Kontextvektor die Performanz und die Eigenschaften des entwickelten Modells beeinflusst. Mit einem multivariaten Kontextvektor könnten die Spline-STGCNs in Zukunft mit den Conditional Sum-Product Networks (CSPNs) [Sha+19] verglichen werden. Zusätzlich wäre es interessant zu prüfen, ob die Spline-STGCNs aufgrund ihrer geringen Ausreißer für die Anomalieerkennung geeignet sind.
Ein letzter Punkt ist die Komplexität des Modells. Wir haben das Modell mit zwei Spline-ST-Conv Blöcken getestet und trainiert. Die Leistungsfähigkeit des Modells würde sich mit einer zunehmenden Anzahl dieser Blöcke wahrscheinlich positiv verändern. Inwiefern das der Fall ist und ob eine Anpassung der Komplexität für andere Problemstellungen sinnvoll ist, wären zwei weitere Punkte, die es für die Zukunft zu betrachten gilt.
Zusammenfassend können wir sagen, dass die Spline-STGCNs großes Potential für die Prognose räumlich verteilter Zeitreihen bieten. Inwiefern sie sich für den Einsatz in der Praxis eignen, muss mit weitergehenden Experimenten getestet werden.

Literatur

[AC79] Mohammed Shakeel Ahmed und Allen Rusty Cook. „Analysis of Freeway Traffic Time-Series Data by Usind Box-Jenkins Techniques“. In: 1979.

[Bar92] Anthony G. Barnston. „Correspondence among the Correlation, RMSE, and Heidke Forecast Verification Measures; Refinement of the Heidke Score“. In: 1992.

[Ben12] Yoshua Bengio. „Practical recommendations for gradient-based training of deep architectures“. In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533.

[Ber+11] James Bergstra et al. „Algorithms for Hyper-parameter Optimization“. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. USA: Curran Associates Inc., 2011, S. 2546–2554. ISBN: 978-1-61839-599-3.

[BK19] Onur Rauf Bingol und Adarsh Krishnamurthy. „NURBS-Python: An open-source object-oriented NURBS modeling framework in Python“. In: *SoftwareX* 9 (2019), S. 85–94.

[Bot18] Alexei Botchkarev. „Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology“. In: *arXiv preprint arXiv:1809.03006* (2018).

[Bro+16] Michael M. Bronstein et al. „Geometric deep learning: going beyond Euclidean data“. In: *CoRR* abs/1611.08097 (2016). arXiv: 1611.08097.

[Bru+14] Joan Bruna et al. „Spectral networks and locally connected networks on graphs“. In: *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*. 2014.

[Cho+14] Kyunghyun Cho et al. „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation“. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078.

[Chu97] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[D18] Werner D. *Funktionsanalysis*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2018. ISBN: 9783662554074.

[Dau+16] Yann N. Dauphin et al. „Language Modeling with Gated Convolutional Networks“. In: *CoRR* abs/1612.08083 (2016). arXiv: 1612.08083.

[DBV16] Michaël Defferrard, Xavier Bresson und Pierre Vandergheynst. „Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering“. In: *Advances in Neural Information Processing Systems 29*. Hrsg. von D. D. Lee et al. Curran Associates, Inc., 2016, S. 3844–3852.

[Fey+17] Matthias Fey et al. „SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels“. In: *CoRR* abs/1711.08920 (2017), S. 869–877. arXiv: 1711.08920.

[GB18] Christian Grimme und Jakob Bossek. *Einführung in die Optimierung: Konzepte, Methoden und Anwendungen*. OCLC: 1054398668. Wiesbaden: Springer Vieweg, 2018, S. 184–185. ISBN: 978-3-658-21150-9.

[GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`, abgerufen am 01. Mai 2019. MIT Press, 2016.

[Geh+17] Jonas Gehring et al. „Convolutional Sequence to Sequence Learning“. In: *CoRR* abs/1705.03122 (2017). arXiv: `1705.03122`.

[Geo+16] By George E. P. Box et al. *Time Series Analysis: Forecasting and Control*. Bd. 68. Jan. 2016. ISBN: 978-1-118-67502-1.

[Hin+12] Geoffrey E. Hinton et al. „Improving neural networks by preventing co-adaptation of feature detectors“. In: *CoRR* abs/1207.0580 (2012). arXiv: `1207.0580`.

[HK06] Rob J. Hyndman und Anne B. Koehler. „Another look at measures of forecast accuracy“. In: *International Journal of Forecasting* 22.4 (2006), S. 679–688.

[Jia+18] Renhe Jiang et al. „DeepUrbanMomentum: An Online Deep-Learning System for Short-Term Urban Mobility Prediction“. In: *AAAI*. 2018.

[KB15] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[KK16] Sungil Kim und Heeyoung Kim. „A new metric of absolute percentage error for intermittent demand forecasts“. In: *International Journal of Forecasting* 32 (Juli 2016), S. 669–679.

[KW16] Thomas N. Kipf und Max Welling. „Semi-Supervised Classification with Graph Convolutional Networks“. In: *CoRR* abs/1609.02907 (2016). arXiv: `1609.02907`.

[LB95] Y. LeCun und Y. Bengio. „Convolutional Networks for Images, Speech, and Time-Series“. In: *The Handbook of Brain Theory and Neural Networks*. Hrsg. von M. A. Arbib. MIT Press, 1995.

[LC98] Erich L. Lehmann und George Casella. *Theory of Point Estimation*. 2. Aufl. Springer-Verlag, 1998.

[Li+17] Yaguang Li et al. „Graph Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting“. In: *CoRR* abs/1707.01926 (2017). arXiv: `1707.01926`.

[Lin+18] Amanda Yan Lin et al. „Using Scaling Methods to Improve Support Vector Regression's Performance for Travel Time and Traffic Volume Predictions“. In: *Time Series Analysis and Forecasting*. Springer International Publishing, 2018, S. 115–127. ISBN: 978-3-319-96944-2.

[LTL12] Hui Liu, Hong-qi Tian und Yanfei Li. „Comparison of two new ARIMA-ANN and ARIMA-Kalman hybrid methods for wind speed prediction“. In: 2012.

[Ma+17] Xiaolei Ma et al. „Learning Traffic as Images: A Deep Convolution Neural Network for Large-scale Transportation Network Speed Prediction“. In: *Sensors* 17 (2017), S. 818. arXiv: `1701.04245`.

[MH00] Spyros Makridakis und Michèle Hibon. „The M3-Competition: results, conclusions and implications“. In: *International Journal of Forecasting* 16.4 (2000). The M3- Competition, S. 451–476.

[Mit97] Thomas M. Mitchell. *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

[MK13] W. Merz und P. Knabner. *Mathematik für Ingenieure und Naturwissenschaftler: Lineare Algebra und Analysis in R*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2013. ISBN: 9783642299803.

[MK17] W. Merz und P. Knabner. *Mathematik für Ingenieure und Naturwissenschaftler: Band 2: Analysis in R^n und gewöhnliche Differentialgleichungen*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2017. ISBN: 9783662547816.

[Nwa+18] Chigozie Nwankpa et al. „Activation Functions: Comparison of trends in Practice and Research for Deep Learning“. In: *CoRR* abs/1811.03378 (2018). arXiv: 1811.03378.

[PDS12] Bei Pan, Ugur Demiryurek und Cyrus Shahabi. „Utilizing Real-World Transportation Data for Accurate Traffic Prediction“. In: *2012 IEEE 12th International Conference on Data Mining* (2012), S. 595–604.

[Ped+11] F. Pedregosa et al. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[PLM13] Nico Piatkowski, Sangkyun Lee und Katharina Morik. „Spatio-temporal random fields: compressible representation and distributed estimation“. In: *Machine learning* 93.1 (2013), S. 115–139.

[PS15] S. Gopal Krishna Patro und Kishore Kumar Sahu. „Normalization: A Preprocessing Stage“. In: *CoRR* abs/1503.06462 (2015). arXiv: 1503.06462.

[PT96] Les Piegl und Wayne Tiller. *The NURBS Book*. second. New York, NY, USA: Springer-Verlag, 1996, S. 47–139. ISBN: 3-540-61545-8.

[RHW88] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. „Neurocomputing: Foundations of Research“. In: Hrsg. von James A. Anderson und Edward Rosenfeld. MIT Press, 1988. Kap. Learning Representations by Back-propagating Errors, S. 696–699. ISBN: 0-262-01097-6.

[RM51] H. Robbins und S. Monro. „A stochastic approximation method“. In: *Annals of Mathematical Statistics* 22 (1951), S. 400–407.

[Sca+09] Franco Scarselli et al. „The Graph Neural Network Model“. In: *Trans. Neur. Netw.* 20.1 (2009), S. 61–80.

[Sha+19] Xiaoting Shao et al. „Conditional Sum-Product Networks: Imposing Structure on Deep Probabilistic Architectures“. In: Bd. abs/1905.08550. 2019. arXiv: 1905.08550. URL: http://arxiv.org/abs/1905.08550.

[Shu+12] David I. Shuman et al. „Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Data Domains“. In: *CoRR* abs/1211.0053 (2012). arXiv: 1211.0053.

[SVL14] Ilya Sutskever, Oriol Vinyals und Quoc V. Le. „Sequence to Sequence Learning with Neural Networks“. In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215.

[SW10] „Mean Absolute Error“. In: *Encyclopedia of Machine Learning*. Hrsg. von Claude Sammut und Geoffrey I. Webb. Boston, MA: Springer US, 2010, S. 652–652. ISBN: 978-0-387-30164-8.

[Wöl13] Martin Wöllmer. „Context-Sensitive Machine Learning for Intelligent Human Behavior Analysis“. Dissertation. München: Technische Universität München, 2013.

[Wu+19] Zonghan Wu et al. „A Comprehensive Survey on Graph Neural Networks“. In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596.

[YYZ17] Bing Yu, Haoteng Yin und Zhanxing Zhu. „Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting“. In: *CoRR* abs/1709.04875 (2017). arXiv: 1709.04875.

[YYZ19] Bing Yu, Haoteng Yin und Zhanxing Zhu. „ST-UNet: A Spatio-Temporal U-Network for Graph-structured Time Series Modeling“. In: *CoRR* abs/1903.05631 (2019). arXiv: 1903.05631.

[Zhe+16] Liang Zheng et al. „Good Practice in CNN Feature Transfer“. In: *CoRR* abs/1604.00133 (2016). arXiv: 1604.00133.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 10. September 2019

Ronja van den Berg