Karlsruher Institut für Technologie

Entwicklung eines zielorientierten Dialogsystems basierend auf Memory Networks

Masterarbeit von

Stefan Georg Constantin

an der Fakultät für Informatik
Institut für Anthropomatik und Robotik (IAR)

Erstgutachter:	Prof. Dr. Waibel
Zweitgutachter:	Prof. Dr. Asfour
Betreuender Mitarbeiter:	Dr. Niehues
Zweiter betreuender Mitarbeiter:	Dr. Cho

21. April 2017 – 20. Oktober 2017

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.
Karlsruhe, 20. Oktober 2017

. . . . . .
(Stefan Georg Constantin)

Zusammenfassung

Die Steuerung technischer Geräte per natürlicher Sprache gewinnt immer mehr an Bedeutung. Um eine Aufgabe für den Nutzer zu erledigen und gegebenenfalls die Ergebnisse dieser Aufgabe auszugeben, werden oftmals in technischen Geräten, die per natürlicher Sprache gesteuert werden, aus den natürlichsprachigen Äußerungen API-Aufrufe gebildet. Diese Arbeit stellt ein Dialogsystem basierend auf End-To-End trainierbaren Memory Networks vor. Dieses Dialogsystem kann mithilfe passender Trainingsbeispiele für beliebige Domänen End-To-End trainiert werden. In dieser Arbeit wird das Dialogsystem in den Domänen Restaurantreservierung und Robotersteuerung evaluiert. Die Datensätze für die Domäne Restaurantreservierung sind die Dialog bAbI Tasks und die Datensätze für die Robotersteuerung wurden für diese Arbeit erstellt. Das Dialogsystem kann seine Äußerungen auf mehrere Arten generieren. Die getesteten Korpora haben gezeigt, dass das Dialogsystem bei nicht allzu großer Abweichung von Äußerungen des Nutzers gegenüber den Äußerungen des fiktiven Nutzers im Trainingsdatensatz sehr gute Ergebnisse bei der Generierung von API-Aufrufen bietet, wenn die Antwortgenerierung über eine Kandidatenauswahl oder Wort für Wort mithilfe eines rekurrenten neuronalen Netzes stattfindet. Die Parameter für die API-Aufrufe können dabei über mehrere Interaktionen verteilt sein.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Motivation

Menschen führen seit jeher Dialoge miteinander, um Informationen zu erhalten, Ziele zu erreichen oder einfach an der Freude der sozialen Interaktion. Für diese Dialoge entwickelten sich natürliche Sprachen.
Es erscheint demnach natürlicher, wenn die Interaktion mit Computersysteme und Roboter über Dialoge stattfindet, die in natürlicher Sprache ohne Einschränkung der Sprache, z. B. auf eine bestimmte Struktur oder ein eingeschränktes Vokabular, verfasst sind. Eine Vereinfachung der Interaktion wäre mit dem Verwenden von Dialogen zur Interaktion ebenfalls in vielen Fällen möglich.
Systeme, die diese Form der Interaktion ermöglichen, kommen in der Science-Fiction schon länger vor. Im Film 2001: Odyssee im Weltraum aus dem Jahre 1968 ermöglicht der Computer HAL 9000 die Steuerung eines Raumschiffes mithilfe von natürlicher Sprache, in Star Wars von 1977 kann mit dem Roboter C-3PO mündlich kommuniziert werden und in Iron-Man von 2008 dient das per natürlicher Sprache zu steuernde System J.A.R.V.I.S. als persönlicher Assistent.
Siri von Apple, Google Now von Google, Amazon Echo von Amazon haben Dialogsysteme vielen Menschen zugänglich gemacht, sind jedoch von der Qualität den Dialogsystemen in der Science-Fiction noch weit unterlegen. Jedoch können sie jetzt schon den Alltag erleichtern. Im Auto kann schon heute per natürlicher Sprache z. B. ein Lied zum Abspielen ausgewählt werden.
Ein nächster Schritt wäre es, Anwendungen, wie z. B. Photoshop, mit natürlicher Sprache zu steuern, ohne umständlich irgendwelche Menüstrukturen oder Tastenkombinationen auswendig lernen zu müssen.
Die immer weiter wachsende Rechenkapazität und weitere Fortschritte im Bereich des maschinellen Lernens und der Sprachverarbeitung verstärken die Hoffnung, dass zukünftige Dialogsysteme, zumindest in bestimmten Domänen, zu Anfragen des Nutzers optimale Antworten liefern.

1.2. Zielsetzung

Die Zielsetzung dieser Arbeit ist ein zielgerichtetes domänen-spezifisches Dialogsystem zu entwickeln. Dabei soll dieses Dialogsystem anpassungsfähig sein. Das heißt, dass die Domäne zwar fest sein soll, jedoch mit wenig Aufwand gewechselt werden können soll, z. B. mit dem Trainieren mit einem Datensatz aus einer anderen Domäne.
Als erstes soll das Dialogsystem für die Domäne Restaurantreservierung entworfen werden. Hierfür existiert ein Datensatz, eine Beschreibung von einem Dialogsystem und Evaluati-

onsergebnisse von diesem System (Bordes, Boureau und Weston, 2017). Bei der Evaluation dieses Dialogsystems soll ein Hauptaugenmerk auf die Erzeugung von API-Aufrufe, deren Parameter im Dialog der Nutzer gibt bzw. die vom System erfragt werden, wenn diese noch nicht vollständig geäußert wurden, gelegt werden. Die Evaluationsergebnisse des entworfenen Systems sollen dann mit den Evaluationsergebnissen des existierenden Systems verglichen werden. Es soll außerdem untersucht werden, ob das existierende System verbessert werden kann und wenn ja wie. Des Weiteren soll analysiert werden, ob mithilfe des Testdatensatzes eine gute Generalisierungsfähigkeit des Dialogsystems getestet werden kann oder ob durch Auswendiglernen des Trainingsdatensatzes gute Ergebnisse im Testdatensatz erreicht werden können. Falls der Testdatensatz keinen guten Test für die Generalisierungsfähigkeit des Dialogsystems bietet, soll diese gesondert getestet werden. Um die Tauglichkeit des Dialogsystems für andere Domänen zu testen, soll ein Korpus entwickelt werden, der Teile der aktuellen und zukünftigen Fähigkeiten des Haushaltsroboters Armar 3 (Asfour u. a., 2006) abbildet. Dieser Korpus soll so erzeugt werden können, dass er ohne viel Aufwand um weitere Fähigkeiten erweitert werden kann.

1.3. Gliederung

Einen Überblick über die Grundlagen, auf denen diese Arbeit aufbaut, wird in Kapitel 2 gegeben. In Abschnitt 2.1 ist ein Überblick über die Verarbeitung von natürlicher Sprache: Welche Aufgaben gibt es in der Verarbeitung natürlicher Sprache, was sind die Schwierigkeiten in der Verarbeitung natürlicher Sprache und wie sind Dialogsysteme aufgebaut. Der nächste Abschnitt, Abschnitt 2.2, gibt einen Überblick über künstliche neuronale Netze: Aus welchen Bestandteile bestehen sie, welche Architekturen gibt es, wie kann aus Trainingsdaten gelernt werden und wie lässt sich natürliche Sprache mit künstlichen neuronalen Netzen verarbeiten. Des Weiteren wird auf Memory Networks im Kontext von Dialogsystemen eingegangen. Die Grundlagen über künstliche neuronale Netze werden mit der Hyperparameteroptimierung abgeschlossen.
Der Stand der Forschung in Bereichen, die für das Entwickeln eines zielgerichteten Dialogsystems nötig sind, und auf welche Arbeiten aufgebaut werden kann, um die Zielsetzung aus Abschnitt 1.2 zu erreichen, wird in Kapitel 3 aufgezeigt.
Kapitel 4 stellt das entwickelte Modell des Dialogsystems vor, welches die Problemstellung aus der Zielsetzung in Abschnitt 1.2 löst. Zuerst werden die Komponenten Natural Language Unterstanding (NLU) und Dialog Manager (DM) in Abschnitt 4.1 und Natural Language Generation (NLG) in Abschnitt 4.2 beschrieben. Das Modell des Korpusgenerators, der für das Erstellen weiterer Testdaten für das Dialogsystem notwendig ist, wird in Abschnitt 4.4 beschrieben. Auf die Implementierung des Dialogsystems und des Korpusgenerators wird in Abschnitt 4.5 eingegangen. Mit der Implementierung des Dialogsystems wird systematisch getestet, welche Hyperparameterwerte gute Ergebnisse ermöglichen. In Abschnitt 4.6 wird die Systematik und die Ergebnisse dieser Hyperparameteroptimierung dargestellt.
In Kapitel 5 wird die Qualität des Dialogsystems evaluiert, um zu überprüfen, ob das entwickelte Dialogsystem die Zielsetzung erfüllt. Um die Datensätze beurteilen zu können, werden in Abschnitt 5.1 die Dialog bAbI Tasks und in Abschnitt 5.2 die für diese Arbeit erstellten Robotersteuerungsaufgaben vorgestellt. Die Ergebnisse, die mit diesen Korpora

erreicht werden konnten, werden in Abschnitt 5.3 gezeigt. Die Generalisierungsfähigkeit des Dialogsystems wird in Abschnitt 5.4 untersucht. Abschnitt 5.5 widmet sich der Frage, ob die Ergebnisse reproduzierbar sind und während dem Training sich stetig verbessern oder ob die Ergebnisse sich unstetig entwickeln. Mit der Ermittlung der Ressourcenbeanspruchung in Abschnitt 5.6 wird die Evaluation abgeschlossen.
In Kapitel 6 werden in Abschnitt 6.1 die Ergebnisse dieser Arbeit zusammengefasst und abschließend bewertet und in Abschnitt 6.2 wird aufgezeigt, welche weiteren Forschungsmöglichkeiten sich, basierend auf den Ergebnissen dieser Arbeit, anbieten.

2. Grundlagen

In diesem Kapitel werden Grundlagen eingeführt, auf denen sich im weiteren Verlauf der Arbeit bezogen und aufgebaut wird.

2.1. Verarbeitung von natürlicher Sprache

Die Verarbeitung von natürlicher Sprache, z. B. Englisch, beinhaltet verschiedene Aufgaben. In Abbildung 2.1 werden einige dieser Aufgaben zweidimensional eingeordnet. Die Dimension linguistische Dimension drückt aus, ob die Syntax, die Semantik oder die Bedeutung der Sprachelemente betrachtet wird und die Dimension Level, ob die Sprachelemente auf Wort-, n-Gramm-, Satz- oder über der Satzebene hinaus betrachtet werden.

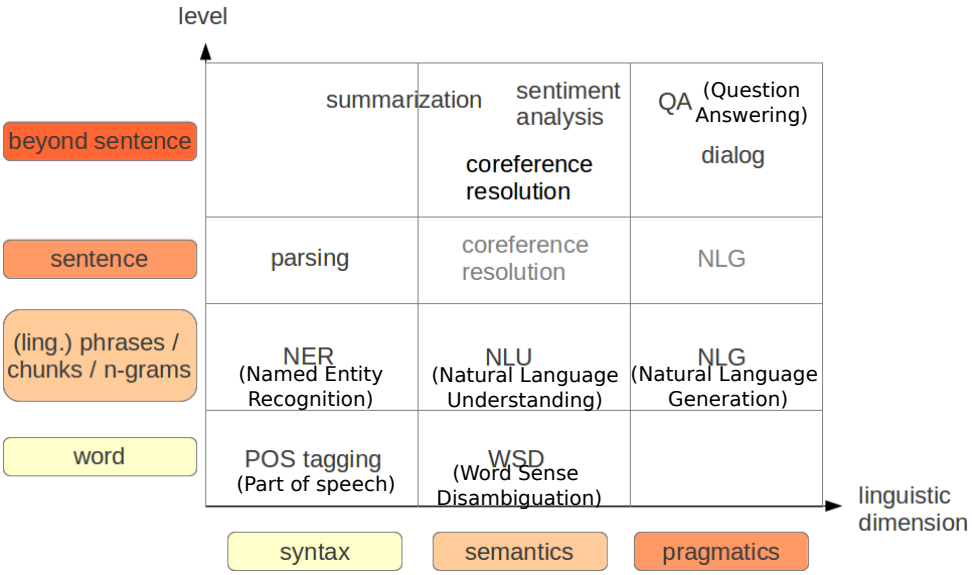

Abbildung 2.1.: Klassifikation von Aufgaben der Verarbeitung von natürlicher Sprache ((Schmidt und Niehues, 2015a) modifiziert)

Einige Aufgaben, wie z. B. die Wortsinndisambiguierung (WSD) oder die Koreferenzauflösung, können als Hilfsmittel für andere Aufgaben, wie z. B. der Dialogführung, eingesetzt werden.

Natürliche Sprachen haben sich im Laufe der Zeit entwickelt und die meisten aktiv verwendeten natürlichen Sprachen werden sich wahrscheinlich auch weiterentwickeln. Oftmals gibt es in natürlicher Sprache Mehrdeutigkeiten, wie z. B. das Wort Bank, das u. a. für ein

Kreditinstitut oder eine Sitzgelegenheit verwendet werden kann. Ein weiteres Problem für die Verarbeitung natürlicher Sprache ist der Long Tail, es gibt sehr viele Wörter, viele davon werden jedoch selten benutzt, müssen aber trotzdem verarbeitet werden können, falls sie vorkommen. Bei der Kommunikation mit natürlicher Sprache wird oft angenommen, dass der Gesprächspartner bestimmtes Wissen hat und deshalb wird die Kommunikation durch Weglassungen verkürzt. Diese Eigenschaften von natürlicher Sprache führen dazu, dass die Verarbeitung natürlicher Sprache gegenüber der Verarbeitung von Sprachen, die für die Verarbeitung an Computersystemen entwickelt wurden, wie z. B. die Programmiersprachen Java, schwieriger ist.

Die Verarbeitung natürlicher Sprache kann in regelbasierte und statistische Verfahren unterteilt werden. Wobei es auch Kombinationen von beiden Verfahren gibt.

Eine Aufgabe in der Verarbeitung natürlicher Sprache sind Dialogsysteme. Dabei können Dialogsysteme unterschieden werden, ob sie auf eine Domäne eingeschränkt werden, z. B. auf die Domäne der Restaurantreservierung, oder ob sie keine solche Einschränkung aufweisen. Ein weiteres Unterscheidungsmerkmal ist, ob alle möglichen Ausgaben fest im Dialogsystem eingespeichert sind oder ob Ausgaben vom System dynamisch erzeugt werden können.

In Abbildung 2.2 sind die verschiedenen Komponenten eines Dialogsystems abgebildet. Die Komponente automatische Automatic Speech Recognition (ASR), die gesprochene Sprache als Text ausgibt, ist nur nötig, wenn das System auf gesprochene Sprache reagieren soll und die Komponente Text-to-Speech (TTS), die Text in gesprochene Sprache ausgibt, ist nur nötig, wenn gesprochene Sprache ausgegeben werden soll.

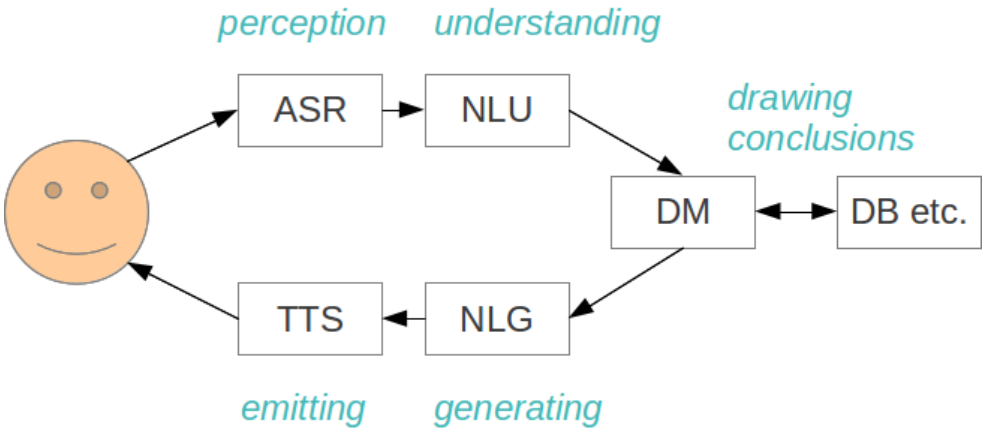

Abbildung 2.2.: Komponenten eines Dialogsystems (Schmidt und Niehues, 2015b)

Die Komponente Natural Language Understanding (NLU) hat die Aufgabe, die Eingabe zu verstehen. Die Komponente Dialog Manager (DM) erstellt aus dem verstandenen Inhalt den Inhalt für die Ausgabe, damit die Komponente Natural Language Generation (NLG) die Ausgabe in natürlicher Sprache aus dem Inhalt für die Ausgabe erzeugen kann.

Für einen detaillierten Überblick über die Grundlagen der Verarbeitung von natürlicher Sprache kann (Jurafsky und Martin, 2009) empfohlen werden.

2.2. Künstliche neuronale Netze

Künstliche neuronale Netze gehören zu den Techniken des maschinellen Lernens.
Die Grundbausteine dieser neuronalen Netze sind die künstlichen Neuronen, die miteinander gerichtet verbunden werden können. Erstmalig wurden diese Neuronen 1943 in (McCulloch und Pitts, 1943) vorgestellt. 1957 wurde im Perzeptron-Modell (Rosenblatt, 1957) das heute bekannte Neuron definiert. Dieses Neuron kann einen oder mehrere Eingänge für die Ausgabe von zu ihm hingerichteten Neuronen haben. Die Eingänge besitzen jeweils eine reelle Zahl als lernbares Gewicht und bekommen als Eingabe eine reelle Zahl. Diese Eingabewerte werden jeweils mit den dazugehörigen Gewichten multipliziert und alle Produkte werden aufsummiert. Ein Neuron kann auch einen festen Wert, den Bias, besitzen, der zu der Summe der Produkte addiert wird. Der aufsummierte Wert dient als Eingabe einer Aktivierungsfunktion, deren Ergebnis, die Ausgabe des Neurons bildet. Die Sigmoidfunktion $sig_a(x) = \frac{1}{1+e^{-ax}}$ kann z. B. als Aktivierungsfunktion eingesetzt werden. Ein Neuron ist in Abbildung 2.3 dargestellt.

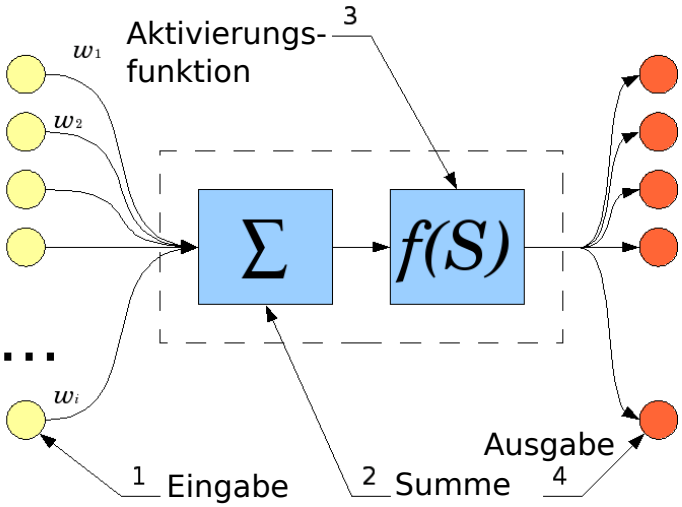

Abbildung 2.3.: Aufbau eines Neurons ((Wikimedia Commons, 2008) modifiziert)

Die Neuronen lassen sich zu unterschiedlichen Architekturen verbinden. Diese können mehrere Schichten haben. Eine Schicht ist eine Gruppierung von Neuronen. Bei großen neuronalen Netzen werden anstatt einzelnen Neuronen oftmals nur die Schichten dargestellt. Als verdeckte Schicht wird eine Schicht bezeichnet, die nicht zur Ein- bzw. Ausgabeschicht gehört. In Abbildung 2.4 ist ein neuronales Netz mit zwei verdeckten Schichten abgebildet.

Neuronale Netze mit mindestens einer verdeckten Schicht können bei Anwendung der Sigmoidfunktion als Aktivierungsfunktion und mit geeigneter Anzahl an Neuronen alle

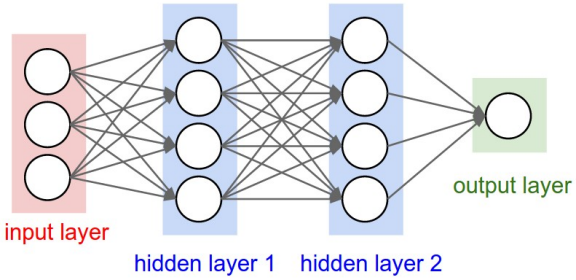

Abbildung 2.4.: ein neuronales Netz mit zwei verdeckten Schichten (Standford Vision Lab, 2015)

kompakten (stetigen und beschränkten) Funktionen lernen (Cybenko, 1989).

Ein Time-Delay Neural Network (TDNN) (Waibel u. a., 1989) ermöglicht es, zeitliche Abhängigkeiten zu verarbeiten, indem Eingaben aus n Zeitabschnitten zu einem Fenster zusammengefasst werden. Die Fenster sind gegenüber ihrem Vorgängerfenster um einen Zeitabschnitt verschoben. Die Fenster dienen dann als Eingabe für eine oder mehrere verdeckte Schichten. In der Ausgabeschicht werden alle Fenster zusammengefasst.
Zeitliche Abhängigkeiten können auch in rekurrenten neuronalen Netzen (RNNs) verarbeitet werden. Dies sind Architekturen bei denen die Verbindungen der Neuronen Zyklen bilden.
In Abbildung 2.5 werden verschiedene neuronale Netzarchitekturen dargestellt.

Ein neuronales Netz kann ein Problem zu lösen lernen, indem die Gewichte so angepasst werden, dass das neuronale Netz, die zu den Eingabewerten passenden Ausgabewerten berechnet. Dieser Vorgang wird als Training bezeichnet. Wenn keine Zwischenwerte gelernt werden, sondern direkt aus den Eingabewerten die Ausgabewerte gelernt werden, wird das neuronale Netz End-To-End trainiert.
Am Anfang werden die Gewichte entweder mit festen Werten oder mit Zufallswerten initialisiert. Beim überwachten Lernen von neuronalen Netzen werden im Training nacheinander die Daten aus dem Trainingsdatensatz, zu denen auch die gewünschten Ergebnisse bekannt sein müssen, als Eingabe für Eingabeschicht des neuronalen Netzes verwendet. Die Fehlerfunktion E vergleicht dann jeweils das Ergebnis des Netzes o mit dem gewünschten Ergebnis t. Als Fehlerfunktion können differenzierbare Funktionen, wie z. B. die Kleinste-Quadrate-Funktion $E(t,o) = \frac{1}{2}\sum_i (t_i - o_i)^2$ oder die Kreuzentropie $E(t,o) = -\sum_i t_i \cdot log(o_i)$, verwendet werden. Die Variable i ist jeweils ein anderes Element der Ausgabeneuronen. Das Ziel ist den Wert der verwendeten Fehlerfunktion zu minimieren. Der Fehler wird, abgeleitet nach den jeweiligen Gewichten, durch das neuronale Netz zurück propagiert (Backpropagationalgorithmus), um die Gewichte zu aktualisieren, sodass das Ziel der Fehlerfunktionsminimierung erreicht werden kann. Es gilt $w_{ij} = w_{ij}^{alt} + \Delta w_{ij}$ und $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$. Die Gewichte zum Ableiten sind implizit in der Fehlerfunktion in der Netzausgabe o enthalten. In (Harris, 2012) wird der Backpropagationsalgorithmus Schritt für Schritt für ein mehrschichtiges neuronales Netz durchgeführt. Der Backpropagations-

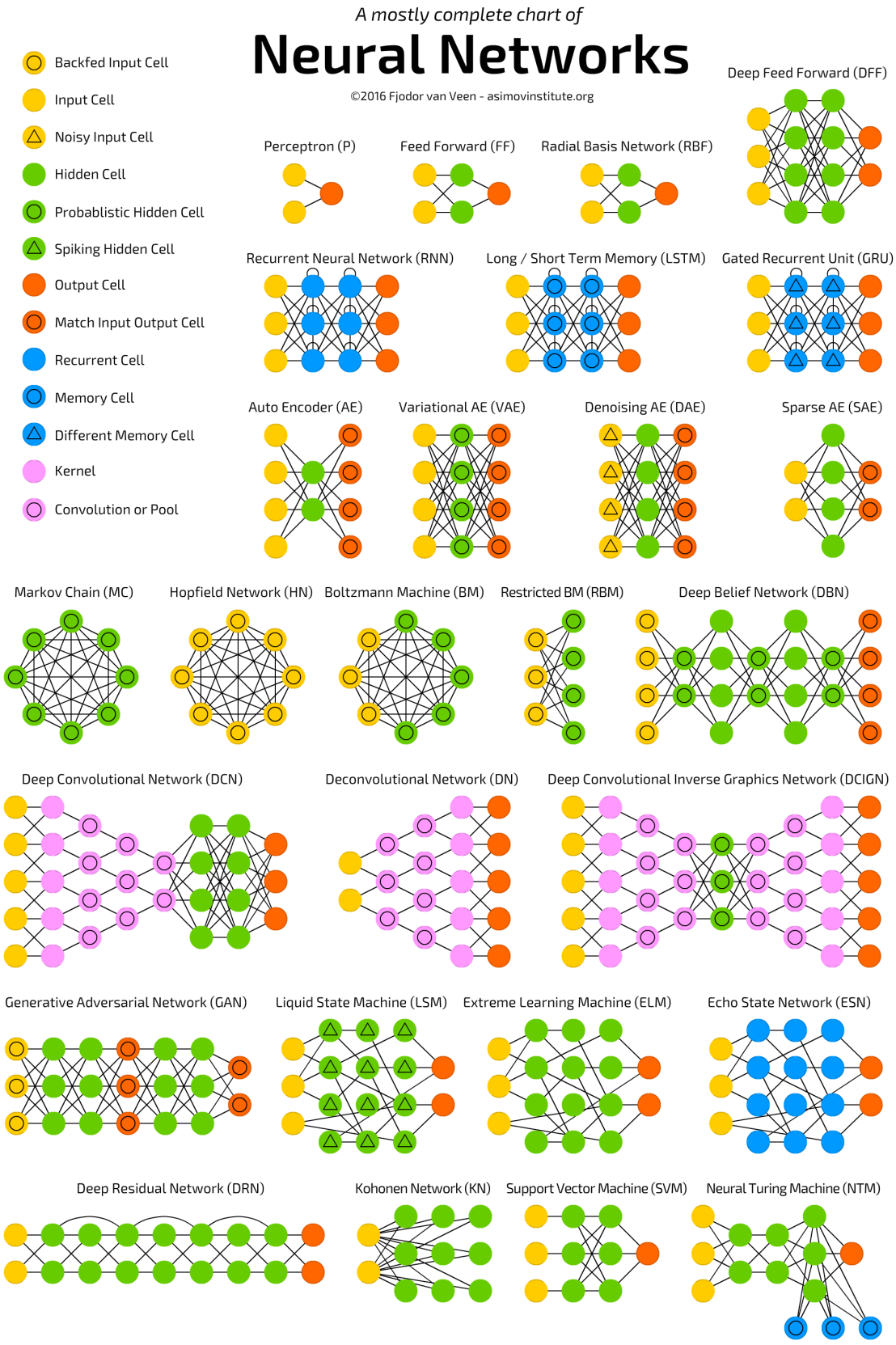

Abbildung 2.5.: verschiedene neuronale Netzarchitekturen (Veen, 2016)

algorithmus funktioniert mit allen differenzierbaren Funktionen, die Gewichte besitzen. Die Lernrate η beeinflusst das Lernverhalten. Höhere Werte für die Lernrate sorgen für größere Gewichtsanpassungen. Dies kann den Lernvorgang beschleunigen, aber auch dazu führen, dass die Anpassungen zu groß sind und es Probleme gibt, das Minimum zu finden. Es ist auch möglich die Lernrate dynamisch während dem Trainieren anzupassen, z. B. sie stetig zu verringern. Es gibt Algorithmen, wie z. B. der Adam Optimierer (Kingma und Ba, 2015), die versuchen, während dem Training die optimale Lernrate zu berechnen. Oftmals werden die Gewichte nicht nach jedem Trainingsbeispiel angepasst, sondern erst nach einer gewissen Zahl an Trainingsbeispielen. Die Anzahl an Beispielen ist die Mini-Batch.
Bei rekurrenten neuronalen Netze kann Backpropagation Through Time als Fehlerfunktionsminimierungsalgorithmus verwendet werden. Bei diesem Algorithmus wird angenommen, dass maximal k-mal die rekurrenten Kanten durchlaufen werden. Das Netz wird dann so aufgefaltet, dass es einem nicht rekurrenten neuronalen Netz entspricht - siehe Abbildung 2.6 für ein rekurrentes neuronales Netz für $k = 3$.

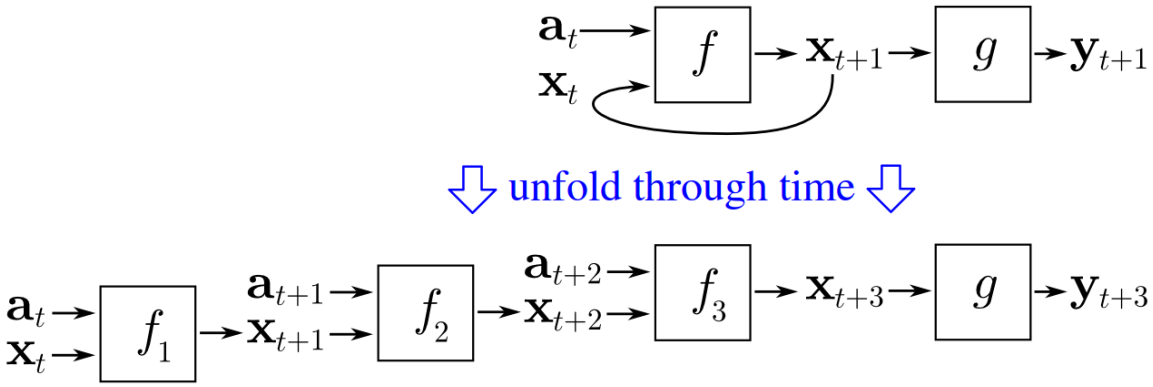

Abbildung 2.6.: Auffaltung eines rekurrenten neuronalen Netzes zur Anwendung des Backpropagation Through Time Algorithmus (Wikimedia Commons, 2010)

Die Parameter, die nicht während dem Training gelernt werden, wie z. B. das Lernverfahren und die Netzarchitektur, bilden die Hyperparameter.

Ein Validationsdatensatz mit Beispielen, die möglichst nicht im Trainingsdatensatz vorkommen, soll die Ergebnisse des Trainings validieren und eine Entscheidungshilfe sein, wann das Training abgebrochen werden kann. Der Testdatensatz mit Beispielen, die möglichst nicht im Trainings- und Testdatensatz vorkommen, soll die Qualität des Trainings evaluieren.

Ein Long short-term memory (LSTM) ist ein rekurrente neuronale Netzarchitektur. Diese Art von Netzarchitektur kann alleine oder innerhalb eines neuronalen Netzes verwendet werden. Diese Netzarchitektur wurde in (Hochreiter und Schmidhuber, 1997) erstmalig vorgestellt. Ein LSTM besitzt ein Inputgate i, ein Forget-Gate f, eine Zelle c und ein Out-

putgate o. Außerdem gibt es einen verdeckten Zustand h, der initialisiert werden muss. In (Wen, Gasic, Mrksic u. a., 2015) wird ein LSTM mit folgenden Gleichungen definiert:

$$i_t = \sigma(W_{wi}w_t + W_{hi}h_{t-1})$$
$$f_t = \sigma(W_{wf}w_t + W_{hf}h_{t-1})$$
$$o_t = \sigma(W_{wo}w_t + W_{ho}h_{t-1})$$
$$\hat{c}_t = tanh(W_{wc}w_t + W_{hc}h_{t-1})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$
$$h_t = o_t \odot tanh(c_t)$$

Der Operator $\odot$ ist hier als elementweise Multiplikation definiert und σ ist die Sigmoidfunktion.
Beim Training werden die W-Matrizen trainiert.
Ein LSTM ist in Abbildung 2.7 abgebildet.

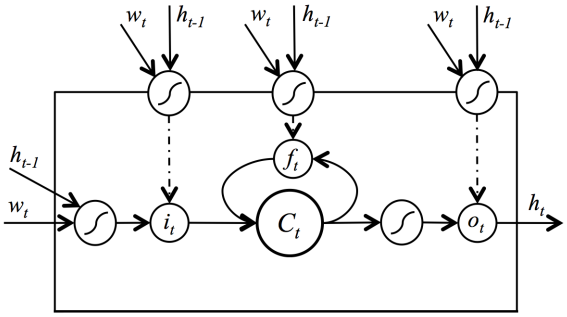

Abbildung 2.7.: Aufbau eines LSTM ((Wen, Gasic, Mrksic u. a., 2015) modifiziert)

Der Vorteil von einem LSTM gegenüber einer klassischen rekurrenten neuronalen Netzarchitektur ist, dass es nicht nur verdeckte Zustände von vorherigen Zeitschritten erhält, sondern auch Werte speichern kann. Was gespeichert und überschrieben werden soll, wird während dem Training gelernt.

Klassische LSTMs, wie in diesem Abschnitt beschrieben, wurden nicht konzipiert, um andere Werte als der vorherige verdeckte Zustand bzw. eine Eingabe zu bekommen, die nicht der LSTM-Zellengröße entspricht. In (Wen, Gasic, Mrksic u. a., 2015) wird eine Erweiterung vorgestellt, die es ermöglicht Eingabedaten zu verwenden, die nicht der LSTM-Zellengröße entspricht. Diese Art von LSTM wird semantically controlled LSTM (SC-LSTM) genannt. Diese besteht aus zwei Teilen. Einem LSTM, bei dem die LSTM-Zelle c_i als Eingabe einen Dialogakt bekommt, der von einer Dialogakt-Zelle bearbeitet wurde, siehe Abbildung 2.8. Ein Dialogakt beinhaltet den Inhalt, der in natürlicher Sprache ausgegeben werden soll. Der Dialogakt ist in (Wen, Gasic, Mrksic u. a., 2015) ein Vektor mit binären Werten.
Gegenüber einem klassischen LSTM kommen nach (Wen, Gasic, Mrksic u. a., 2015) folgende Gleichungen hinzu:

$$r_t = \sigma(W_{wr} w_t + \alpha W_{hr} h_{t-1})$$

$$d_t = r_t \odot d_{t-1}$$

Die Berechnung des Wertes der Zelle ändert sich nach (Wen, Gasic, Mrksic u. a., 2015) zu dieser Gleichung:

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t + tanh(W_{dc} d_t)$$

Dabei ist α eine Konstante.

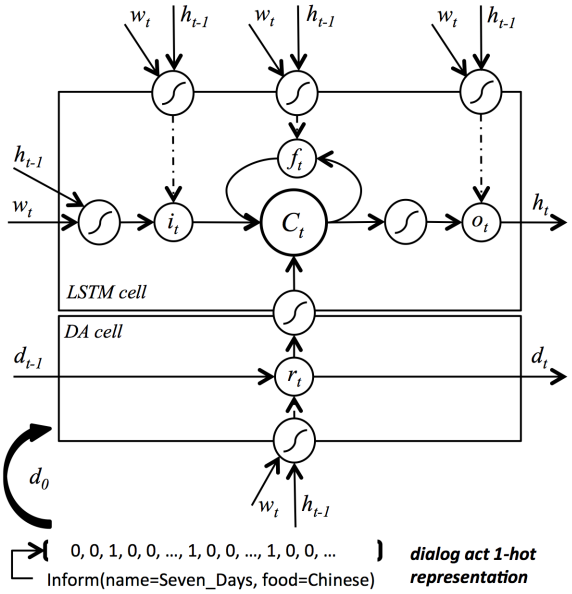

Abbildung 2.8.: Aufbau eines SC-LSTM (Wen, Gasic, Mrksic u. a., 2015)

Embeddings

Natürliche Sprache besteht aus Elementen wie z. B. Buchstaben, Phoneme, Wörter und Sätze. Es gibt zwar Ordnungen, so ordnet das Alphabet die Buchstaben, jedoch haben diese für die Bedeutung der Elemente keine Auswirkung.
Für das Verarbeiten von natürlicher Sprache ist es sinnvoll, zu bestimmen, wie ähnlich sich Sprachelemente sind. Hierfür können Embeddings verwendet werden. Satzelemente werden auf Vektoren abgebildet. Je nach Datensatz und Trainingsziel führen Wort- bzw. Äußerungs-Embeddings dazu, dass ähnliche Wörter bzw. Äußerungen eine kleinere Distanz zueinander als unähnlichere Wörter bzw. Äußerungen haben. Ähnlichkeiten in einigen Bereichen führen zu kürzen Distanzen in Dimensionen, die mit diesen Bereichen korrelieren. Ein Beispiel für mögliche Distanzen von Wort-Embeddings und Äußerungs-Embeddings befindet sich in Abbildung 2.9.

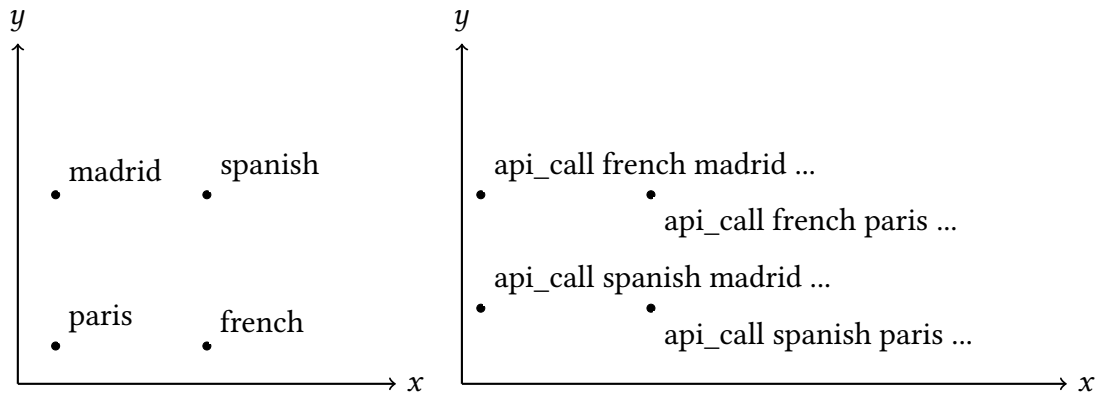

Abbildung 2.9.: Beispiele für Wort- und Äußerungs-Embeddings

Memory Networks für Dialogsysteme

Memory Networks sind eine neuronale Netzarchitektur, die auf den Speicher zugreifen kann. In (Bordes, Boureau und Weston, 2017) wird ein Memory Network beschrieben, welches den Komponenten NLU, DM und NLG eines Dialogsystems entspricht und im Folgenden vorgestellt wird.
Jeder Dialog D aus dem Datensatz wird in Teildialoge D_i aufgeteilt. Der erste Teildialog D_1 enthält die zwei Elemente erste Äußerung des Nutzers c_1^u und die dazugehörige erste Äußerung des Systems c_1^r. D_2 erweitert D_1 um die zwei Elemente zweite Äußerung des Nutzers c_2^u und die dazugehörige Äußerung des Systems c_2^r. Ein Dialog hat n Teildialoge und der n-te Teildialog entspricht dem vollständigen Dialog D. Es gilt:

$$\forall i < j : D_i \subset D_j$$

$$D_n = D$$

Beim Training und beim Führen eines realen Dialogs eines Nutzers mit dem System wird jeder Teildialog D_i als eigenständiger Dialog angesehen. D_i wird in drei Teile aufgeteilt. Der erste Teil ist der bisheriger Konversationsverlauf S_i. Es gilt $S_i = D_{i-1}$. Der zweite Teil ist die gegenüber D_{i-1} neu hinzugekommene Äußerung des Nutzers c_i^u und der dritte Teil ist die gegenüber D_{i-1} neu hinzugekommene Äußerung des Systems c_i^r.
Vor dem Training wird ein Wörterbuch mit dem Vokabular angelegt. Dieses umfasst alle Wörter, die im Datensatz vorkommen. Neben den vorkommenden Wörtern gibt es im Vokabular noch spezielle Schlüsselwörter z. B. für unbekannte Wörter und ob der Nutzer oder das System die Äußerung getroffen hat. Die Vokabulargröße wird im Folgenden mit V bezeichnet. Jedes der Wörter im Vokabular ist eindeutig einer natürlichen Zahl zugewiesen. Alle Zahlen sind dabei paarweise verschieden.
Die einzelnen Äußerungen des Konversationsverlaufs S_i werden jeweils in einem Bag-of-Words gespeichert. Die einzelnen Wörter werden dabei mit der ihnen zugewiesenen Zahl repräsentiert. Die Äußerung „may i have a table with spanish cuisine in the spanish capital madrid“ wird z. B. mit einem bestimmten Wörterbuch als {45, 30, 25, 10, 55, 65, 50, 20, 35, 60, 50, 15, 40} gespeichert. Dieses Bag-of-Words wird danach auf einen V-dimensionalen Vektor abgebildet. Jedes Wort des Vokabulars ist eindeutig auf eine Position im Vektor abgebildet.

Wenn im Bag-of-Words ein Wort vorkommt, so ist an der Position des Wortes im Vektor eine 1, ansonsten ist an dieser Position eine 0 im Vektor. Die Funktion für die Bildung des V-dimensionalen Vektors aus den Äußerungen wird im Folgenden mit Φ bezeichnet. Zusätzlich zu den Wörtern der Äußerung wird ein bestimmtes reserviertes Schlüsselwort zu dem Bag-of-Words hinzugefügt, wenn das System die Äußerung ausgegeben hat. Ein anderes bestimmtes reserviertes Schlüsselwort wird hinzugefügt, wenn der Benutzer die Äußerung getätigt hat. Außerdem gibt es Schlüsselwörter, die die Nummer der Interaktion wiedergeben und zu dem Bag-of-Words hinzugefügt werden.
Der Bag-of-Words der einzelnen Äußerungen wird auf ein d-dimensionales Äußerungs-Embedding abgebildet. Die Abbildung entspricht der Multiplikation von links mit einer trainierbaren Matrix A, die die Dimension $d \times V$ besitzt, wobei d der Größe des Embedding-Vektors entspricht.
Die folgenden Gleichungen in diesem Abschnitt wurden aus (Bordes, Boureau und Weston, 2017) entnommen. Teilweise wurden Variablennamen umbenannt, um in diese Arbeit hineinzupassen.
Die Äußerungs-Embeddings des Konversationsverlaufs S_i werden folgendermaßen als Array m im Speicher abgelegt:

$$m = (A\Phi(c_1^u), A\Phi(c_1^r)..., A\Phi(c_{t-1}^u), A\Phi(c_{t-1}^r))$$

Die letzte Äußerung des Nutzers c_t^u wird ebenfalls mit dem Anwenden der Funktion Φ auf die Äußerung und der Multiplikation von links der Matrix A zu einem Äußerungs-Embedding q gewandelt:

$$q = A\Phi(c_t^u)$$

Zu der Äußerung des Nutzers werden Äußerungen aus dem bisherigen Konversationsverlaufs herausgesucht, die für die Antwort des Systems relevant sind. Ein Beispiel in der Restaurantreservierungsdomäne ist, welche nötigen Parameter für den API-Aufruf schon bekannt sind. Die Relevanz p_i der Äußerung der i-ten Speicherstelle wird folgendermaßen berechnet:

$$p_i = Softmax(q^T m_i)$$

Die Elemente des bisherigen Konversationsverlaufes werden, mit der jeweiligen Relevanz gewichtet, auf den d-dimensionaler Vektor o abgebildet:

$$o = R \sum_i p_i m_i$$

R ist eine trainierbare $d \times d$-dimensionale Matrix.
Die Äußerung des Nutzers als Äußerungs-Embedding q dient als Eingabe für das Memory Network. Die Addition von q mit u ist die Ausgabe des Memory Networks.
Es lässt sich aber auch ein Memory Network bilden, das aus mehreren solch beschriebenen Memory Networks besteht, ein gestapeltes Memory Network. Die Anzahl an Memory Networks im Stapel werden als Anzahl an Hops bezeichnet. Die Variable h beinhaltet die Anzahl an Hops. Die Ausgabe des vorherigen Memory Network $q_2 = q + o$ bildet die Eingabe des nächsten Memory Networks.
Die Ausgabe aus den gestapelten Memory Networks ist dann:

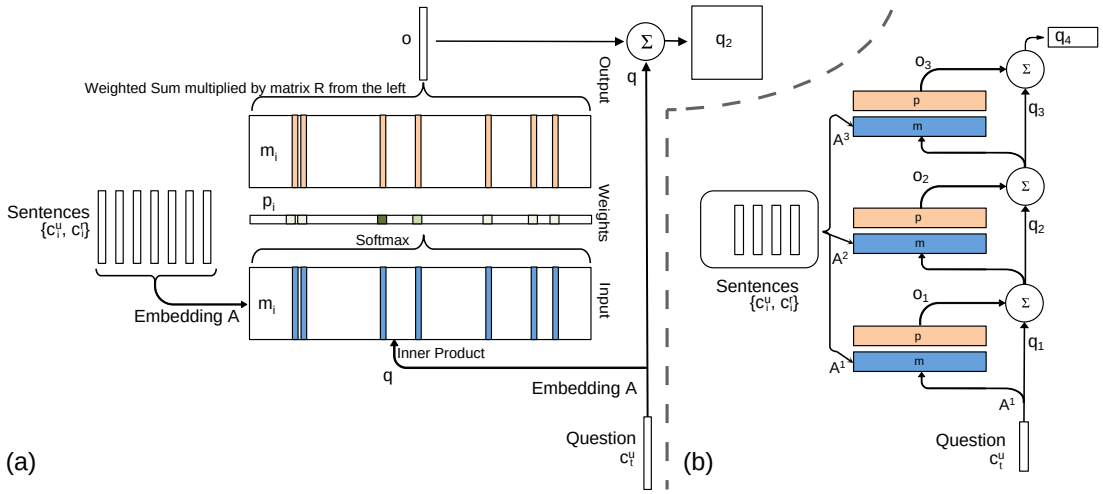

Abbildung 2.10.: das End-To-End Memory Network für die Berechnung des Eingabeparameters für die NLG-Komponente ((Sukhbaatar u. a., 2015) modifiziert)

$$q_{h+1} = q_h + o_h$$

Das Memory Network, welches q_{h+1} berechnet, entspricht den Komponenten NLU und DM eines Dialogsystems.

Zur Ausgabe der optimalen Antwort, dies entspricht der Komponente NLG eines Dialogsystems, werden alle mögliche Antwortkandidaten als V-dimensionaler Vektor gespeichert. Die Anzahl der Antwortkandidaten wird im Folgenden als C bezeichnet. Hierfür wird q_{h+1} von rechts mit einer trainierbaren $d \times V$-dimensionalen Matrix W multipliziert und dessen Ergebnis wird von links mit dem V-dimensionalen Vektor der einzelnen Antwortkandidaten multipliziert. Das Ergebnis ist ein C-dimensionaler Vektor. Nach dem Anwenden der Softmax-Funktion auf diesen Vektor, wird jedem Kandidat die Wahrscheinlichkeit zugeordnet, dass der jeweilige Kandidat der Beste ist. Der Kandidat mit der höchsten Wahrscheinlichkeit wird dann als berechnete beste Antwort $\hat{a}$ ausgewählt:

$$\hat{a} = Softmax({q_{h+1}}^T W\Phi(y_1), ..., {q_{h+1}}^T W\Phi(y_c))$$

Der Aufbau des beschriebenen Memory Networks ist in Abbildung 2.10 abgebildet.

Hyperparameteroptimierung

Die Performanz eines neuronalen Netzes hängt vor allem von diesen fünf Einflussfaktoren ab: dem Datensatz für Training und Validation, der Fehlerfunktion, des Lernverfahrens, der Netzarchitektur und der Anfangsinitialisierung der trainierbaren Strukturen. Die Optimierung dieser Hyperparameter, die während des Trainings fest sind, ermöglicht ein performanteres neuronales Netz. Die Performanz kann unter anderem sein, wie fähig ein neuronales Netz von den Trainingsdaten auf Testdaten generalisieren kann oder ob es bei

ausreichender Generalisierungsfähigkeit eine sehr schnelle Verarbeitung bietet. Es gibt eine Vielzahl von Verfahren, um die Hyperparameter zu optimieren. Im Folgenden werden ein paar Verfahren vorgestellt und deren Vor- und Nachteile genannt. Alle Verfahren haben gemeinsam, dass verschiedene Belegungen von Hyperparametern getestet werden und dass die Ergebnisse mit den getesteten Belegungen miteinander verglichen werden. Ein fairer Vergleich ist nur möglich, wenn nach der Wahl der Hyperparameter der Ablauf deterministisch ist. Bei nicht deterministischen Abläufen kann die Vergleichbarkeit zwar dadurch erhöht werden, dass bei gleicher Hyperparameterbelegung mehrere Durchläufe durchgeführt werden und dann auf die Ergebnisse der Durchläufe eine Funktion wie z. B. das Minimum, das Maximum, der Median oder das arithmetische Mittel durchgeführt wird, trotzdem ist kein fairer Vergleich mehr möglich. Das Problem des Nichtdeterminismus kommt z. B. vor, wenn Gewichte anfangs zufällig initialisiert werden.

Kartesisches Produkt

Aus allen möglichen Belegungen der einzelnen Hyperparameter wird das kartesische Produkt gebildet. Die Hyperparameterbelegungen jedes einzelnen Elements des kartesischen Produktes wird getestet.
Der Vorteil des kartesischen Produktes ist, dass zu 100 % die perfekte Hyperparameterbelegung aus den möglichen Belegungen gefunden wird, wenn das Training deterministisch ist.
Der Nachteil des kartesischen Produktes ist, dass wenn mindestens ein Parameter einen unendlichen Wertebereich besitzt, z. B. es gilt $0 < h \leq 1$ für den Hyperparameter h, das kartesische Produkt unendlich groß wird und somit die Optimierung endlos laufen würde. Bei ausschließlich endlichen Werten terminiert zwar die Optimierung, das Finden des Optimums wird aber mit einer Laufzeit von $O(\prod_{h \in Hyperparametern} \text{Anzahl an Hyperparameterwerte}(h))$ erkauft.

Grid Search

Der Nachteil des kartesischen Produktes, dass es durch Hyperparameter, die einen unendlichen Wertebereich besitzen, unendlich groß wird, wird durch das Grid Search Verfahren eliminiert. Unendliche Wertebereiche werden diskretisiert, sodass das kartesische Produkt immer endlich ist.
Der Vorteil von diesem Verfahren ist, dass es bei ausschließlich endlichen Werten von Hyperparametern eine hundertprozentige Genauigkeit bietet (dann ist es im Prinzip aber auch ein kartesisches Produkt) und bei Diskretisierung mit wenig Abstand zwischen den Werten auch eine sehr hohe Genauigkeit bietet. Ein sehr geringer Abstand wäre die minimale Differenz zwischen einer Gleitkommazahl mit einfacher Genauigkeit. Damit kommt Grid Search sehr nah an das Optimum der Hyperparameteroptimierung heran. Der Nachteil von Grid Search ist, dass es mit $O(\prod_{h \in Hyperparametern} \text{Anzahl an Hyperparameterwerte}(h))$ eine hohe Laufzeit besitzt.

Reduziertes Grid Search

Um die Laufzeit zu reduzieren, wird bei diesem Verfahren die Anzahl der möglichen Werte der Hyperparameter reduziert.
Dadurch ergibt sich als Vorteil eine geringere Laufzeit.
Die geringere Laufzeit wird, je nach Stärke der Reduktion, durch eine Hyperparameterbelegung, die gegebenenfalls nicht so nah am Optimum ist, erkauft.

Random Search

Jedem Hyperparameter wird eine Verteilung zugeordnet und festgelegt wie viele Werte aus dieser Verteilung gezogen werden sollen. Aus den gezogenen Verteilungen wird das kartesische Produkt gebildet und es wird die Performanz der Hyperparameterbelegungen dieses kartesischen Produktes getestet.
Der Vorteil dieses Verfahrens ist, dass die Laufzeit durch die Anzahl der gezogenen Werte angepasst werden kann.
Der Nachteil ist, dass aus den Verteilungen je nach Verteilungsfunktion und Anzahl der gezogenen Werte, Werte gezogen werden können, die weit vom Optimum entfernt sind.

Hand-Tuning

Bei diesem Verfahren wird eine initiale Hyperparameterbelegung festgelegt und die Performanz gemessen. Dann wird ein Parameter oder mehrere Parameter variiert und es wird wieder die Performanz gemessen. Die Variationen und wann das Ausprobieren gestoppt wird, wird intuitiv von einer oder mehreren Personen entschieden.
Dieses Verfahren kann eine geringe Laufzeit haben, vor allem bei Personen, die viel Erfahrung mit der Hyperparameteroptimierung haben, da durch eine möglichst gute Auswahl an Variationen schnell eine gute Performanz gefunden werden kann.
Durch das manuelle Auswählen der Variationen anhand der letzten Performanzergebnissen kann das Verfahren nicht automatisch durchlaufen, sondern benötigt oftmals Eingriffe.
Außerdem kann es sein, dass Parameterbelegungen, die sehr nah am Optimum sind, übersehen werden, da die optimierende Person ein mentales Modell besitzt, welches eine solche Parameterbelegung nicht in Betracht zieht.

Tree-structured Parzen Estimator

Dieser Abschnitt gibt eine Zusammenfassung der Erklärung des Tree-structured Parzen Estimators (TPE) nach (Shevchuk, 2016).
Jedem Hyperparameter wird eine Zufallsverteilung zugeordnet. Mithifle der Zufallsverteilungen werden Hyperparameterwerte gezogen und damit Hyperparameterbelegungen gebildet. Zu diesen Belegungen wird dann die Qualität, wie z. B. die Genauigkeit der Ergebnisse, berechnet. Die Hyperparameterbelegungen werden dann anhand der gemessenen Qualität in zwei Gruppen aufgeteilt. In der ersten Gruppe sind die x % besten Ergebnisse. Ein üblicher Wert für x ist nach (Shevchuk, 2016) zwischen 10 und 25. Zu der ersten Gruppe wird eine Wahrscheinlichkeitsverteilung aufgestellt, wie wahrscheinlich es ist, dass eine

Hyperparameterbelegung zu der ersten Gruppe gehört. Solch eine Wahrscheinlichkeitsverteilung wird auch für die zweite Gruppe ermittelt.
Vor jeder Iteration werden zufällig mehrere Hyperparameterbelegungen gezogen und zu jeder wird die erwartete Verbesserung (Expected Improvement (EI)) berechnet. Diese wird nach (Shevchuk, 2016) folgendermaßen gebildet:

$$EI(\text{Hyperparameterbelegung}) = \frac{P(\text{Hyperparameterbelegung in der ersten Gruppe})}{P(\text{Hyperparameterbelegung in der zweiten Gruppe})}$$

Die Hyperparameterbelegung mit der höchsten erwarteten Verbesserung, wird für die aktuelle Iteration verwendet, d. h. es wird die Performanz der Hyperparameterbelegung mit der höchsten erwarteten Verbesserung gemessen, die Hyperparameterbelegung zu einer der ersten beiden Gruppen zugeordnet und die Wahrscheinlichkeitsverteilung beider Gruppen aktualisiert. Damit ist eine Iteration abgeschlossen und es kann die nächste Iteration beginnen.
Diese Methode wird Parzen Estimator (Kerndichtenschätzer) genannt, da Wahrscheinlichkeitsverteilungen von Zufallsvariablen berechnet werden.
Tree-structured kommt daher, dass Hyperparameter untereinander Abhängigkeiten haben können, z. B. ist die Größe einer zweiten verdeckten Schicht nur nötig, wenn eine zweite verdeckte Schicht auch getestet wird. Diese Abhängigkeiten können als Baum dargestellt werden.

3. Stand der Forschung

Für das Trainieren und Testen von Dialogsysteme existieren einige öffentlich zugängliche Dialogdatensätze. Mehr als 50 existierende Datensätze werden in (Serban u. a., 2017) aufgelistet. Zu jedem Datensatz ist angegeben, ob er in schriftlicher oder gesprochener Sprache verfasst ist und wie groß der Umfang des Datensatzes ist.
In (Dodge u. a., 2016) werden verschiedene Datensätze vorgestellt und beschrieben, welche Fähigkeiten eines Dialogsystems mit diesen getestet werden können. Des Weiteren werden Ergebnisse mit diesen Datensätzen von verschiedenen Verfahren veröffentlicht.
Mit den Dialog bAbI Tasks wurden in (Bordes, Boureau und Weston, 2017) sechs Aufgaben veröffentlicht, von denen die ersten fünf Aufgaben schriftliche Dialoge sind, die synthetisch aus Mustern generiert wurden. In (Weston, Bordes u. a., 2016) wird die Intention von synthetischen Datensätze im Hinblick auf die 20 bAbI Tasks dargestellt. Diese Intention gilt vermutlich auch für die Erstellung der ersten fünf Aufgaben der Dialog bAbI Tasks. Die Intention ist, dass wenn synthetische Dialoge mit wenigen Muster nicht gelernt werden können, wird ein System auch bei realen Aufgaben höchstwahrscheinlich scheitern. Umgekehrt heißt dies, dass wenn ein System synthetische Dialoge perfekt lernen kann, es nicht unbedingt bei realen Aufgaben erfolgreich ist. Dort kommen oftmals die allgemeinen Probleme der Sprachverarbeitung wie z. B. Mehrdeutigkeiten und der Long Tail vor. Die sechste Aufgabe basiert auf der zweiten Dialog State Challenge (Henderson, Thomson und Williams, 2014), wurde aber an das Format der fünften Aufgabe der Dialog bAbI Tasks angepasst. Der Korpus der zweiten Dialog State Challenge ist in (Serban u. a., 2017) gelistet und wurde unter Zuhilfenahme des Dienstes Amazon Mechanical Turk erstellt. Dieser Dienst ermöglicht es, Aufgaben öffentlich zu stellen und von Menschen gegen ein Entgelt lösen zu lassen.

Synthetische Korpora haben den Vorteil, dass sie schnell erzeugt werden können. Der textcorpus-generator (PagesJaunes, 2017) ermöglicht es, Musteräußerungen in einer Datei zu definieren und dabei Variablen zu verwenden. Die Variablen werden mit möglichen Wörter, die in einer anderen Datei gespeichert sind, befüllt, bis eine Mindestanzahl an Äußerungen generiert ist.

In (Bordes, Boureau und Weston, 2017) wird verglichen, wie hoch die Genauigkeit von den Verfahren Supervised Embedding Models, TF-IDF Match, Nächste-Nachbarn-Klassifikation, regelbasiertes System und zwei Arten von Memory Networks bei dem Datensatz Dialog bAbI Tasks ist. Das regelbasierte System wurde gezielt für die fünf ersten Aufgaben entwickelt. Aus diesem Grund liefert das regelbasierte Verfahren mit 100 % Genauigkeit die besten Ergebnisse. Die zweitbesten Ergebnisse haben insgesamt bei den ersten fünf Aufgaben jeweils eines der beiden Memory Networks. Bei Aufgabe sechs haben die Memory Networks die höchste Genauigkeit geliefert. Memory Networks wurden in (Weston,

Chopra und Bordes, 2015) erstmalig vorgestellt und in (Sukhbaatar u. a., 2015) erweitert, sodass sie End-To-End trainiert werden können. In Abschnitt 2.2 wird die Funktionsweise von dem End-To-End lernbaren Memory Network zur Dialogmodellierung vorgestellt.
Es gibt mehrere Implementierungen von Memory Networks. Von Facebook gibt es mehrere Implementierungen in einem GitHub-Repository (Facebook, 2017a) u. a. für die 20 bAbI Tasks oder zur Verwendung von Reinforcement Learning mit Memory Networks. Das GitHub-Repository verweist auch auf Implementierungen von Entwicklern, die unabhängig von Facebook sind.

Neben den Memory Networks gibt es weitere neuronale Netze, die auf einen Speicher zurückgreifen können und damit sich für Dialogsysteme eignen, u. a. die Recurrent Entity Networks (Henaff u. a., 2017), der Differentiable Neural Computer (Graves, Wayne, Reynolds u. a., 2016), die Neural Turing Machine (Graves, Wayne und Danihelka, 2014) und Dynamic Memory Networks (Kumar u. a., 2016).

In (Wen, Gasic, Mrksic u. a., 2015) wird eine Erweiterung der LSTM, die SC-LSTM vorgestellt, welche für die Komponente der Spracherzeugung (NLP) eines Dialogsystems prädestiniert ist. Die SC-LSTM ermöglicht, den Inhalt, der in natürlicher Sprache ausgegeben werden soll, als binäre Parameter als Eingabe einzugeben. Die Funktionsweise und Vorteile werden in Abschnitt 2.2 vorgestellt.

In (Bergstra u. a., 2011) werden Ansätze entwickelt, um Hyperparameter systematischer zu optimieren. Hierfür werden Ansätze des maschinellen Lernens, wie z. B. Gaußprozesse und Kerndichteschätzer, verwendet, um zu berechnen, welche Hyperparameterbelegungen getestet werden müssen, um mit möglichst wenig zu testenden Belegungen auf eine gute Hyperparameteroptimierung zu kommen. Der Ansatz des Kerndichteschätzers (Parzen Estimators) wird in Abschnitt 2.2 beschrieben.

4. Dialogsystem

In diesem Kapitel wird das Modell des Dialogsystems und des Korpusgenerators vorgestellt. Des Weiteren wird auf die Implementierung des Dialogsystems, des Korpusgenerators, auf die Hilfsprogramme für die Evaluation des Dialogsystems und die Optimierung der Hyperparameter des Modells und der Implementierung eingegangen.

4.1. Dialogsystem: NLU und DM

Das Dialogsystem basiert auf dem Memory Network für Dialogsysteme aus (Bordes, Boureau und Weston, 2017) beschrieben in Abschnitt 2.2.
Die Komponenten NLU und DM wurden nur minimal verändert. So wird eine trainierbare $d \times d$-dimensionalen Matrix H verwendet, um q_{h+1} folgendermaßen nach (Sukhbaatar u. a., 2015) zu berechnen:

$$q_{h+1} = Hq_h + o_h$$

Um die Reihenfolge der einzelnen Wörter in den Äußerungen beizubehalten, wird, wie in (Sukhbaatar u. a., 2015) vorgestellt, ein Positionsencoding vorgenommen. Die Speicherstellen m_i werden hierfür nach (Sukhbaatar u. a., 2015) folgendermaßen berechnet:

$$m_i = \sum_j l_j \cdot Ax_{ij}, \; l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$$

Dabei ist J die maximale Satzlänge, k der Index des betrachteten Elements des Embedding-Vektors, j der Index des betrachteten Wortes. Diese Encodierung hat auch den Vorteil, dass encodiert wird, wenn Wörter mehrmals vorkommen.
Die zeitliche Reihenfolge der Äußerungen im Konversationsverlauf wird, wie in (Bordes, Boureau und Weston, 2017) beschrieben, direkt in die Konversation encodiert. Dafür werden insgesamt t Schlüsselwörter definiert, die nicht im Datensatz vorkommen dürfen und nur als Schlüsselwörter für die zeitliche Codierung verwendet werden dürfen. Zu den Äußerungen wird dann jeweils das Schlüsselwort hinzugefügt, welches encodiert, wie viele Äußerungen sie jeweils vor der aktuellen Äußerung getätigt wurden. Bei Dialogen, die mehr als t Äußerungen haben, wird das Schlüsselwort für den t-ten Dialog vor der aktuellen Äußerung auch für die Äußerungen davor verwendet.
Eine Alternative dazu wäre, die Vorgehensweise aus (Sukhbaatar u. a., 2015) zu verwenden. Es wird eine trainierbare Matrix $T_A \in \mathbb{R}^{\text{maximale Dialoglänge} \times d}$ verwendet, um das Äußerungs-Embedding der Speicherstellen zu berechnen. Die Berechnung von m_i mit Positionsencoding und zeitlicher Codierung sieht dann nach (Sukhbaatar u. a., 2015) folgendermaßen aus:

$$m_i = l_j \cdot \sum_j Ax_{ij} + T_A(i)$$

Diese Alternative hat jedoch den Nachteil, dass T_A eine feste Größe hat und damit nur Dialoge bis zu einer gewissen Länge unterstützt. Diese Einschränkung soll das Dialogsystem nicht haben und deshalb wird dieses Verfahren nicht im Dialogsystem eingesetzt.

4.2. Dialogsystem: NLG

Eine Antwortgenerierungsart bei der aus einer Menge von Kandidaten der beste Kandidat ausgewählt wird, wird im Dialogsystem von (Bordes, Boureau und Weston, 2017) verwendet und wird in Abschnitt 2.2 vorgestellt. Dieses Verfahren wird im Folgenden als Baseline dienen.
Nicht berücksichtigt im Modell des Dialogsystems wird die in (Bordes, Boureau und Weston, 2017) vorgestellte Matching-Technik. Diese Technik funktioniert folgendermaßen: Es werden dem Vokabular Schlüsselwörter hinzugefügt. Diese dürfen ausschließlich für die Matching-Technik verwendet werden. Mögliche Schlüsselwörter in der Restaurantreservierungsdomäne sind Schlüsselwörter, die die Attribute der Restaurants encodieren. Wenn im Dialog ein möglicher Wert eines Attributs vorkommt, werden die Äußerungen der Kandidaten, in denen der Wert des Attributes auch vorkommt, um das jeweilige Schlüsselwort erweitert. Diese Lösung wurde jedoch nicht in das Modell des Dialogsystems in dieser Arbeit übernommen, da die Zielsetzung des Dialogsystems ist, möglichst intelligent zu werden. Bei der Äußerung „i want spanish food because i hate french food“ lässt sich nach diesem Verfahren nicht zuordnen, welche Nationalität der Küche gemeint ist und die Zuweisung von Schlüsselwörtern zu falschen Kandidaten ist kontraproduktiv.

Die Kandidatenauswahl hat den Nachteil, dass erstmal die Kandidaten erstellt werden müssen. Weitere Nachteile sind, dass die Antwortkandidaten gespeichert werden müssen und dass für jede Antworterzeugung jeder Kandidat mit dem Produkt aus dem Vektor q_{h+1} und der Matrix W multipliziert werden muss. Dies ist kein Problem im synthetischen Datensatz des Restaurantsreservierungssystems aus (Bordes, Boureau und Weston, 2017), da es dort nur 1 200 Restaurants gibt. Jedoch existieren alleine in Deutschland über 70 000 Restaurants (Destatis, 2017).

Für die Komponente NLG werden in diesem Abschnitt weitere Verfahren vorgestellt, die die Nachteile der Kandidatenauswahl beseitigen sollen. Diese Verfahren haben jeweils eine andere Vorgehensweise und in der Evaluation in Kapitel 5 wird analysiert, welche Vorgehensweise im tatsächlichen Betrieb die besten Ergebnisse bietet und nicht nur in der Theorie gut zu funktionieren scheint.

4.2.1. Antwortgenerierung Wort für Wort mit einem klassischen RNN

Die erste Alternative ist die Antworterzeugung Wort für Wort mit einem klassischen RNN. Die Ausgabe des Memory Networks q_{h+1} stellt zusammen mit den Wort-Embeddings der zuletzt ausgegebenen m-Wörter w_t, w_{t-1}, ..., w_{t-m+1} die Eingabe eines klassischen RNN dar. Für die Wörter bei denen t kleiner als 1 ist, wird ein Schlüsselwort verwendet, welches nicht im Datensatz vorkommen darf und ausschließlich als Anfangswort für das RNN

verwendet werden darf.
Die Variable m ist ein Hyperparameter. Die Größe ist vor allem in Sätzen relevant in denen Wörter mehrmals vorkommen. Falls die Größe von m 1 beträgt, kommt es z. B. in dem Satz „do you prefer cappuccino or espresso or ristretto“ zu einem Problem, da das Netz nach der Ausgabe der Wörter „or“ jeweils die gleiche Eingabe hat und so in mindestens einem Fall die Ausgabe den falschen Wert haben wird, falls nicht ein Nichtdeterminismus dies zufällig verhindert. Deshalb sollte m mindestens die Größe des längsten sich wiederholenden n-gramms in einem möglichen Satz haben.
Die Eingabe des Netzes ist voll vernetzt mit einer verdeckten Schicht, deren Neuronenanzahl ein Hyperparameter ist. Optional gibt es auch eine zweite verdeckte Schicht. Ob es eine zweite verdeckte Schicht gibt und wie viele Neuronen diese hat, ist ebenfalls ein Hyperparameter.
Die zweite verdeckte Schicht ist, wenn sie vorhanden ist, voll vernetzt mit der ersten verdeckten Schicht.
Die Ausgabeschicht besitzt so viele Neuronen wie es verschiedene Wörter im Vokabular gibt und jedes Wort ist eindeutig einem Neuron zugeordnet. Das Wort zu dem das dazugehörige Neuron den höchsten Wert aller Neuronen der Ausgabeschicht hat, ist das auszugebende Wort. Die Ausgabeschicht ist mit der zweiten verdeckten Schicht bzw. mit der ersten verdeckten Schicht, falls keine zweite verdeckte Schicht existiert, voll vernetzt mit einem Bias b verbunden.
Zu dem auszugebenden Wort w_{t+1} wird ein Wort-Embedding w'_{t+1} berechnet. Dieses wird in der Eingabe des nächsten Zeitschritts verwendet.
Diese Netzarchitektur ist in Abbildung 4.1 abgebildet.
Die Gewichte des RNNs werden zufällig gleichverteilt im Intervall von -1 bis 1 initialisiert. Ohne diese weite Spannweite von Werten liefert diese Antworterzeugungsart schlechte Ergebnisse. Ein großer Nachteil dieser Spannweite ist, dass die Ergebnisse nicht deterministisch mit einer großen Varianz sind und mehrere Durchläufe durchgeführt werden müssen, um aussagekräftige Ergebnisse zu gewinnen. In Abschnitt 5.5 sind Abbildungen von der Genauigkeit von sechs Durchläufen nach verschiedenen Epochen ausgewertet.
Das RNN wird genauso oft durchlaufen, wie die maximal mögliche Satzlänge des Dialogsystems ist. Falls der aktuelle Satz jedoch, bevor die maximale Länge erreicht ist, beendet ist, wird ein für das Satzende reserviertes Schlüsselwort, welches nicht im Datensatz oder anderweitig verwendet werden darf, ausgegeben.
Das RNN wird für das Training aufgefaltet (Backpropagation Through Time) und bildet zusammen mit dem Memory Network, welches q_{h+1} berechnet, ein neuronales Netz. Das gesamte neuronale Netz kann dabei weiterhin End-To-End trainiert werden, da die Fehlerfunktion sich nach den Gewichten sowohl des RNNs als auch des Memory Networks ableiten lässt und beim Übergang keine Funktionsanwendung stattfindet.

4.2.2. Antwortgenerierung Wort für Wort mit einem LSTM

Der Ansatz mit dem klassischen RNN hat den Nachteil, dass für die Worterzeugung nur die zuletzt ausgegebenen m-Wörter und der Wert des Memory Networks q_{h+1} bei der Berechnung der Ausgabe des nächsten Wortes zur Verfügung stehen, längere Abhängigkeiten

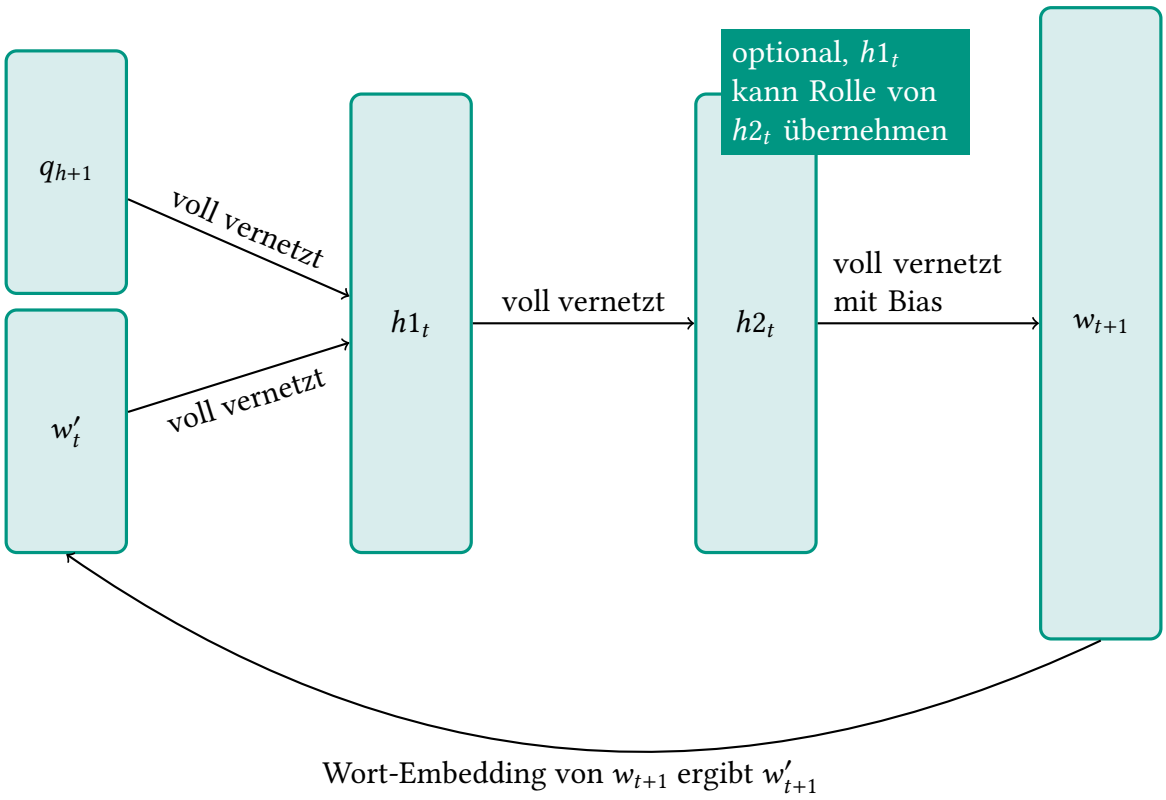

Abbildung 4.1.: NLG mit einem klassischen RNN (ein vorheriges Wort als Eingabe)

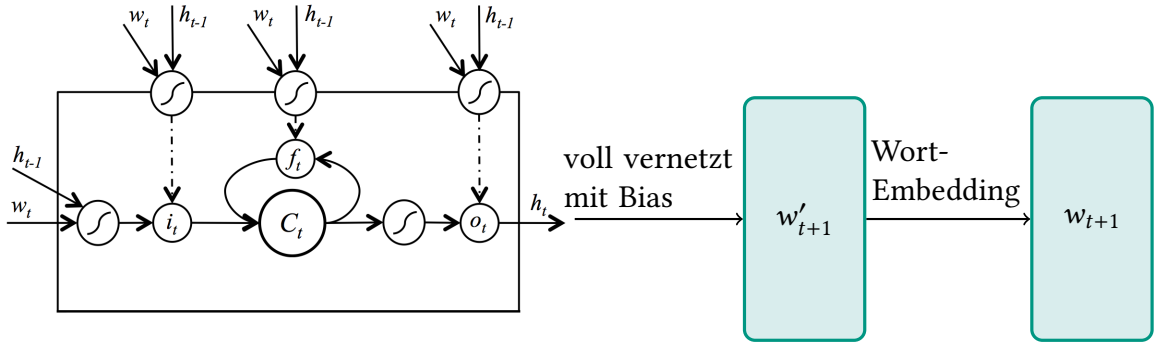

Abbildung 4.2.: NLG mit einem LSTM ((Wen, Gasic, Mrksic u. a., 2015) modifiziert)

werden nicht gespeichert. Ein LSTM ermöglicht durch seine Zelle längere Abhängigkeiten zu speichern.
Diese Fähigkeit von LSTMs macht sich die Antworterzeugung Wort für Wort mit einem LSTM zunutze. Dies ist theoretisch auf mehrere Arten möglich. Es kann die Zelle c_0 oder der verdeckte Zustand h_0 mit q_{h+1} initialisiert werden. Die Eingabe ist das zuletzt ausgegebene Wort als Wort-Embedding. Die dritte Möglichkeit ist sowohl die Zelle als auch den verdeckten Zustand mit einem Anfangszustand zu initialisieren und wie im Ansatz mit dem klassischen RNN die Ausgabe des Memory Networks q_{h+1} zusammen mit dem Wort-Embedding als Eingabe des LSTMs zu verwenden. Im Dialogsystem kann ausgewählt werden, welche Art verwendet werden soll. Zu der Ausgabe des LSTMs h_t muss das dazugehörige Wort berechnet werden. Diese Berechnung entspricht der Berechnung des Wortes aus der letzten verdeckten Schicht bei der Worterzeugung Wort für Wort mit einem klassischen RNN. In Abbildung 4.2 ist die beschriebene Architektur abgebildet.
Die Gewichte der LSTM werden genauso wie beim Ansatz mit den klassischen RNNs gleichverteilt im Intervall von -1 bis 1 initialisiert.
Da die Fehlerfunktion für alle Gewichte der Architektur ableitbar ist, kann die Worterzeugung Wort für Wort mit einem LSTM zusammen mit dem Memory Network End-To-End trainiert werden. Dabei wird das LSTM, eine Spezialisierung eines RNN, aufgefaltet. Die Auffaltung entspricht der maximalen Satzlänge.

4.2.3. Antwortgenerierung Wort für Wort mit einem SC-LSTM

Klassische LSTMs wurden nicht konzipiert, um andere Werte als den vorherigen verdeckten Zustand bzw. das zuletzt ausgegebene Wort als Eingabe zu bekommen. SC-LSTMs, vorgestellt in Abschnitt 2.2, haben zusätzlich einen weiteren Eingang für Daten. Der weitere Eingang für Daten ist ursprünglich für binäre Daten gedacht, die encodieren, was in der Ausgabe enthalten sein soll (Wen, Gasic, Mrksic u. a., 2015). Für den Ansatz Antworterzeugung Wort für Wort wird aber anstatt binäre Daten die Ausgabe des Memory Networks q_{h+1} an den Dateneingang angelegt. Der Ansatz entspricht der Antworterzeugung Wort für Wort mit einem LSTM mit dem Unterschied, dass die LSTM durch eine SC-LSTM ersetzt

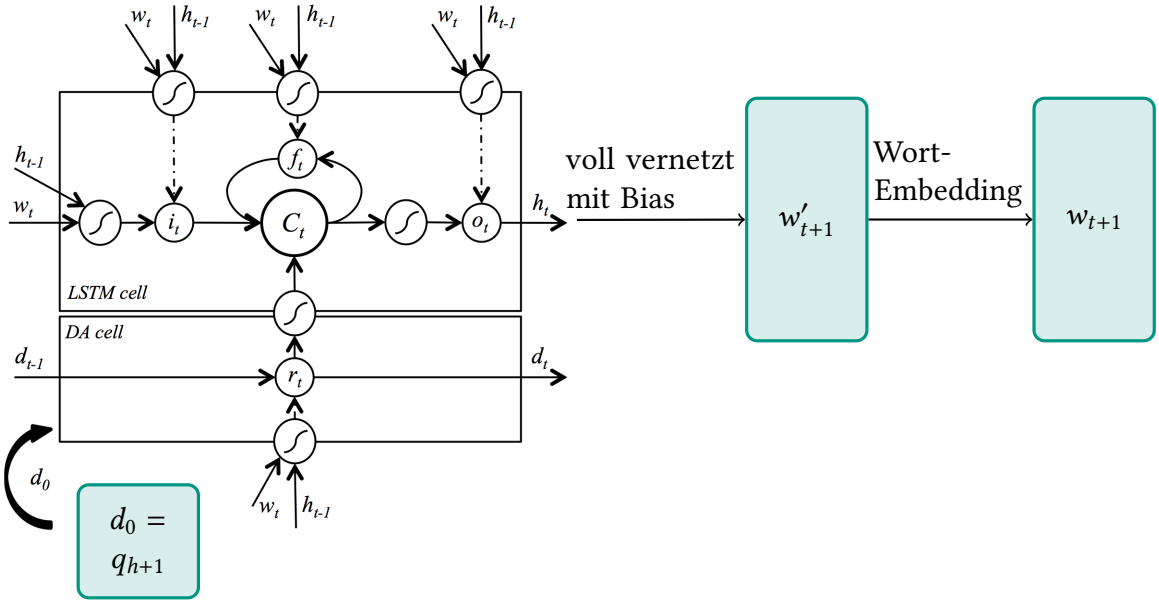

Abbildung 4.3.: NLG mit einem SC-LSTM ((Wen, Gasic, Mrksic u. a., 2015) modifiziert)

wurde. Das α lässt sich anpassen. In Abbildung 4.3 wird dieser Ansatz dargestellt.

4.3. Training des Dialogsystems

Alle Gewichte werden, wenn es in der Komponentenbeschreibung nicht anders beschrieben wurde, normalverteilt mit einem Erwartungswert von 1 und einer Standardabweichung von 0,1 initialisiert.
Um beim Training Oszillationen zu vermeiden zu können, wird eine Batch-Size von 32 für alle vorgestellten Antwortgenerierungsarten verwendet und die Datensätze können auf mehrere Arten durcheinandergewürfelt werden. Entweder die Unterdialoge werden nicht durchgewürfelt, die Unterdialoge werden zu Batches zusammengefasst und die Batches werden jedes Mal neu durchgewürfelt, alle Unterdialoge werden einmal durchgewürfelt oder die Unterdialoge werden jedes Mal neu durchgewürfelt.
Egal, ob bei der Kandidatenauswahl oder der Antworterzeugung Wort für Wort, es wird eine Klassifikation durchgeführt, da am Ende entweder ein Kandidat oder ein Wort ausgewählt wird. Als Fehlerfunktion wird die Kreuzentropie verwendet, da diese bei der Klassifikation gute Ergebnisse erzielt (McCaffrey, 2013).
Beim Training wird der Adam Optimierer verwendet, da er in einem Vergleich von verschiedenen Optimierer (Ruder, 2017) gute Ergebnisse ermöglicht hat.

4.4. Korpusgenerator

Für das Erzeugen der Robotersteuerungsaufgaben zu dem Haushaltsroboter Armar 3 wird ein synthetischer Dialoggenerator verwendet. Dieser ermöglicht eigene Muster zu definieren und diese Muster mit allen Kombinationen (kartesisches Produkt) von Attributswerten von einer beliebigen Anzahl von Attribute zu füllen. Dabei werden die Dialoge auf einen Trainings,- Validations- und Testdatensatz aufgeteilt. Um sicherzustellen, dass die Dialoge sinnvoll auf die verschiedenen Datensätze aufgeteilt werden, wird folgendes Modell verwendet: Die Attribute haben untereinander eine vollständige Ordnung. Die Elemente des kartesischen Produktes werden so geordnet ausgegeben, dass bei einem Attribut A erst der nächste Wert ausgewählt wird, wenn die Attribute, die in der Ordnung niedriger als A sind, nach dem letzten Wertewechsel von A wieder alle möglichen Werte durchlaufen haben. Nach jedem Wertewechsel des Attributs mit dem höchsten Wert in der Ordnung wird der Datensatz, in dem die Dialoge eingefügt werden, in der Reihenfolge Trainings-, Validations- und Testdatensatz gewechselt. Eine dieser Belegungen muss jedoch zu dem Trainingsdatensatz gespeichert werden, da ansonsten das Vokabular unvollständig sein könnte. Eine gute Aufteilung setzt voraus, dass in einem Musterdialog das Attribut mit dem höchsten Wert in der Ordnung vorkommen muss.
In Abschnitt 4.5 ist aufgeführt, wie mit der Implementierung des Dialoggenerators Dialoge aus selbst definierten Mustern erzeugen werden können. Dieser Generator hat den Vorteil, dass er eine Python-API bietet, die mehr Flexibilität als der existierende Korpusgenerator textcorpus-generator aus (PagesJaunes, 2017) ermöglicht. Dieser kann nur statisch Muster mit Wörtern aus Dateien befüllen.

4.5. Implementierung

Um für die Implementierung das Rad nicht neu zu erfinden, wird auf eine bestehende Implementierung aufgebaut. Es gibt mehrere Implementierungen von Memory Networks (Facebook, 2017b). Die Implementierungen aus (Facebook, 2017b) sind jedoch auf die 20 bAbI Tasks ausgelegt, die in (Weston, Bordes u. a., 2016) vorgestellt wurden und per End-To-End-trainierbarem Memory Network in (Sukhbaatar u. a., 2015) erstmalig versucht wurden zu lösen. In den 20 bAbI Tasks wird eine Geschichte erzählt und es muss ein Wort, welches in der Geschichte vorkommt, passend zu einer Frage zu der Geschichte ausgewählt werden. Im Abschnitt A.2 befinden sich hierfür Beispiele. Die Implementierung aus (Luna, 2017) wurde als Basis für die Implementierung ausgewählt. Die Implementierung ist in Python 3.5 geschrieben und verwendet Tensorflow 1.1 für die Berechnungen, die im Zusammenhang mit den neuronalen Netzen stehen. Die Version aus (Luna, 2017), die für das Lösen der 20 bAbI Tasks entwickelt wurde, wurde so angepasst, dass es dem in dieser Arbeit erstelltem Modell des Dialogsystems entspricht und damit auf Äußerungen von einem Nutzer passend antworten kann und dabei vorherige Äußerungen des Nutzers und vorherige Äußerungen des Systems berücksichtigt. Die Art der Antworterzeugung kann eingestellt werden.
Ein Training mitsamt Evaluation lässt sich durch das Ausführen der Anwendung *train.py* durchführen. Die Hyperparameterwerte, die Anzahl der Epochen, wie viele Epochen zwi-

schen den Evaluationen liegen, ob der Zustand des Dialogsystems der besten Evaluation mit dem Validationsdatensatz gespeichert werden sollen und die Pfade, wo einzelne Dateien abgerufen bzw. gespeichert werden, können per Kommandozeilenparameter gegenüber der Standardeinstellung verändert werden. In Abschnitt A.3 ist eine Auflistung aller Kommandozeilenargumente enthalten. Der Zustand des Dialogsystems umfasst die Gewichte und einige Einstellungen, unter anderem die Hyperparameterbelegungen, die Abbildung der Wörter auf die natürlichen Zahlen und die Antwortkandidaten. Die Einstellungen werden in der Datei settings.pickle gespeichert, deren Speicherort per Kommandozeilenparameter ausgewählt werden kann.
Die Implementierung wurde so angelegt, dass das Training und die Evaluation von einer anderen Anwendung, die Python-Code verwenden kann, eingebunden werden kann. Dafür muss die Klasse *Trainer* aus *train.py* in die andere Anwendung importiert werden und die gewünschten Funktionen müssen aufgerufen werden. Dies wird für die Hyperparameteroptimierung, beschrieben in Abschnitt 4.6, gemacht.
Neben dem Training und dem Speichern der besten Trainingsdurchläufe, ermöglicht die Anwendung *train.py*, zu einem gespeicherten Zustand, die Genauigkeit für den Trainings-, Validations- und die Testdatensätze zu bestimmen.
Die Genauigkeit lässt sich anzeigen in dem Maß, wie viele Dialoge komplett richtig beantwortet wurden und wie viele Äußerungen komplett richtig beantwortet wurden. Für die weitere Analyse können auch die fehlerhaften kompletten Dialoge ausgegeben werden.
Dialoge werden in folgender Form dargestellt: Die Zeilen eines Dialoges sind bei 1 anfangend durchnummeriert. Eine Leerzeile trennt zwei Dialoge voneinander. Eine Zeile umfasst eine Äußerung eines Nutzers und eine Äußerung des Systems. Beide Äußerungen sind mit einem Tab voneinander getrennt. Zeilen in einem Dialog ohne Tabzeichen sind für Systemausgaben z. B. nach einem API-Aufruf reserviert. Vermutlich sollte diese Form die meisten Domänen abdecken können.

Ein Dialogsystem ergibt nur Sinn, wenn mit ihm richtige Dialoge geführt werden können. Deshalb wurde für diese Arbeit eine Webanwendung erstellt. Diese bietet eine Oberfläche, in der ein Nutzer eine Äußerung an das Dialogsystem senden kann und vom System eine Äußerung als Antwort bekommt. Die Eingaben des Nutzers werden dabei in einer Datenbank gespeichert und mit einem Cookie über einen Schlüssel miteinander verbunden, sodass der Nutzer jederzeit den Dialog unterbrechen und wieder fortführen kann, falls das Cookie nicht gelöscht wird oder ausläuft. In der Implementierung sind die Cookies zwei Wochen gültig und die Gültigkeit wird wieder auf zwei Wochen zurückgesetzt, wenn eine neue Äußerung getätigt wird. Alternativ lässt sich der bisher geführte Dialog auch löschen, um einen neuen Dialog anfangen zu können.
In Abbildung 4.4 ist die Oberfläche der Webanwendung abgebildet.
Die Webanwendung wurde in Python 3.5 geschrieben und verwendet Django 1.10 als Webframework. Dabei wird, soweit wie möglich, auf den Quellcode der Trainingsanwendung zurückgegriffen, um die gleichen Antworten wie bei der Evaluation mit Dialogen, gespeichert in Textdateien, zu bekommen und um den Wartungsaufwand beider Anwendungen gering zu halten.
Die Webanwendung ermöglicht, die Art der Antworterzeugung jederzeit zu wechseln. Dabei können nur Antworterzeugungsarten ausgewählt werden, für die die korrekte

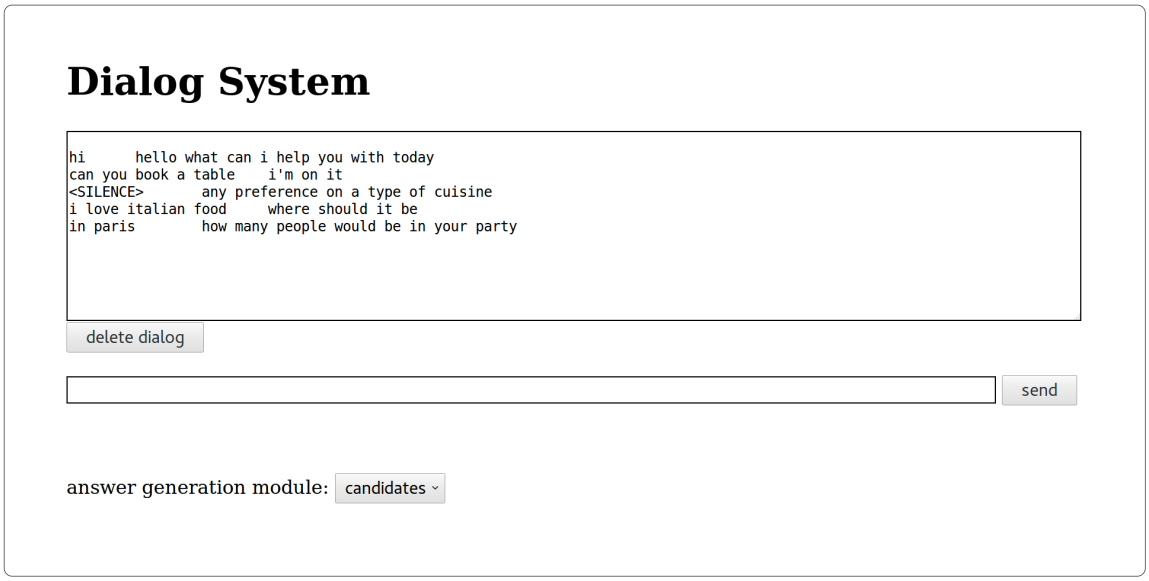

Abbildung 4.4.: Oberfläche der Webanwendung des Dialogsystems

Gewichte erfolgreich geladen werden konnten. Um die Gewichte laden zu können, ist es notwendig, dass alle Elemente des Dialogsystems, die für das Laden der Gewichte notwendig sind, die gleichen Einstellungen haben, die auch beim Training verwendet wurden. Dies wird erreicht, indem die passende Einstellungsdatei *settings.pickle* geladen wird. Ohne diese Einschränkung, dass alle Zustände die gleichen Einstellungen haben, würde das Trainingsprogramm komplexer werden. Falls die Webanwendung in Zukunft in den produktiven Einsatz gehen würde, würde sich ohnehin für die bestmögliche Antworterzeugungsart entschieden werden, da dem Nutzer keine Evaluation von den verschiedenen Antworterzeugungsarten zugemutet werden könnte. Als Workaround um Antworterzeugungsarten mit verschiedenen Einstellungen zu verwenden, können mehrere Instanzen der Webanwendung gestartet werden und jeder Instanz wird per Kommandozeilenparameter der Ort der Gewichte und der Einstellungsdatei mitgegeben und erhält entweder einen anderen ungenutzten Port oder eine andere Adresse mit einem ungenutzten Port.
Die Webanwendung wird mit dem Befehl *python3 manage.py runserver --noreload* gestartet. Die Option *noreload* verhindert das doppelte Laden des Netzes. Optional kann der Ort der Einstellungsdatei und der Gewichte per Argument *--backendDir* mitgegeben werden. Mit dieser Kommandozeilenoption ist es notwendig, dass die Kommandozeilenoption *noreload* verwendet wird. Alternativ kann auch per Django-Einstellungsdatei *settings.py* die Variable mitgegeben werden. Der Wert in der Einstellungsdatei *settings.py*, falls vorhanden, wird gegenüber dem Wert des Kommandozeilenparameters präferiert. Als weiterer Parameter kann optional auch die IP-Adresse mit Doppelpunkt getrennt als positionelles Kommondozeilenargument mitgegeben werden. Ein Kommando mit beiden optionalen Argumenten ist: *python3 manage.py runserver --noreload --backendDir PATH 192.178.21:80.*

Der Dialoggenerator besteht aus der Python-Klasse *DialogGenerator*, mit der Dialoge aus selbst definierten Mustern, gefüllt mit allen Kombinationen aus Attributswerte, er-

```
from collections import OrderedDict
from corpus_generator import DialogsGenerator


def gebePersonSpieltMusterZurueck(dictNameValue, dictNameValues, extraData):
    return '{} spielt {}'.format(dictNameValue['PERSON'], dictNameValue['SPIEL'])

robotDialogsGenerator = DialogsGenerator()
dictNameValues = OrderedDict([('PERSON', ['Eunah', 'Jan', 'Stefan']), ('SPIEL', ['
    Badminton', 'Handball'])])
robotDialogsGenerator.generateDialogsFromPatterns(dictNameValues,
    gebePersonSpieltMusterZurueck)
robotDialogsGenerator.writeDialogs('example')
```

Abbildung 4.5.: Beispiel für die Erzeugung von Dialoge mithilfe des Dialoggenerators

zeugt werden können. Hierfür muss eine beliebige Instanz aus dieser Klasse instanziiert werden.
Die Funktion *generateDialogsFromPatterns(self, dictNameValues, *patternFcts, extraData= None)* einer Instanz von *DialogGenerator* erzeugt mithilfe von Musterfunktionen Dialoge. Im Parameter *dictNameValues* wird ein *collections.OrderedDictionary* übergeben. Dieses ordnet jedem Attributsnamen eine Liste von möglichen Attributswerten zu. Dabei ist dieses Dictionary absteigend nach der Wertigkeit der Attribute, die Wertigkeit der Attribute wurde in Abschnitt 4.4 definiert, geordnet. Das optionale Dictionary *extraData* besitzt keine festgelegte Struktur und kann zur Datenübergabe an die Musterfunktionen verwendet werden. Die Variable **patternFcts* steht für beliebige Musterfunktionen, die übergeben werden. Diese Musterfunktionen werden bei der Ausführung von *generateDialogsFromPatterns* aufgerufen und die Parameter *dictNameValue*, *dictNameValues* und *extraData* werden übergeben. Der Parameter *dictNameValue* entspricht einem Dictionary, bei dem zu den Attributsnamen jeweils der Wert des aktuell betrachteten Elements des kartesischen Produktes ausgegeben wird. Die Parameter *dictNameValues* und *extraData* entsprechen den Parametern, die der Funktion *generateDialogsFromPatterns* übergeben wurden. Eine Musterfunktion hat die Aufgabe einen String zurückzugeben. Zur Erzeugung dieses Strings kann auf die übergebenen Parametern zurückgegriffen werden. Der zurückgegebene String wird dann dem aktuell zu erzeugende Dialog übergeben. Ein Dialog ist erzeugt, wenn alle übergebenen Musterfunktionen aufgerufen wurden. Für jede mögliche Belegung von Attributen wird ein Dialog erzeugt. Die Dialoge werden im Arbeitsspeicher gespeichert. Mit der Methode *writeDialog* können dann die Dialoge im Arbeitsspeicher auf die Dateien *PATH_train.txt*, *PATH_val.txt* und *PATH_test.txt* aufgeteilt werden. *PATH* ist ein beliebiger Pfad, der der Funktion *writeDialog* übergeben wird.
In Abbildung 4.5 ist eine Beispielanwendung des Dialoggenerators abgebildet. Eine größere Anwendung wurde für die Erzeugung der Robotersteuerungsaufgaben für diese Arbeit entwickelt und ist die Anwendung *util/robot_corpus_generator.py*.

Neben der Trainingsanwendung, der Webanwendung und des Dialoggenerators wurden auch einige Hilfsanwendungen entwickelt. Diese dienen zur Visualisierung von Evaluationsergebnissen, der Bewertung der Datensätze und der Erzeugung bzw. Veränderung von Datensätzen. Eine genaue Auflistung der Hilfsanwendungen ist in Abschnitt A.4.

4.6. Hyperparameteroptimierung

In diesem Abschnitt wird die Hyperparameteroptimierung des Dialogsystems gezeigt. Dabei wird dargestellt, wie verschiedene Hyperparameterwerte die Genauigkeit beeinflussen. Die Hyperparameteroptimierung wird für die Dialog bAbI Tasks durchgeführt.
Das Ziel ist eine Hyperparameterbelegung zu finden, die gute Ergebnisse ermöglicht. Aufgrund einer nicht unbegrenzt zur Verfügung stehenden Rechenleistung und des Nichtdeterminismus ist aber nicht das Ziel, zu beweisen, dass dies wirklich die beste Hyperparameterbelegung ist. Bei den Dialog bAbI Tasks kann anhand der Resultate einer nicht veröffentlichten Referenzimplementierung (Bordes, Boureau und Weston, 2017) beurteilt werden, ob die Ergebnisse gut sind.

4.6.1. Vorgehen Hyperparameteroptimierung

Die bisherige übliche Hyperparameteroptimierung wird von (Bergstra u. a., 2011) als „more of an art than a science“ beschrieben. (Bergstra u. a., 2011) zeigt, deshalb mehrere Ansätze aus dem Bereich des maschinellen Lernens, um eine systematische Hyperparameteroptimierung durchführen zu können.
Die Hyperparameteroptimierungsverfahren Random Search und Tree-structured Parzen Estimator (TPE) sind in der BSD-lizenzierten Python-Bibliothek Hyperopt (Bergstra, 2017) implementiert (Bergstra, 2013). Das Verfahren TPE erzielt in (Bergstra u. a., 2011) gute Ergebnisse.
Für die Hyperparameteroptimierung wird TPE mit 100 Iterationen durchlaufen, um Hyperparameter zu finden, die gute Resultate ermöglichen. Dabei wird jede Hyperparameterbelegung viermal durchlaufen und als Ergebnis wird der Median gewählt. Dadurch sollen Ausreißer bedingt durch den Nichtdeterminismus verringert werden.
Aus den besten Ergebnissen wird das beste Ergebnis ausgewählt, das sich reproduzieren lässt.
Nach der Anwendung des TPE werden die Hyperparameter getrennt voneinander optimiert. Im Folgenden wird diese Hyperparameteroptimierung voneinander getrennte Hyperparameteroptimierung genannt. Zu jeweils einem Hyperparameter werden hierfür verschiedene Werte getestet, die anderen Hyperparameter behalten während diesen Tests fest ihren Wert für den sich nach Ausführung des TPEs entschieden wurde. Die Anzahl der Durchläufe soll sich an der Stabilität der Ergebnisse bei der Hyperparameteroptimierung mit dem TPE orientieren.
Die zu testende Werte sind jeweils äquidistant in einem Intervall. Es werden vier Werte getestet. Der beste getestete Wert und der Wert des linken bzw. rechten Nachbarwertes, je nach dem welcher Nachbar ein besseres Ergebnis geliefert hat, bilden ein Intervall für das wieder vier äquidistante Werte gewählt und getestet werden. Die Hyperparameter, die nicht gerade getestet werden, besitzen den Wert der Hyperparameterbelegung, die nach der TPE-Hyperparameteroptimierung ausgewählt wurde.
Nach dem vollständigen Testen wird zu jedem Hyperparameter der Wert mit dem Ergebnis ausgewählt. Die Hyperparameter werden dann absteigend nach ihrem besten Ergebnis sortiert. Jeder dieser Hyperparameter wird dann, wie oben beschrieben, in der sortierten Reihenfolge nochmal mit verschiedenen Werten getestet. Die ersten vier zu testende Werte

sind die gleichen wie im unsortierten Durchlauf, die zweiten vier zu testende Werte eines Hyperparameters richten sich wieder danach, in welchem Bereich die besten Ergebnisse erzielt werden konnten. Nach dem Erhalt der Ergebnisse aller Werte eines Hyperparameters in dem sortierten Durchlauf wird der Hyperparameterwert dieses Hyperparameters mit dem besten Ergebnis ausgewählt und beim Testen der nächsten Hyperparameter verwendet. Der erste Hyperparameter in der Sortierung braucht natürlich nicht nochmal getestet werden, da für ihn die gleichen Rahmenbedingungen wie im unsortierten Durchlauf bestehen.
Das Verwenden der voneinander getrennten Hyperparameteroptimierung hat den Vorteil, dass wenn die Hyperparameter nicht oder nur wenig korrelieren, mit wenig Aufwand eine gute Lösung gefunden wird. Falls jedoch die Hyperparameter stark korrelieren, werden die Ergebnisse der voneinander getrennten Hyperparameteroptimierung keine oder nur eine kleine Verbesserung gegenüber dem TPE ergeben. Falls keine Verbesserung gegenüber der TPE-Hyperparameteroptimierung erreicht wird, kann auf dessen Ergebnis zurückgegriffen werden.
Dieses Vorgehen stellt so einen Kompromiss zwischen Laufzeit und dem Finden einer guten Lösung dar.

4.6.2. Hyperparameteroptimierung Dialog bAbI Tasks

Die Hyperparameteroptimierung wird, wie in Unterabschnitt 4.6.1 beschrieben, für Aufgabe 5 der Dialog bAbI Tasks durchgeführt. Die Dialog bAbI Tasks werden ausführlicher in Abschnitt 5.1 vorgestellt. Es wurde Aufgabe 5 für die Hyperparameteroptimierung gewählt, da Aufgabe 5 die Muster der ersten vier Aufgaben enthält und die Ergebnisse in ein paar Testdurchläufen, im Gegensatz zu den Ergebnissen von Aufgabe 6, vielversprechend waren. Bei diesen Testdurchläufen war, unter Beachtung der Ergebnisse der Referenz (Bordes, Boureau und Weston, 2017), jedoch noch Potential vorhanden. Das Dialogsystem aus der Referenz schneidet bei Aufgabe 6 mit 44,1 % Genauigkeit bei den Antworten und 0 % Genauigkeit, wenn die Genauigkeit der Beantwortung kompletter Dialoge betrachtet wird, schlechter als bei Aufgabe 5 ab, bei der eine Genauigkeit von 96,1 % bei den Antworten und 49,4 % bei der Führung der kompletten Dialogen erreicht wird. Gründe für dieses schlechte Abschneiden bei Aufgabe 6 werden in Abschnitt 5.1 gegeben.

In Abbildung 4.6 sind die Intervalle, die für die Hyperparameterwerte für die Hyperparameteroptimierung mit dem TPE gewählt wurden, dargestellt. Die Intervalle wurden intuitiv anhand einiger Testdurchläufe festgelegt.
Bei allen Hyperparameteroptimierungsarten wurden 100 Epochen trainiert und jede vierte Epoche wurde bei der Antworterzeugung mit der Kandidatenauswahl und jede Epoche wurde bei der Antworterzeugung Wort für Wort mit dem klassischen RNN evaluiert. Dabei wurde für die Ermittlung des Fehlers der Validationsdatensatz verwendet.

Die Ergebnisse der Hyperoptimierung mit dem TPE sind in Abbildung 4.8 für die Antworterzeugung mit der Kandidatenauswahl und in Abbildung 4.11 für die Antworterzeugung Wort für Wort mit dem klassischen RNN dargestellt. In beiden Abbildungen sind die Fehler der einzelnen Hyperparameterbelegungen jeweils auf einem Zahlenstrahl abgebildet.

Hyperparameter	Verteilung	Intervall
Lernrate	Gleichverteilung	$x \in [0{,}001; 0{,}01]$
Hops	ganzzahlige Gleichverteilung	$x \in [1; 7]$
Embedding-Größe	ganzzahlige Gleichverteilung	$x \in [1; 150]$

zusätzlich für die Antwortgenerierung Wort für Wort mit dem klassischen RNN:

Hyperparameter	Verteilung	Intervall
erste verdeckte Schichtgröße	ganzzahlige Gleichverteilung	$x \in [1; 150]$
optionale zweite verdeckte Schichtgröße	ganzzahlige Gleichverteilung	$x \in [1; 150]$

Abbildung 4.6.: zu optimierende Hyperparameter bei der Hyperparameteroptimierung mit dem TPE

Dabei sind die Abstände proportional zum Unterschied des Fehlers. Von dort aus geht jeweils ein Pfeil zu dem dazugehörigen Fehler und der dazugehörigen Hyperparameterbelegung. Die Hyperparameter sind die Lernrate des Adam Optimierers, die Größe des Embeddings und die Anzahl an Hops. Bei der Antworterzeugung mit einem klassischen RNN gibt es zusätzlich diese Parameter: die Größe der ersten verdeckten Schicht und die Größe der zweiten verdeckten Schicht - None, wenn diese nicht existiert. Der Fehler und die Hyperparameterbelegungen sind Semikolon-separiert in der Reihenfolge in der die Hyperparameter in diesem Abschnitt vorgestellt wurden.

Bei der Antworterzeugung mit der Kandidatenauswahl hat sich gezeigt, dass die Werte der einzelnen Durchläufe mit der gleichen Hyperparameterbelegung relativ wenig Genauigkeitsunterschied haben. Dies beruht darauf, dass der Zufall in der Initialisierung der Gewichte in der NLG-Komponente, beschrieben in Abschnitt 4.2, relativ gering ist. Bei den Belegungen, die gute maximale Ergebnisse - mindestens 92 % Genauigkeit - erzielt haben, betrug der Unterschied zwischen maximalen und minimalen Ergebnis der vier Durchläufe maximal 4,1 %. Da die beste Hyperparameterbelegung ermittelt durch den TPE reproduzierbar ist, wurde diese ausgewählt. Diese Hyperparameterbelegung beinhaltet folgende Werte: 0,0093 für die Lernrate des Adam Optimierers, 38 für die Embedding-Größe und 1 für die Anzahl an Hops. Da die Abweichung zwischen den einzelnen Durchläufen relativ gering ist, wurde für die voneinander getrennte Hyperparameteroptimierung drei Durchläufe verwendet. Die Intervalle für diese Optimierung sind in Abbildung 4.7 abgebildet. Die Ergebnisse der voneinander getrennten Hyperparameteroptimierung sind in Abbildung 4.9 und Abbildung 4.10 abgebildet. Das Ergebnis der voneinander getrennten Hyperparameteroptimierung ist, dass die beste Belegung folgende Werte beinhaltet: 0,0058 für die Lernrate des Adam Optimierers, 44 für die Embedding-Größe und 1 für die Anzahl an Hops.

Im Gegensatz zur Antworterzeugung mit der Kandidatenauswahl wurden mit dem Dialogsystem mit der Antworterzeugung Wort für Wort mit dem klassischem RNN größere Genauigkeitsunterschiede zwischen den einzelnen Durchläufen mit der gleichen Hy-

perparameterbelegung erzeugt. Dies ist darauf zurückzuführen, dass der Zufall in der Initialisierung der Gewichte des RNNs, beschrieben in Abschnitt 4.2, größer ist. Für die voneinander getrennte Hyperparameteroptimierung wird folgende Belegung ausgewählt: Lernrate des Adam Optimierers 0,002, Embedding-Größe 56, Hops 9, erste verdeckte Schicht 50 und keine zweite verdeckte Schicht. Diese Belegung hat sich als reproduzierbarer als die beste Belegung der Hyperparameteroptimierung des TPEs gezeigt. Wegen dem großen Genauigkeitsunterschied werden acht Durchläufe für die voneinander getrennte Hyperparameteroptimierung verwendet. Die Intervalle für diese Optimierung sind in Abbildung 4.7 abgebildet. In Abbildung 4.12 und Abbildung 4.13 sind die Ergebnisse der voneinander getrennten Hyperparameteroptimierung abgebildet. Als beste Belegung wurde folgende Belegung ermittelt: Lernrate für den Adam Optimierer 0,0022, Embedding-Größe 59, Anzahl an Hops 3 und Neuronenzahl erste verdeckte Schicht 56. Insgesamt sind die Ergebnisse bei der Antworterzeugung Wort für Wort mit dem klassischen RNN schlecht reproduzierbar. Eine ausführlichere Untersuchung davon befindet sich in Abschnitt 5.5.

Hyperparameter	Verteilung	Intervall
Lernrate	Gleichverteilung	$x \in [0{,}001; 0{,}01]$
Hops	ganzzahlige Gleichverteilung	$x \in [1; 4]$
Embedding-Größe	ganzzahlige Gleichverteilung	$x \in [35; 80]$

zusätzlich für die Antwortgenerierung Wort für Wort mit einem klassischen RNN:

Hyperparameter	Verteilung	Intervall
erste verdeckte Schichtgröße	ganzzahlige Gleichverteilung	$x \in [20; 80]$

Abbildung 4.7.: zu optimierende Hyperparameter bei der voneinander getrennten Hyperparameteroptimierung

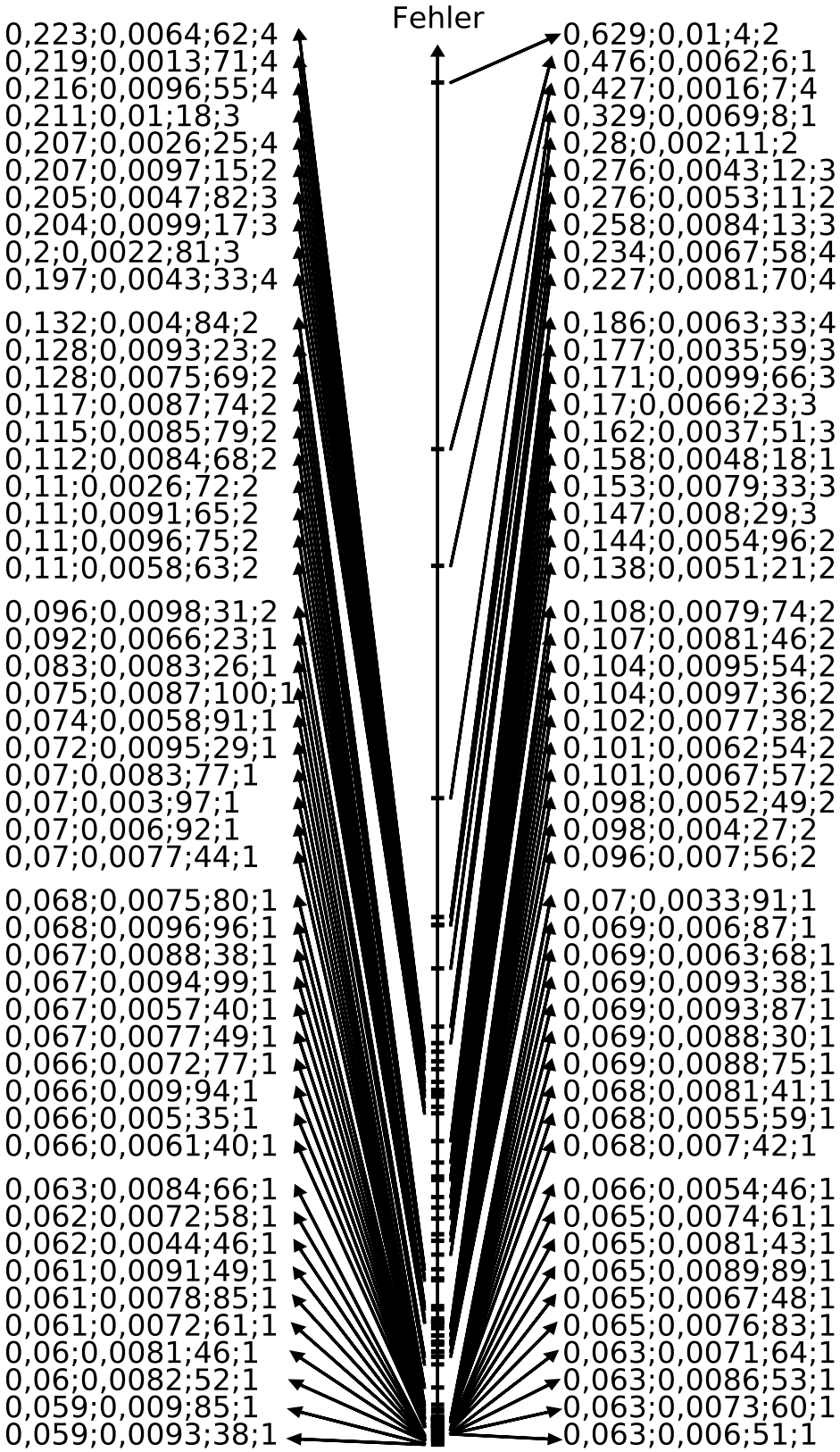

Abbildung 4.8.: Ergebnisse der TPE-Hyperparameteroptimierung mit der NLG-Komponente Kandidatenauswahl (Fehler;Lernrate;Embedding-Größe;Hops)

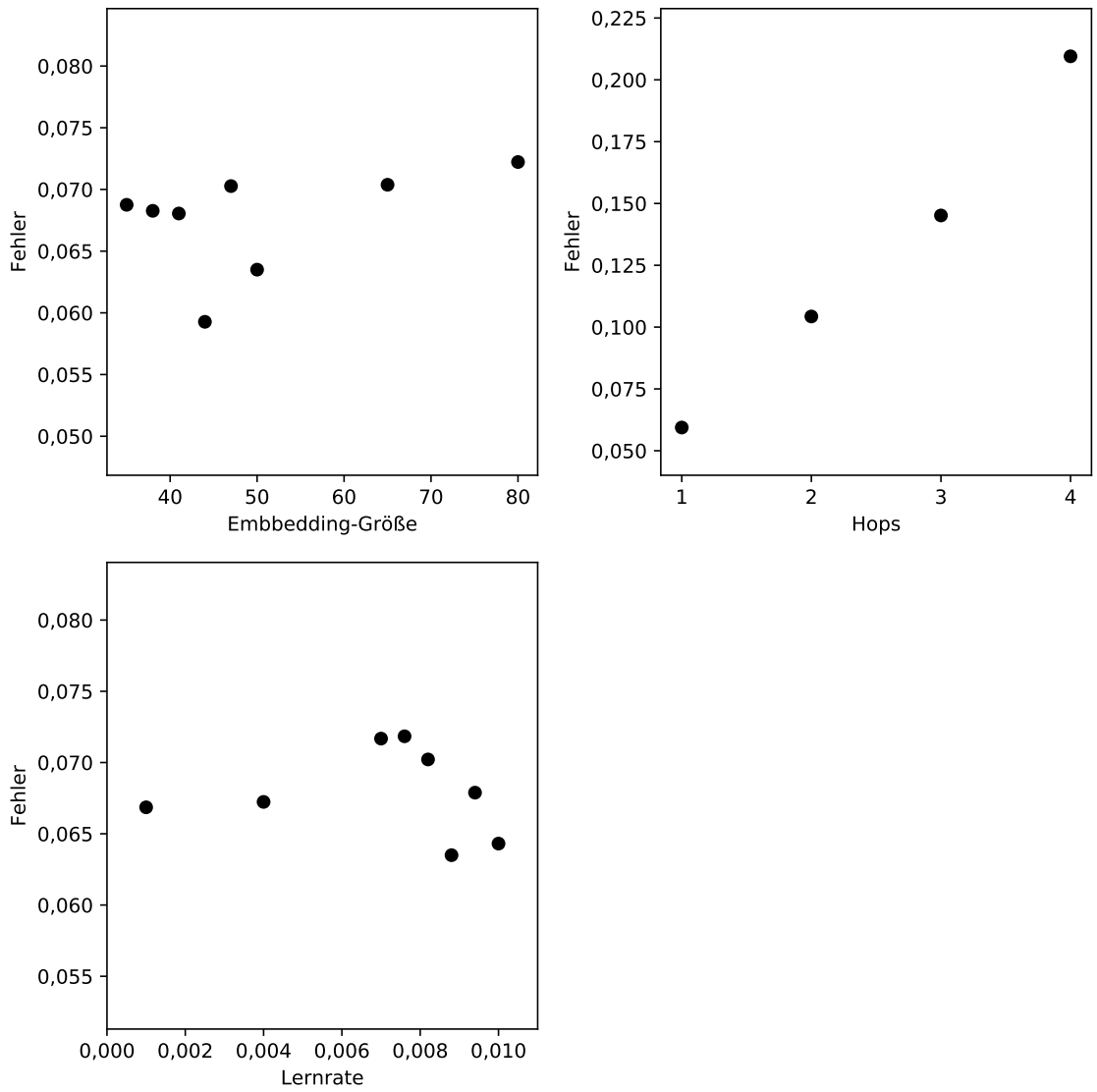

Abbildung 4.9.: Ergebnisse der voneinander getrennten Hyperparameteroptimierung mit der NLG-Komponente Kandidatenauswahl

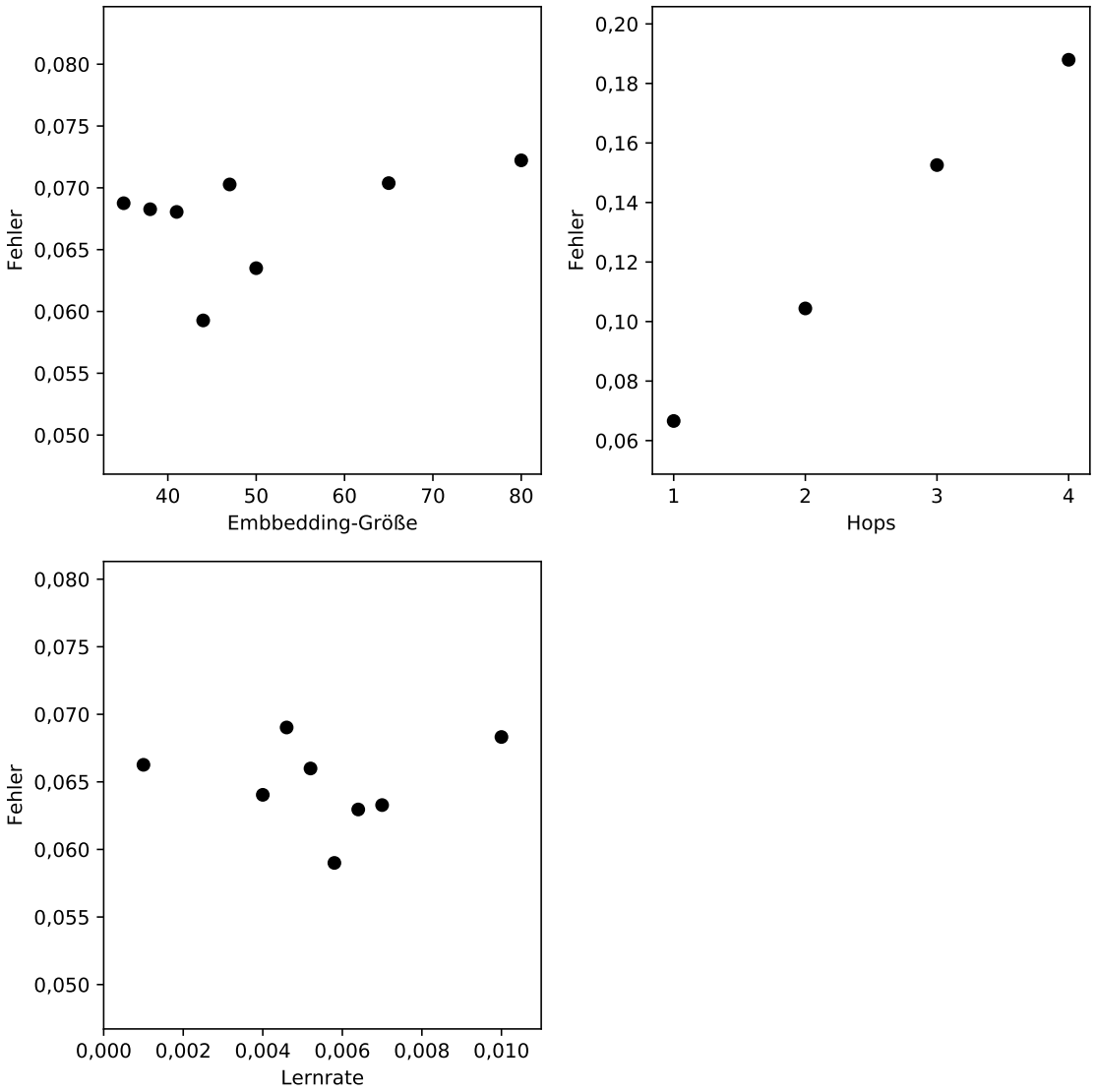

Abbildung 4.10.: Ergebnisse der voneinander getrennten Hyperparameteroptimierung mit der NLG-Komponente Kandidatenauswahl (sortierter Durchlauf)

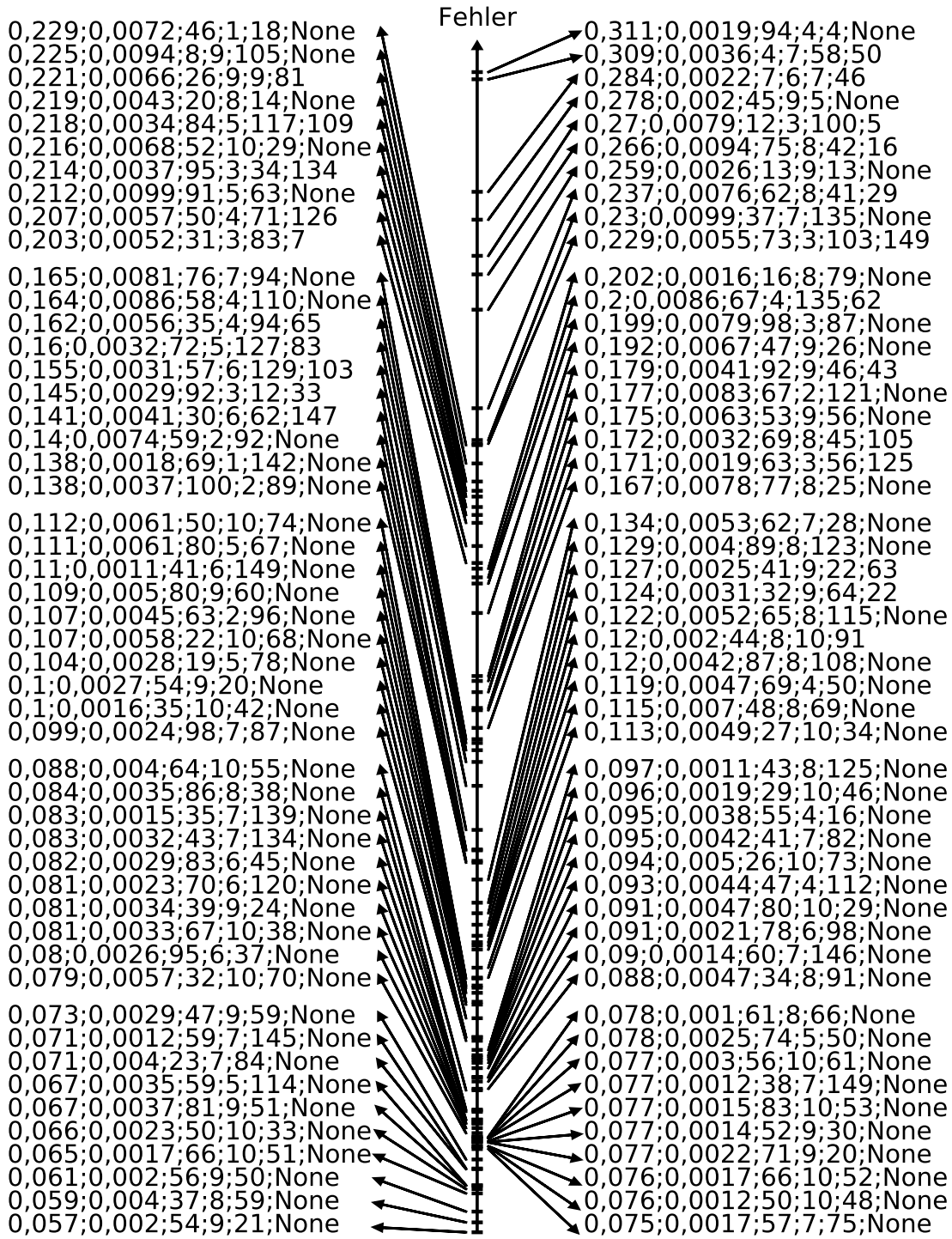

Abbildung 4.11.: Ergebnisse der TPE-Hyperparameteroptimierung mit der NLG-Komponente klassisches RNN (Fehler;Lernrate;Embedding-Größe;Hops;erste verdeckte Schicht;zweite verdeckte Schicht)

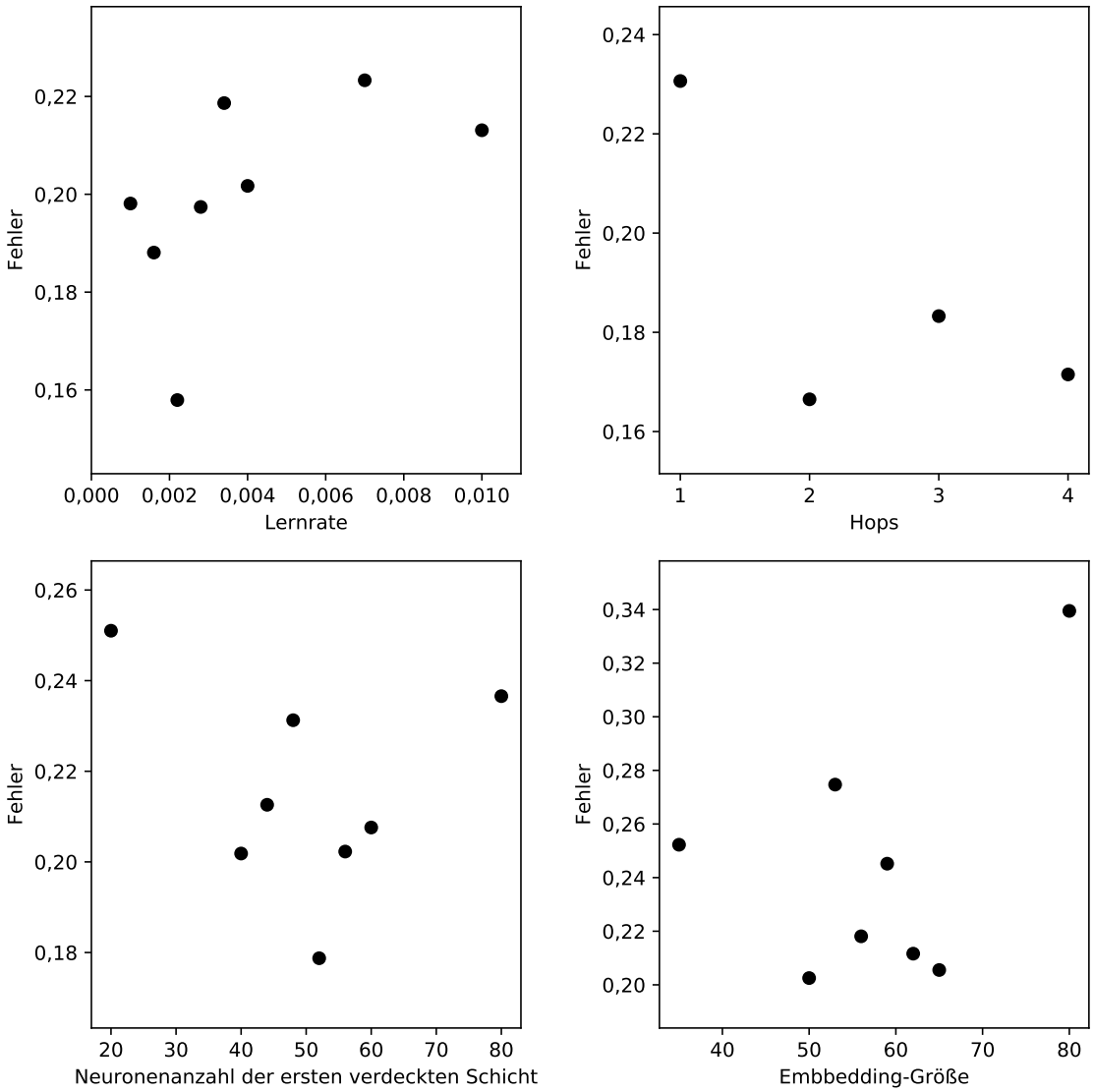

Abbildung 4.12.: Ergebnisse der voneinander getrennten Hyperparameteroptimierung mit der NLG-Komponente klassisches RNN

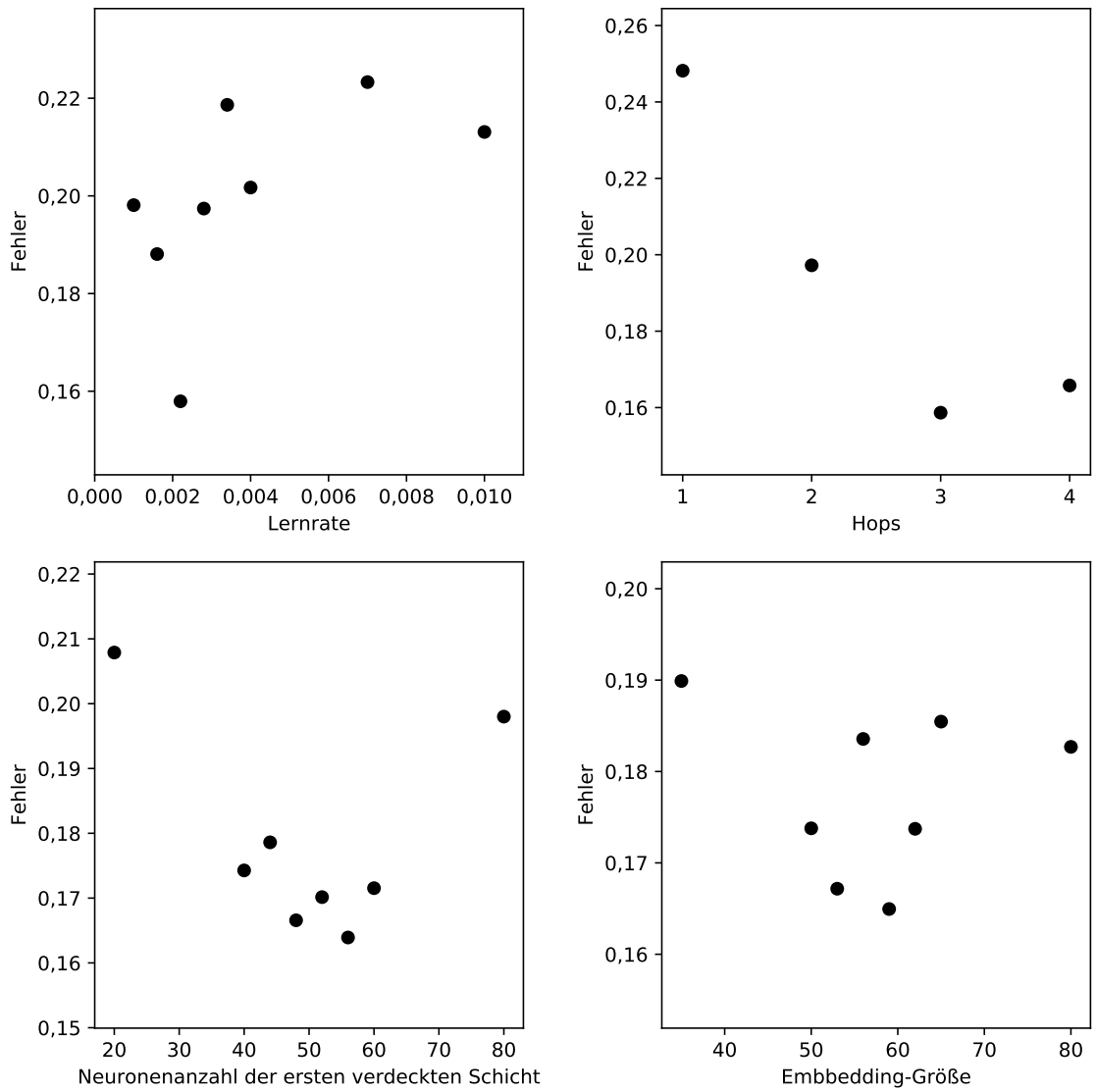

Abbildung 4.13.: Ergebnisse der voneinander getrennten Hyperparameteroptimierung mit der NLG-Komponente klassisches RNN (sortierter Durchlauf)

5. Evaluation

In diesem Kapitel wird das Dialogsystem evaluiert. Zuerst werden die Datensätze vorgestellt, die für die Evaluation verwendet werden. Danach wird die Genauigkeit, Fähigkeit zur Generalisierung, Reproduzierbarkeit der Ergebnisse und Ressourcenbeanspruchung des Dialogsystems untersucht.

5.1. Dialog bAbI Tasks

Dieser Datensatz ist in der Domäne Restaurantreservierung. Er umfasst eine Wissensdatenbank und sechs Aufgaben. Der Datensatz ist für (Bordes, Boureau und Weston, 2017) entworfen worden und wird unter (Weston, 2016a) zur Verfügung gestellt.

In der Wissensdatenbank sind für die ersten fünf Aufgaben alle Restaurants und ihre Attribute enthalten. Die Attribute sind Nationalität der Küche, Ort, Preis, Bewertung Telefonnummer, Adresse und Anzahl an Personen, für die ein Tisch reserviert werden kann.

Die ersten fünf Aufgaben besitzen einen Trainings-, Validations- und Testdatensatz mit jeweils 1 000 Dialogen. Es gibt außerdem noch einen weiteren Testdatensatz, der, im Gegensatz zu dem Validations- und dem anderen Testdatensatz, Restaurants und Attributwerte enthält, die nicht im Trainingsdatensatz vorkommen und auch 1 000 Dialoge umfasst. Dieser Testdatensatz wird Testdatensatz OOV (out of vocabulary) genannt. Die durschnittliche, minimale und maximale Dialoglänge der einzelnen Aufgaben ist in Abbildung 5.5 abgebildet.
Das allgemeine Format eines Dialogs ist als unvollständige Erweiterte Backus-Naur-Form (EBNF) in Abbildung 5.1 für die ersten fünf Aufgaben und in Abbildung 5.2 für die sechste Aufgabe dargestellt.
Die EBNF ist unvollständig, da nicht alle Nichtterminalsymbole in der EBNF definiert sind, da ansonsten die EBNF zu lang würde oder es nicht möglich ist, diese Nichtterminalsymbole in der EBNF zu definieren. Das nicht definierte Nichtterminalsymbol NUMBERING steht für die Nummerierung der Zeilen eines Dialogs. Die erste Zeile eines Dialogs wird mit 1 nummeriert. Jede Zeile erhöht die Nummer gegenüber der Vorgängerzeile um eins. Die Leerzeile, die die Dialoge voneinander trennt, hat keine Nummer. Die Terminalsymbole, für die die restlichen nicht definierten Nichtterminalsymbole stehen, lassen sich für die Aufgaben 1 bis 5 aus der Wissensdatenbank entnehmen.
Abbildung 5.4 zeigt einen konkreten Dialog aus Aufgabe fünf.
Die ersten fünf Aufgaben wurden mit einem Generator synthetisch erzeugt. Dabei bestehen die Äußerungen eines fiktiven Nutzers aus 69 Muster und die Äußerungen des

```
DIALOG = USER_BOT_INTERACTION DIALOG | INTERACTION_WITH_API_CALL DIALOG | EMPTY_LINE ;
USER_BOT_INTERACTION = NUMBERING, " ", USER_INPUT, "\t", BOT_INPUT_NOT_API_CALL  ;
INTERACTION_WITH_API_CALL = NUMBERING, " ", USER_INPUT, "\t" API_CALL, "\n",
    API_CALL_RESPONSE ;
API_CALL_RESPONSE = NUMBERING, " ", REAL_API_CALL_RESPONSE API_CALL_RESPONSE |
    REAL_API_CALL_RESPONSE ;
USER_INPUT = { ALL_CHARS - "\t" - "\n"}  ;
BOT_INPUT_NOT_API_CALL = { ALL_CHARS - "api_call" - "\t" }, " \n" :
API_CALL = "api_call ", CUISINE, " ", LOCATION, " ", PRICE, " ", NUMBER ;
REAL_API_CALL_RESPONSE = RESTAURANT_NAME " ", RESTAURANT_ATTRIBUT_ID, " ",
    RESTAURANT_ATTRIBUT_VALUE, "\n" ;
EMPTY_LINE = "\n" ;
ALL_CHARS = ? all characters ? ;
```

Abbildung 5.1.: unvollständige EBNF der Dialog bAbI Tasks für Aufgabe 1 bis 5

```
DIALOG = USER_BOT_INTERACTION DIALOG | INTERACTION_WITH_API_CALL DIALOG | EMPTY_LINE ;
USER_BOT_INTERACTION = NUMBERING, " ", USER_INPUT, "\t", BOT_INPUT_NOT_API_CALL  ;
INTERACTION_WITH_API_CALL = NUMBERING, " ", USER_INPUT, "\t" API_CALL, "\n",
    API_CALL_RESPONSE ;
API_CALL_RESPONSE = NUMBERING, " ", REAL_API_CALL_RESPONSE API_CALL_RESPONSE |
    REAL_API_CALL_RESPONSE ;
USER_INPUT = { ALL_CHARS - "\t" - "\n"}  ;
BOT_INPUT_NOT_API_CALL = { ALL_CHARS - "api_call" - "\t" }, " \n" :
API_CALL = "api_call ", CUISINE, " ", LOCATION, " ", PRICE  ;
REAL_API_CALL_RESPONSE = RESTAURANT_NAME " ", RESTAURANT_ATTRIBUT_ID, " ",
    RESTAURANT_ATTRIBUT_VALUE, "\n" | api\_call no result ;
EMPTY_LINE = "\n" ;
ALL_CHARS = ? all characters ? ;
```

Abbildung 5.2.: unvollständige EBNF der Dialog bAbI Tasks für Aufgabe 6

Systems aus 16 Muster. Ein Beispiel für ein Muster ist „may i have a table with <Name der Küche, z. B. spanish> cuisine“. Eine Liste aller extrahierten Muster befindet sich in Abschnitt A.1.
In der ersten Aufgabe geht es darum, einen API-Aufruf zu tätigen. Hierfür werden die gewünschten Werte für die Nationalität der Küche, den Ort, den Preis und der Anzahl an Personen, für die reserviert werden soll, ermittelt und damit ein API-Aufruf getätigt. Die Antwort des API-Aufrufs befindet sich bei den nachfolgenden Aufgaben im Antwortdialog, erkennbar durch das fehlende Tabzeichen in der Dialogzeile. Die Antwort des API-Aufrufs könnte aber auch aus der Wissensdatenbank abgeleitet werden.
In der zweiten Aufgabe werden nach dem API-Aufruf Parameterwerte des API-Aufrufs geändert und mit den aktualisierten Parametern wird ein neuer API-Aufruf getätigt.
In Aufgabe drei soll basierend auf dem Ergebnis des API-Aufrufs das bestmögliche Restaurant angezeigt werden. Das einzige sinnvolle Qualitätsunterscheidungsmerkmal bei der Struktur des API-Aufrufs und der Wissensdatenbank ist hierfür die Bewertung des Restaurants.
Weitere Informationen nach der Auswahl des Restaurants wie z. B. die Adresse oder die Telefonnummer anzuzeigen, sind die Aufgaben in Aufgabe vier.
Die ersten vier Aufgaben werden in Aufgabe fünf zu vollständigen Dialogen mit Elementen aus den ersten vier Aufgaben zusammengefasst.
Die sechste Aufgabe stammt von der zweiten Dialog State Tracking Challenge (Henderson, Thomson und Williams, 2014). Der Korpus der zweiten Dialog State Tracking Challenge wurde von Personen über den Dienst Amazon Mechanical Turk erstellt. Die Struktur des Korpus wurde an Aufgabe fünf von (Bordes, Boureau und Weston, 2017) angepasst. Aufgabe sechs umfasst einen Trainings-, Validations- und Testdatensatz. Da es keine Wissensdatenbank für die Aufgabe sechs gibt, müssen die Ergebnisse des API-Aufrufs immer aus dem Antwortdialog entnommen werden. Die Vielfalt der Art der Küchen ist mit 91 gegenüber den 10 Nationalitäten in den Aufgaben 1 bis 5 größer. Die Anzahl an Preiskategorien ist gleich und die Anzahl der Ortsangaben halbiert sich gegenüber den anderen Aufgaben, bei denen es 10 Ortsangaben gibt. Die Anzahl der Personen, für die eine Tischreservierung getätigt werden soll, entfällt. In Aufgabe sechs sind im Korpus Rechtschreibfehler vorhanden. Im Trainingsdatensatz wird z. B. food als ffood falsch geschrieben. Außerdem gibt es Äußerungen des Systems, die nicht immer zu den Äußerungen des Nutzers passen. Einige dieser Fehler im Trainingsdatensatz sind exemplarisch in Abbildung 5.3 aufgeführt.

Aufgabe drei und vier haben eine vermeidbare Komplexität. Das Dialogsystem bekommt die unbearbeiteten Ergebnisse der Wissensdatenbankanfrage. Eine niedrigere Komplexität hätte es, wenn intern die einzelnen Restaurantnamen mit Stellvertreternamen ersetzt würden. So müsste das Dialogsystem maximal so viele Stellvertreternamen lernen, wie die maximale Anzahl an Restaurants aus einer Wissendatenbankabfrage ist. Auf den Stellvertreternamen würde das Dialogsystem seine Entscheidung treffen und dann würden die Stellvertreternamen wieder zu den ursprünglichen Restaurantnamen zurücktransformiert werden. Um zu testen, ob diese Herangehensweise sinnvoll ist, wurde für diese Arbeit Aufgabe drei vereinfacht und Aufgabe vier vereinfacht den Dialog bAbI Tasks hinzugefügt. Bei Aufgabe vier ist jedoch das Problem, dass immer nur ein Eintrag von der Wissensdatenbank enthalten ist und so das Ergebnis bei Aufgabe vier auswendig gelernt werden

1. <SILENCE>↹Hello , welcome to the Cambridge restaurant system . You can ask for restaurants by area , price range or food type . How may I help you ?
2. more ↹What part of town do you have in mind?
3. more↹What kind of food would you like?
4. asian↹I'm sorry but there is no restaurant serving swiss food

1. <SILENCE>↹Hello , welcome to the Cambridge restaurant system . You can ask for restaurants by area , price range or food type . How may I help you ?
2. west↹What part of town do you have in mind?
3. west part of town↹What kind of food would you like?
4. doesnt matter↹api_call R_cuisine west moderate

7. im looking for thai food↹Did you say you are looking for a restaurant in the centre of town?
8. yes↹You are looking for a moroccan restaurant right?
9. no thai↹I'm sorry but there is no restaurant serving moroccan food

Abbildung 5.3.: Fehler in Aufgabe 6 der Dialog bAbI Tasks

kann, da immer die Adresse oder die Telefonnummer zum gleichen Stellvertreternamen zurückgegeben werden kann.
Diese Optimierung ist unter der Sichtweise, dass die Aufgaben drei und vier am besten nicht von dem neuronalen Netz gelöst werden sollten, jedoch hinfällig. Es wäre möglich die API-Aufrufe *api_call get_best_restaurant*, *api_call get_address* bzw. *api_call get_phone* zu tätigen. Diese Problemstellung ist prädestiniert für ein regelbasiertes System und somit könnten die Stärken von einem neuronalen Netz mit den Stärken von einem regelbasierten System kombiniert werden.

5.2. Robotersteuerungsaufgaben

Der Datensatz, der das System für eine Tauglichkeit für den Haushaltsroboter Armar 3 evaluieren soll, hat nahezu das gleiche Format wie die Aufgaben der Dialog bAbI Tasks. Zeilen eines Dialoges sind mit 1 beginnend durchnummeriert, jede Zeile beinhaltet eine Äußerung des Nutzers und eine Äußerung als Antwort des Systems und die Äußerungen des Nutzers und des Systems sind mit einem Tabzeichen und die Dialoge mit einer Leerzeile getrennt. Der einzige Unterschied im Format ist, dass es keine Zeilen mehr für die Ergebnisse für einen API-Aufruf gibt.
Mit dem Datensatz wird sowohl getestet, ob aus Befehlen passende API-Aufrufe generiert werden können als auch ob komplexere Dialoge geführt werden können. Der Korpusgenerator von (Ovchinnikova, 2016) wurde als Grundlage genommen. Zusätzlich wurden noch weitere Fähigkeiten aufgenommen, um zu zeigen, dass Parameter für API-Aufrufe aus einem längeren Dialogverlauf gewonnen werden können. In diesen komplexen Dialogen werden Rückfragen gestellt und Parameter verstreut im ganzen Dialog für die Generierung

```
1 hello↹hello what can i help you with today
2 can you book a table for two in rome in a moderate price range↹i'm on it
3 <SILENCE>↹any preference on a type of cuisine
4 with french cuisine↹ok let me look into some options for you
5 <SILENCE>↹api_call french rome two moderate
6 actually i would prefer in madrid↹sure is there anything else to update
7 no↹ok let me look into some options for you
8 <SILENCE>↹api_call french madrid two moderate
9 resto_madrid_moderate_french_3stars R_phone resto_madrid_moderate_french_3stars_phone
10 resto_madrid_moderate_french_3stars R_cuisine french
11 resto_madrid_moderate_french_3stars R_address
    resto_madrid_moderate_french_3stars_address
12 resto_madrid_moderate_french_3stars R_location madrid
13 resto_madrid_moderate_french_3stars R_number two
14 resto_madrid_moderate_french_3stars R_price moderate
15 resto_madrid_moderate_french_3stars R_rating 3
16 resto_madrid_moderate_french_1stars R_phone resto_madrid_moderate_french_1stars_phone
17 resto_madrid_moderate_french_1stars R_cuisine french
18 resto_madrid_moderate_french_1stars R_address
    resto_madrid_moderate_french_1stars_address
19 resto_madrid_moderate_french_1stars R_location madrid
20 resto_madrid_moderate_french_1stars R_number two
21 resto_madrid_moderate_french_1stars R_price moderate
22 resto_madrid_moderate_french_1stars R_rating 1
23 resto_madrid_moderate_french_8stars R_phone resto_madrid_moderate_french_8stars_phone
24 resto_madrid_moderate_french_8stars R_cuisine french
25 resto_madrid_moderate_french_8stars R_address
    resto_madrid_moderate_french_8stars_address
26 resto_madrid_moderate_french_8stars R_location madrid
27 resto_madrid_moderate_french_8stars R_number two
28 resto_madrid_moderate_french_8stars R_price moderate
29 resto_madrid_moderate_french_8stars R_rating 8
30 <SILENCE>↹what do you think of this option: resto_madrid_moderate_french_8stars
31 i love that↹great let me do the reservation
32 may i have the address of the restaurant↹here it is
    resto_madrid_moderate_french_8stars_address
33 what is the phone number of the restaurant↹here it is
    resto_madrid_moderate_french_8stars_phone
34 you rock↹is there anything i can help you with
35 no thank you↹you're welcome
```

Abbildung 5.4.: ein Dialog aus Aufgabe 5 der Dialog bAbI Tasks (Bordes, Boureau und Weston, 2017)

Aufgabe	Training	Validation	Test	Test OOV
1	4/6,02/6/8	4/6,02/6/8	4/5,94/6/8	4/6,02/6/8
2	8/9,5/9/11	8/9,51/10/11	8/9,46/9/11	8/9,46/9/11
3	26/33,21/32/52	26/34,46/33/61	26/34,28/33/61	26/33,69/32/60
4	10/10,51/11/11	10/10,53/11/11	10/10,5/10/11	10/10,51/11/11
5	34/41,96/41/69	34/42,26/41/70	34/42,21/41/67	34/41,99/40/68
6	3/51,6/25/491	5/56,23/26/454	4/37,93/24/788	0/0/0/0

Abbildung 5.5.: Dialoglängen (Minimum/Durchschnitt/Median/Maximum) der einzelnen Aufgaben der Dialog bAbI Tasks

des API-Aufrufs verwendet.
Im Datensatz gibt es API-Aufrufe für die 1, 2, 3 und 5 Interaktionen nötig sind, um den API-Aufruf zu generieren. Dabei sind die Parameter und der nötige API-Aufruf über 1, 2, 3 bzw. 4 Interaktionen verteilt.
In Abschnitt A.5 befinden sich die Musterdialoge, deren Attribute und die Attributswerte. Das Programm *util/robot_corpus_generator.py* erzeugt mithilfe des DialogGenerators daraus Dialoge.

Dieser Datensatz hat dazu beigetragen, bei der Antworterzeugung per klassischem RNN nicht nur ein Wort, sondern die letzten *n*-Wörter zu verwenden.
Um den Datensatz verwenden zu können, wurde in der Anwendung *train.py* geändert, dass anstatt der Nummer einer Aufgabe der Dialog bAbI Tasks auch alternativ der Namen des Trainings-, Validations- und der Testdatensätze eingeben werden kann.

5.3. Ergebnisse

Alle Genauigkeiten im Folgenden beziehen sich auf die Genauigkeit im Testdatensatz, gemessen an dem Trainingsstand der Epoche mit der höchsten Genauigkeit im Validationsdatensatz bei Betrachtung der Teildialoge. Dabei wurden die Aufgaben jeweils sechsmal mit 100 Epochen trainiert und nach jeder Epoche wurde evaluiert.
Die Antwortgenerierungsart Wort für Wort mit einer LSTM wurde nicht weiter untersucht oder optimiert, da sie bei Aufgabe eins mit einer Genauigkeit von 88,72 % bei der Betrachtung der Teildialogen und 33,37 % bei der Betrachtung, wie viele Dialoge komplett richtig beantwortet werden konnten, nicht gut abschnitt. In Aufgabe eins waren unoptimiert für die Antwortgenerierung mit der Kandidatenauswahl und der Antwortgenerierung Wort für Wort mit dem RNN gute Ergebnisse möglich.
Auch die Antwortgenerierung mit dem SC-LSTM erzielte mit einer Genauigkeit von 83,29 % bei Betrachtung der Teildialogen und 0,90 % bei der Betrachtung, wie viele Dialoge komplett richtig beantwortet werden konnten, keine guten Ergebnisse und wurde deswegen ebenfalls auch nicht weiter untersucht oder optimiert.
Für die anderen Antworterzeugungsarten sind die Ergebnisse in Prozent in der folgenden Tabelle aufgeführt. Das Ergebnis ist jeweils die Genauigkeit des Testdatensatzes für die Teildialoge, in Klammern ist die Genauigkeit wie viele Dialoge komplett richtig beantwortet werden konnten. Bei dem RNN wurde bei den Robotersteuerungsaufgaben die letzten beiden ausgegebenen Wörter, ansonsten nur das letzte ausgegebene Wort, als Eingabe verwendet.

Aufgabe	regelbasiert, evaluiert in (Bordes, Boureau und Weston, 2017)	Kandidaten, evaluiert in (Bordes, Boureau und Weston, 2017)	Kandidaten	RNN
Aufgabe 1	100 (100)	99,6 (99,6)	99,95 (99,70)	99,92 (99,50)
Aufgabe 2	100 (100)	100 (100)	99,71 (97,30)	98,65 (87,30)
Aufgabe 3	100 (100)	74,9 (2)	74,85 (0,00)	74,89 (0,00)
Aufgabe 4	100 (100)	59,5 (3)	57,35 (0,20)	57,18 (0,00)
Aufgabe 5	100 (100)	96,1 (49,4)	95,04 (37,50)	85,60 (3,90)
Aufgabe 6	33,3 (0)	41,1 (0)	37,93 (0,18)	21,19 (0,00)
Aufgabe 3 vereinfacht	- (-)	- (-)	93,27 (52,60)	96,22 (70,70)
Aufgabe 4 vereinfacht	- (-)	- (-)	100,00 (100,00)	100,00 (100,00)
Robotersteuerungsaufgaben	- (-)	- (-)	98,79 (95,73)	99,76 (99,15)

Die Hyperparameter der für diese Arbeit evaluierten Dialogsysteme wurden für eine möglichst hohe Genauigkeit bei den Teildialogen in Aufgabe fünf optimiert. Die nicht veröffentlichte Referenzimplementierung aus (Bordes, Boureau und Weston, 2017) hat bei Betrachtung, wie viele Dialoge komplett richtig beantwortet werden konnten, eine erheblich größere Genauigkeit. Hier rächt sich wahrscheinlich, dass in die Optimierung nicht die Genauigkeit der kompletten Dialoge mitgewichtet wurde.
Als Hyperparameter wurde für die Antwortgenerierung per Kandidatenauswahl 0,0058 für die Lernrate des Adam Optimierers, 44 für die Embedding-Größe und 1 für die Anzahl an Hops und für die Antwortgenerierung Wort für Wort mit dem RNN 0,0022 für die Lernrate des Adam Optimierers, 59 für die Embedding-Größe, 3 für die Anzahl an Hops und 50 für die Neuronenanzahl der ersten und einzigen verdeckten Schicht ausgewählt.
Bei allen Antwortgenerierungsarten wurden die Batches vor jeder Epoche neu durgewürfelt.

5.4. Fähigkeit zur Generalisierung

In diesem Abschnitt wird untersucht, ob das Dialogsystem, beschrieben in Kapitel 4, eine gute Generalisierungsfähigkeit bietet. Die Genauigkeit des Dialogsystems wurde mit den Dialog bAbI Tasks und den Robotersteuerungsaufgaben gemessen. Da nicht bekannt ist, ob eine Anforderung der Dialog bAbI Tasks ist, die Generalisierungsfähigkeit zu testen, wird in diesem Abschnitt analysiert, ob die Generalisierungsfähigkeit getestet wird und wo sie nicht getestet wird, wird der Datensatz so erweitert, dass er diese auch testet.
In den OOV-Testdatensätzen der Dialog bAbI Tasks werden in den Dialogen ausschließlich Orte und Nationalitäten der Küchen, die nicht im Trainingsdatensatz vorkommen, verwendet. Jedoch kann das Dialogsystem, wie in Abschnitt 4.2 begründet, damit nicht umgehen. In dieser Hinsicht bietet das System eine schlechte Generalisierungsfähigkeit.

Die normalen Testdatensätze verwenden nur API-Aufrufe mit Orten und Nationalitäten der Küche, die in den jeweiligen Trainingsdatensätzen vorhanden sind. In Abbildung 5.6 ist aufgeführt, wie viele API-Aufrufe aus den Validations- und Testdatensätze mit dem Trainingsdatensatz identisch sind. Der OOV-Testdatensatz hat keine gemeinsame API-Aufrufe, da der Ort und die Küche immer Bestandteil des API-Aufrufes sind und der Ort und die Küche des OOV-Testdatensatzes nicht im Trainingsdatensatz enthalten sind. Aufgabe 3 und 4 haben keine API-Aufrufe.

Aufgabe	Training	Training ∩ Validation	Training \ Validation	Training ∩ Test	Training \ Test
1	1000	0	1000	0	1000
2	2000	1932	68	1924	76
3	0	0	0	0	0
4	0	0	0	0	0
5	2000	1319	681	1309	691
6	1844	592	75	1046	42

Abbildung 5.6.: Anzahl der API-Aufrufe der Dialog bAbI Tasks

In Aufgabe 1 sind alle API-Aufrufe des Testdatensatzes nicht im Trainingsdatensatz enthalten, deswegen kann festgestellt werden, dass das System nicht die API-Aufrufe auswendig lernt, sondern sie dynamisch zusammensetzen kann. In Aufgabe 2 sind die meisten API-Aufrufe des Testdatensatzes im Trainingsdatensatzes enthalten. Jedoch sind von den veränderten API-Aufrufe 27 Stück nicht im Trainingsdatensatz enthalten. Die Genauigkeiten mit denen die Dialoge mit den API-Aufrufen, die nicht im Trainingsdatensatz vorhanden sind, vom System beantwortet werden, werden zuerst für das Dialogsystem mit der Kandidatenauswahl und dann für das System mit der Antwortgenerierung Wort für Wort mit dem RNN genannt. Diese Dialoge haben eine Genauigkeit von 100 % bzw. 97,73 % bei Betrachtung der Teildialoge und 100 % bzw. 77.78 % bei der Betrachtung, wie viele Dialoge komplett richtig beantwortet wurden, d. h. dass 27 bzw. 21 von den 27 Dialogen komplett richtig beantwortet wurden. Damit kann auch für diese Aufgabe festgestellt werden, dass das System, wenigstens bei der Verwendung der Kandidatenauswahl, fähig ist, API-Aufrufe dynamisch zu erzeugen. Aufgabe 5 muss nicht betrachtet werden, da sie nur eine Kombination aus den ersten vier Aufgaben ist. Es kann also festgestellt werden, dass die Muster beliebig mit Werten aufgefüllt werden können, solange solche Werte schon einmal im Trainingsdatensatz vorgekommen sind.
Wie weit kann sich jedoch von den Mustern wegbewegt werden? Alle Muster, die im Testdatensatz vorkommen, kommen auch im Trainingsdatensatz vor. Deshalb wurden einige Muster ausgewählt und modifiziert. Es wurden entweder Wörter entfernt, hinzugefügt oder sowohl entfernt als auch hinzugefügt. Dann ersetzen sie die ursprüngliche Version in ausgewählten Dialogen und es wird gemessen, ob diese Dialoge weiterhin vollständig korrekt ausgeführt werden können. Die ursprünglichen Muster, ihre Modifikationen und die Ergebnisse sind in Abschnitt A.6 aufgeführt. Ein Häkchen bedeutet, dass das Dialogsystem trotz modifizierter Äußerung den kompletten Dialog richtig beantwortet hat. Wohingegen ein X bedeutet, dass der Dialog nicht komplett richtig beantwortet werden konnte. Das

erste Häkchen oder X gehört zur Antworterzeugung per Kandidatenauswahl, das zweite zur Antworterzeugung Wort für Wort mit dem RNN. Die unmodifizierten Dialoge konnten alle komplett von beiden Antwortgenerierungsarten gelöst werden. Der Test mit den modifizierten Muster zeigt, dass keine allgemeine Aussage getroffen werden kann, wann Äußerungen zu stark modifiziert sind. Es kommt von Fall zu Fall darauf an.

5.5. Reproduzierbarkeit der Ergebnisse

In Abschnitt 5.3 werden die besten Ergebnisse aus 6 Durchläufen mit jeweils 100 Epochen abgebildet. Um die Reproduzierbarkeit der Ergebnisse festzustellen, werden zu jeder Aufgabe und zu den beiden Antworterzeugungsarten Kandidatenauswahl und Antworterzeugung Wort für Wort mit einem RNN die 6 Durchläufe, evaluiert nach jeder fünften Epoche, in Abschnitt A.7 abgebildet. Dabei bildet eine Abbildung jeweils eine Aufgabe ab und beinhaltet die Genauigkeit des Testdatensatzes.
Zusammengefasst lässt sich für die Reproduzierbarkeit sagen, dass die Ergebnisse mit der Kandidatenauswahl sich relativ gut reproduzieren lassen und beim Erreichen guter Werte auch stabil diese Werte behalten. Bei der Antworterzeugung Wort für Wort mit dem RNN sind oftmals mehrere Durchläufe nötig, um gute Ergebnisse zu erzielen und nach dem Erreichen guter Werte werden diese oftmals auch nicht beibehalten. Hier ist es wichtig nach jeder Epoche zu evaluieren und gute Zustände zwischenzuspeichern. Diese schlechte Reproduzierbarkeit kommt wahrscheinlich von dem großen Zufall in der Initialisierung der Gewichte des RNNs, beschrieben in Abschnitt 4.2.

5.6. Ressourcenbeanspruchung

Die Evaluation wurde auf einem Computer durchgeführt, der für die Evaluation eine nVidia Titan X (Pascal) bzw. eine nVidia Geforce GTX 1080 Ti, 2 Threads eines Intel Xeon Prozessor E5-2620 v4 und 6 Gigabyte Arbeitsspeicher zur Verfügung stellte. Die Dauer für die Auswertung von 100 Epochen mit Evaluation nach jeder Epoche und die verwendete Grafikkarte sind für alle in diesem Kapitel vorgestellten Aufgaben in Abbildung 5.7 abgebildet. Dabei wurde die Python-Funktion *time.process_time*, die nur die tatsächliche Prozesszeit zählt, verwendet und das arithmetische Mittel der sechs Durchläufe, aus denen die Ergebnisse für Abschnitt 5.3 ermittelt werden, verwendet.
Der Speicherverbrauch der Checkpoint-Dateien, die von Tensorflow erzeugt werden, um das trainierte Modell wieder herzustellen, sind in Abbildung 5.8 abgebildet. Zusätzlich muss noch die Einstellungsdatei *settings.pickle* gespeichert werden, die für die Aufgaben der Datensätze in dieser Arbeit aber jeweils kleiner als 1 Megabyte ist.

Aufgabe	Kandidatenauswahl	RNN
Aufgabe 1	18	36
Aufgabe 2	38	68
Aufgabe 3	115	150
Aufgabe 4	13	22
Aufgabe 5	203	273
Aufgabe 6	241*	295
Aufgabe 3 vereinfacht	112	123
Aufgabe 4 vereinfacht	12	15
Robotersteuerungsaufgaben	1	2*

* nVidia Geforce GTX 1080 Ti (ansonsten: nVidia Titan X (Pascal))

Abbildung 5.7.: Dauer des Trainings mit Evaluation nach jeder Epoche (gerundet in volle Minuten)

Aufgabe	Kandidatenauswahl	RNN
Aufgabe 1	62,7	3
Aufgabe 2	62,7	3
Aufgabe 3	63,1	3
Aufgabe 4	62,9	3
Aufgabe 5	63,3	3
Aufgabe 6	10,8	1,3
Aufgabe 3 vereinfacht	0,8	0,7
Aufgabe 4 vereinfacht	0,8	0,7
Robotersteuerungsaufgaben	0,3	0,7

Abbildung 5.8.: Größe der Checkpoint-Dateien von Tensorflow (gerundet in Megabyte)

6. Fazit

6.1. Fazit

Das Dialogsystem konnte in der Evaluation in vielen Bereichen von seiner Qualität überzeugen.
Bei Aufgabe 1 der Dialog bAbI Tasks - bei der API-Aufrufe mit 4 Parametern, die über den Dialog verteilt sind, generiert werden - konnten sowohl mit der Antwortgenerierungsart Kandidatenauswahl als auch mit der Antwortgenerierungsart Wort für Wort mit einem klassischen RNN über 99,5 % der Dialoge des Testdatensatzes vollständig korrekt geführt werden. Diese Werte und dass die API-Aufrufe des Testdatensatzes nicht mit den gleichen Parametern im Trainingsdatensatz vorkommen, lassen schlussfolgern, dass prinzipiell mit dem Dialogsystem API-Aufrufe generiert werden können, deren Parameter in einem Dialog verteilt sind. Damit konnte diese Anforderung der Zielsetzung erreicht werden.
Bei der Neugenerierung von API-Aufrufen, es wurde gegenüber dem letzen API-Aufruf mindestens ein Parameter geändert, getestet in Aufgabe 2 der Dialog bAbI Tasks, hat das Dialogsystem mit der Antwortgenerierungsart Kandidatenauswahl über 97 % der Dialoge richtig durchgeführt. Bei der Antwortgenerierungsart Wort für Wort mit einem klassischen RNN hat sich jedoch die Qualität mit etwas über 87 % korrekt geführten Dialogen stark verschlechtert. Das Dialogsystem mit der Antwortgenerierungsart Kandidatenauswahl kann diese Genauigkeit in Aufgabe 2 der Dialog bAbI Tasks noch verbessern, wenn nur die API-Aufrufe betrachtet werden, die nicht im Trainingsdatensatz vorhanden sind. Damit lässt sich feststellen, dass das Aktualisieren von API-Aufrufen prinzipiell mit dem Dialogsystem bei Verwendung der Antwortgenerierungsart Kandidatenauswahl möglich ist.
Bei der Auswahl des besten Restaurants und der Ausgabe der Adresse bzw. der Telefonnummer aus der Liste mit Restaurants, die von einem API-Aufruf erzeugt wird, konnte das Dialogsystem mit allen Antwortgenerierungsarten mit einer Genauigkeit von unter 1 % keine zufriedenstellenden Ergebnisse liefern. Es hat sich zwar gezeigt, dass durch Transformationen der Eingaben die Ergebnisse verbessert werden konnten, jedoch kann dann gleich ein API-Aufruf für diese Aufgabe eingeführt werden, der dann diese Aufgabe perfekt lösen kann und damit die Vorteile eines Memory Networks mit den Vorteilen eines regelbasierten Systems kombiniert.
Das Dialogsystem bietet mit dem End-To-End Training eine einfache Möglichkeit auf verschiedene Domänen trainiert zu werden. Damit wird auch die Anforderung aus der Zielsetzung erfüllt, dass sich das System auf verschiedene Domänen anpassen lassen soll. Dabei ist für die Anpassung nur ein Trainingsdatensatz nötig. Die Erstellung des Datensatzes kann jedoch Domänenwissen erfordern. Diese Einfachheit der Anpassung wird jedoch damit erkauft, dass die Qualität des Dialogsystem sehr von dem Datensatz abhängig ist, mit dem es trainiert wurde. Es ist zwar möglich API-Aufrufe zu generieren

und zu aktualisieren, die Äußerungen dürfen jedoch nicht zu stark von den Äußerungen des Trainingsdatensatzes abweichen. Dies war bei den Dialog bAbI Tasks und den Robotersteuerungsaufgaben der Fall. Diese nur geringe mögliche Abweichung lässt sich begründen, dass Äußerungs-Embeddings berechnet werden. Bei untrainierten Äußerungen hat das berechnete Äußerungs-Embedding gegebenfalls ein zu großer Abstand von den bekannten Äußerungs-Embeddings und lässt sich nicht einordnen. Es lässt sich nicht genau festlegen, ab wann der Abstand zu groß ist. Wie in der Evaluation untersucht, reicht manchmal ein zugefügtes Wort, um den Abstand zu groß werden zu lassen. Ein anderes Mal können mehrere Wörter hinzugefügt und entfernt werden. Die Einfachkeit des Trainings, welche mit einem umfangreichen Trainingsdatensatz erkauft werden muss, bestätigt das No-free-Lunch-Theoreme.
Die Anwortgenerierung mit der Kandidatenauswahl hat in der Evaluation in fast allen Bereichen bessere Ergebnisse als die Antworterzeugung Wort für Wort mit einem klassischen RNN erzielt. Dafür nimmt bei der Antworterzeuguung Wort für Wort mit dem RNN nicht linear mit der Kandidatengröße der Speicherplatzverbrauch und die Laufzeit zu. Bei den kleinen Kandidatengrößen von maximal 4212 Kandidaten bei den verwendeten Datensätzen war die Laufzeit der Antwortgenerierung Wort für Wort mit dem klassischen RNN sogar größer. Der größere Speicherplatzverbrauch macht sich jedoch schon bei dieser Kandidatengröße bemerkbar, bei den Dialog bAbI Tasks benötigt das Dialogsystem mit der Antwortgenerierungsart Kandidatenauswahl eine Größenordnung mehr Speicherplatz.

6.2. Weiterführende Arbeit

Die für diese Arbeit entworfene Robotersteuerungsaufgaben haben nur die Aufgabe der Evaluation, ob das in dieser Arbeit entwickelte Dialogsystem überhaupt sinnvoll ist, im Armar 3 zu verwenden, zu dienen. Wie sich in der Evaluation herausgestellt hat, besitzt das Dialogsystem eine gute Performanz, benötigt jedoch einen umfassenden Korpus, der alle Fähigkeiten und eine große Anzahl von Äußerungen abdecken muss. Solch ein umfassender Korpus könnte in Zusammenarbeit mit dem Team von Armar 3 entwickelt werden. Mithilfe dieser Zusammenarbeit könnte das Dialogsystem auch in den Armar 3 implementiert werden.

Das Anlegen eines umfassenden Korpus ist für jede Domäne notwendig. In zukünftigen Arbeiten könnte untersucht werden, ob es sinnvoll ist, Äußerungs-Embeddings unabhängig von der Domäne zu trainieren. Die vortrainierten Äußerungs-Embeddings könnten dann im Dialogsystem verwendet werden.

Da jedes Wort auf eine natürliche Zahl abgebildet wird, werden falsch geschriebene Wörter auf unterschiedliche Zahlen abgebildet, d. h. im Äußerungs-Embedding muss gelernt werden, dass die Äußerungen sich auch mit Rechtschreibfehler nicht ändern. Dies erfordert Trainingssätze mit Rechtschreibfehler, die zumindest die häufigsten Fehler abdecken. Es erscheint jedoch sinnvoller eine Rechtschreibkorrekturkomponente dem eigentlichen Dialogsystem vorzulagern, um Rechtschreibfehler nicht in die Trainingsdatensätze aufnehmen zu müssen.

Des Weiteren könnte untersucht werden, inwiefern eine Normalisierung von Wörtern die Qualität des Dialogsystems beeinflusst.
Eine weitere Verbesserung könnte gegebenenfalls erreicht werden, wenn unbekannte Wörter von einer Vorkomponente ausgefiltert werden würden.

Eine weitere Untersuchungsmöglichkeit wäre, mit größeren Datensätze zu trainieren, um auch in der Praxis die Geschwindigkeitsvorteile der Antwortgenerierungsarten Wort für Wort herauszuarbeiten.

In der Arbeit werden klassische rekurrente neuronale Netze, LSTM und SC-LSTMs für die Antworterzeugung Wort für Wort verwendet. Als weiterführende Arbeit könnten weitere Elemente für die Antworterzeugung Wort für Wort, wie z. B. das in (Wen, Gasic, Kim u. a., 2015) vorgestellte rekurrente neuronale Netz mit einem Convolutional Sentence Reranking, implementiert und evaluiert werden.

Bei der Hyperparameteroptimierung könnte auch mitgewichtet werden, wie hoch die Genauigkeit bei der Beantwortung der kompletten Dialoge ist. Nach der Hyperparameteroptimierung könnte dann evaluiert werden, ob diese Gewichtung Vorteile bei der Genauigkeit der Beantwortung der kompletten Dialoge gebracht hat.

Es wurden mehrere Ansätze vorgestellt, wie neuronale Netzarchitekturen auf einen Speicher zurückgreifen können. Die evaluierten Korpora könnten auch mit einem Recurrent Entity Network (Henaff u. a., 2017), einem Differentiable Neural Computer (Graves, Wayne, Reynolds u. a., 2016), einer Neural Turing Machine (Graves, Wayne und Danihelka, 2014) und einem Dynamic Memory Network (Kumar u. a., 2016) trainiert und evaluiert werden.

Natürliche Sprache kann sehr umfangreich und komplex sein. Auch mit sehr großen Datensätze kann nicht alles abgedeckt werden. Ein intelligentes Dialogsystem sollte also bei falschen Antworten vom Nutzer korrigiert werden können. In (Weston, 2016b) werden im Stile der 20 bAbI Tasks 10 Aufgaben definiert, bei denen das System auf verschiedene Arten auf Antworten reagiert, z. B. wird in natürlicher Spache geantwortet, ob die Antwort korrekt oder nicht korrekt ist oder es werden Hinweise gegeben, die dem System helfen sollen, korrekt zu antworten. Daneben werden zwei Verfahren vorgestellt, um aus den Antworten des Nutzers zu lernen. In (Li u. a., 2017a) wird zusätzlich, zu den zwei Verfahren in (Weston, 2016b), eine Lerntechnik, die auf dem Reinforcement Learning basiert, vorgestellt.
Ein System welches korrigiert werden muss, scheint jedoch unprofessionell. Besser wäre es, bei Unsicherheiten nachzufragen. Jedoch sollte nur so viel wie nötig, aber so wenig wie möglich, nachgefragt werden, um einen flüssigen Dialog zu ermöglichen. In (Li u. a., 2017b) wird ein Online Reinforcement Learning Verfahren vorgestellt, bei dem gelernt werden soll, wann es sinnvoll ist, nachzufragen und wann es nicht sinnvoll ist.

Literatur

Asfour, Tamim u. a. (2006). „ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control“. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots, Genova, Italy, December 4-6, 2006*. IEEE, S. 169–175.

Bergstra, James (2013). *Hyperopt*. URL: http://hyperopt.github.io/hyperopt/.

Bergstra, James u. a. (2017). *hyperopt*. GitHub repository. URL: https://github.com/hyperopt/hyperopt.

Bergstra, James u. a. (2011). „Algorithms for Hyper-Parameter Optimization“. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*. Hrsg. von John Shawe-Taylor u. a., S. 2546–2554. URL: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.

Bordes, Antoine, Y-Lan Boureau und Jason Weston (2017). „Learning End-to-End Goal-Oriented Dialog“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*. URL: http://arxiv.org/abs/1605.07683v4.

Cybenko, G. (1989). „Approximation by superpositions of a sigmoidal function“. In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4, S. 303–314.

Destatis, Statistisches Bundesamt (2017). *Finanzen und Steuern Umsatzsteuerstatistik (Voranmeldungen) 2015*. URL: https://www.destatis.de/DE/Publikationen/Thematisch/FinanzenSteuern/Steuern/Umsatzsteuer/Umsatzsteuer2140810157004.pdf?__blob=publicationFile.

Dodge, Jesse u. a. (2016). „Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016*. URL: http://arxiv.org/abs/1511.06931v6.

Facebook (2017a). *MemNN*. GitHub repository. URL: https://github.com/facebook/MemNN.

– (2017b). *Memory-Augmented Neural Networks*. URL: https://raw.githubusercontent.com/facebook/MemNN/master/README.md.

Graves, Alex, Greg Wayne und Ivo Danihelka (2014). „Neural Turing Machines“. In: *CoRR* abs/1410.5401. URL: http://arxiv.org/abs/1410.5401v2.

Graves, Alex, Greg Wayne, Malcolm Reynolds u. a. (2016). „Hybrid computing using a neural network with dynamic external memory“. In: *Nature* 538.7626, S. 471–476.

Harris, Ryan (2012). *Neural network tutorial: The back-propagation algorithm*. Youtube. URL: https://www.youtube.com/watch?v=aVId8KMsdUU.

Henaff, Mikael u. a. (2017). „Tracking the World State with Recurrent Entity Networks“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*. URL: http://arxiv.org/abs/1612.03969v3.

Henderson, Matthew, Blaise Thomson und Jason Williams (2014). „The Second Dialog State Tracking Challenge“. In: *Proceedings of the SIGDIAL 2014 Conference, The 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 18-20 June 2014, Philadelphia, PA, USA*. The Association for Computer Linguistics, S. 263–272.

Hochreiter, Sepp und Jürgen Schmidhuber (1997). „Long Short-Term Memory“. In: *Neural Computation* 9.8, S. 1735–1780.

Jurafsky, Daniel und James H. Martin (2009). *Speech and Language Processing (Second Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kingma, Diederik P. und Jimmy Ba (2015). „Adam: A Method for Stochastic Optimization.“ In: *Proceedings of the International Conference on Learning Representations, ICLR 2015, San Diego, USA, May 7-9, 2015*. URL: https://arxiv.org/abs/1412.6980v8.

Kumar, Ankit u. a. (2016). „Ask Me Anything: Dynamic Memory Networks for Natural Language Processing“. In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Hrsg. von Maria-Florina Balcan und Kilian Q. Weinberger. Bd. 48. JMLR Workshop and Conference Proceedings. JMLR.org, S. 1378–1387. URL: http://jmlr.org/proceedings/papers/v48/kumar16.html.

Li, Jiwei u. a. (2017a). „Dialogue Learning With Human-In-The-Loop“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*. URL: https://arxiv.org/abs/1611.09823v3.

– (2017b). „Learning Through Dialogue Interactions“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*. URL: http://arxiv.org/abs/1612.04936v4.

Luna, Dominique (2017). *memm2n*. GitHub repository. URL: https://github.com/domluna/memn2n.

McCaffrey, James D. (2013). *Why You Should Use Cross-Entropy Error Instead Of Classification Error Or Mean Squared Error For Neural Network Classifier Training*. URL: https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/.

McCulloch, W. S. und W. Pitts (1943). „A logical calculus of ideas immanent in nervous activity“. In: *Bulletin of Mathematical Biophysics* 5, S. 115–133.

Ovchinnikova, Ekaterina (2016). *KIT_LU_pipeline*. GitHub repository. URL: https://github.com/eovchinn/KIT_LU_pipeline.

PagesJaunes (2017). *textcorpus-generator*. GitHub repository. URL: https://github.com/pagesjaunes/textcorpus-generator.

Rosenblatt, Frank (1957). *The perceptron a perceiving and recognizing automaton*. Techn. Ber. 85-460-1. Cornell Aeronautical Laboratory.

Ruder, Sebastian (2017). „An overview of gradient descent optimization algorithms“. In: *CoRR* abs/1609.04747. URL: http://arxiv.org/abs/1609.04747v2.

Schmidt, Maria und Jan Niehues (2015a). „Introduction to Natural Language Processing and Dialog Modeling“. Vorlesungsfolien.

– (2015b). „Introductions to Spoken Dialog Systems (SDSs)“. Vorlesungsfolien.

Serban, Iulian Vlad u. a. (2017). „A Survey of Available Corpora for Building Data-Driven Dialogue Systems“. In: *CoRR* abs/1512.05742. URL: http://arxiv.org/abs/1512.05742v3.

Shevchuk, Yurii (2016). *Hyperparameter optimization for Neural Networks*. URL: http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html.

Standford Vision Lab (2015). *neural_net2.jpeg*. URL: https://github.com/cs231n/cs231n.github.io/blob/master/assets/nn1/neural_net2.jpeg.

Sukhbaatar, Sainbayar u. a. (2015). „End-To-End Memory Networks“. In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Hrsg. von Corinna Cortes u. a., S. 2440–2448. URL: http://papers.nips.cc/paper/5846-end-to-end-memory-networks.

Veen, Fjodor van (2016). *The Neural Network Zoo*. URL: http://www.asimovinstitute.org/neural-network-zoo/.

Waibel, Alex u. a. (1989). „Phoneme recognition using time-delay neural networks“. In: *IEEE transactions on acoustics, speech, and signal processing* 37.3, S. 328–339.

Wen, Tsung-Hsien, Milica Gasic, Dongho Kim u. a. (2015). „Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking“. In: *Proceedings of the SIGDIAL 2015 Conference, The 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 2-4 September 2015, Prague, Czech Republic*. The Association for Computer Linguistics, S. 275–284.

Wen, Tsung-Hsien, Milica Gasic, Nikola Mrksic u. a. (2015). „Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems“. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. Hrsg. von Lluis Marquez u. a. The Association for Computational Linguistics, S. 1711–1721.

Weston, Jason (2016a). *bAbI Tasks Data 1-20 (v1.2)*. URL: http://www.thespermwhale.com/jaseweston/babi/tasks_1-20_v1-2.tar.gz.

– (2016b). „Dialog-based Language Learning“. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Hrsg. von Daniel D. Lee u. a., S. 829–837. URL: http://papers.nips.cc/paper/6264-dialog-based-language-learning.

Weston, Jason, Antoine Bordes u. a. (2016). „Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016*. URL: http://arxiv.org/abs/1502.05698v10.

Weston, Jason, Sumit Chopra und Antoine Bordes (2015). „Memory Networks“. In: *Proceedings of the International Conference on Learning Representations, ICLR 2015, San Diego, USA, May 7-9, 2015*. URL: http://arxiv.org/abs/1410.3916v11.

Wikimedia Commons (2008). *File:Artificial Neuron Scheme.png*. URL: https://commons.wikimedia.org/wiki/File:Artificial_Neuron_Scheme.png.

– (2010). *File:Unfold through time.png*. URL: https://commons.wikimedia.org/wiki/File:Unfold_through_time.png.

A. Anhang

A.1. Muster der Dialog bAbI Tasks

Die folgenden Muster stellen alle Muster dar, die aus (Weston, 2016a) extrahiert werden können. Elemente in dopellten Spitzenklammern sind Platzhalter für tatsächliche Werte.

A.1.1. System

1. any preference on a type of cuisine
2. api_call «Küchennationalität» «Ort» «Personenanzahl» «Preis»
3. great let me do the reservation
4. hello what can i help you with today
5. here it is «Adresse»
6. here it is «Telefonnummer»
7. how many people would be in your party
8. i'm on it
9. is there anything i can help you with
10. ok let me look into some options for you
11. sure is there anything else to update
12. sure let me find an other option for you
13. what do you think of this option: «Restaurantnamen»
14. where should it be
15. which price range are looking for
16. you're welcome

A.1.2. Nutzer

1. actually i would prefer for «Personenanzahl»
2. actually i would prefer for «Personenanzahl» people
3. actually i would prefer in «Ort»
4. actually i would prefer in a «Preis» price range
5. actually i would prefer with «Küchennationalität» cuisine
6. actually i would prefer with «Küchennationalität» food
7. can you book a table for «Personenanzahl»
8. can you book a table for «Personenanzahl» in «Ort»
9. can you book a table for «Personenanzahl» in «Ort» in a «Preis» price range
10. can you book a table for «Personenanzahl» people
11. can you book a table for «Personenanzahl» people in a «Preis» price range

12. can you book a table for «Personenanzahl» people in a «Preis» price range with «Küchennationalität» cuisine
13. can you make a restaurant reservation for «Personenanzahl» in «Ort»
14. can you make a restaurant reservation for «Personenanzahl» in «Ort» with «Küchennationalität» cuisine
15. can you make a restaurant reservation for «Personenanzahl» in a «Preis» price range in «Ort»
16. can you make a restaurant reservation for «Personenanzahl» people
17. can you make a restaurant reservation in «Ort» for «Personenanzahl» people in a «Preis» price range with «Küchennationalität» food
18. can you provide the address
19. do you have its address
20. do you have its phone number
21. do you have something else
22. for «Personenanzahl» people please
23. for «Personenanzahl» please
24. good morning
25. hello
26. hi
27. i'd like to book a table
28. i'd like to book a table for «Personenanzahl»
29. i'd like to book a table for «Personenanzahl» in «Ort» in a «Preis» price range
30. i'd like to book a table for «Personenanzahl» with «Küchennationalität» food
31. i'd like to book a table with «Küchennationalität» food
32. i am looking for a «Preis» restaurant
33. i love «Küchenmationalität» food
34. i love that
35. in «Ort»
36. in a «Preis» price range please
37. instead could it be for «Personenanzahl»
38. instead could it be for «Personenanzahl» people
39. instead could it be in «Ort»
40. instead could it be in a «Preis» price range
41. instead could it be with «Küchennationalität» cuisine
42. instead could it be with «Küchennationalität» food
43. it's perfect
44. let's do it
45. may i have a table for «Personenanzahl»
46. may i have a table for «Personenanzahl» in a «Preis» price range
47. may i have a table for «Personenanzahl» in a «Preis» price range with «Küchennationalität» cuisine
48. may i have a table for «Personenanzahl» in a «Preis» price range with «Küchennationalität» food
49. may i have a table for «Personenanzahl» people

50. may i have a table in «Ort»
51. may i have a table in «Ort» for «Personenanzahl» in a «Preis» price range
52. may i have a table with «Küchennationalität» food
53. may i have the address of the restaurant
54. may i have the phone number of the restaurant
55. no
56. no i don't like that
57. no thanks
58. no thank you
59. no this does not work for me
60. <SILENCE>
61. thanks
62. thank you
63. that looks great
64. we will be «Personenanzahl»
65. what is the phone number of the restaurant
66. with «Küchennationalität» cuisine
67. with «Küchennationalität» food
68. you rock
69. «Ort» please

A.2. Beispiele aus den bAbI Tasks

Task 1 (Weston, Bordes u. a., 2016):

1. Mary went to the bathroom.
2. John moved to the hallway.
3. Mary travelled to the office.
4. Where is Mary? A: office

Task 19 (Weston, Bordes u. a., 2016):

1. The kitchen is north of the hallway.
2. The bathroom is west of the bedroom.
3. The den is east of the hallway.
4. The office is south of the bedroom.
5. How do you go from den to kitchen? A: west, north
6. How do you go from office to bathroom? A: north, west

A.3. Kommandozeilenargumente von train.py

Argument	Erklärung	Standard
-h, --help	zeige Hilfe an	
--learning_rate RATE	verwende RATE als Lernrate für den Adam Optimierer	0.002
--epsilon EPSILON	verwende EPSILON als ϵ für den Adam Optimierer	10^{-8}
--hops HOPS	verwende HOPS Hops im Memory Network	1
--embedding_size EMBEDDING	verwende EMBEDDING als Embedding-Größe	38
--batch_size SIZE	verwende SIZE als Batch-Größe	32
--random_seed SEED	verwende SEED als random seed für Tensorflow	None
--evaluation_interval_candidates INTERVAL	evaluiere alle INTERVAL Durchläufe das Training, falls als Antworterzeugung Kandidatenauswahl verwendet wird	4
--evaluation_interval_words INTERVAL	evaluiere alle INTERVAL Durchläufe das Training, falls als Antworterzeugung eine der Wort-für-Wort-Verfahren verwendet wird	1
--epochs_candidates EPOCHS	führe EPOCHS Epochen beim Training aus, falls als Antworterzeugung Kandidatenauswahl verwendet	100
--epochs_words EPOCHS	führe EPOCHS Epochen beim Training aus, falls als Antworterzeugung eine der Wort-für-Wort-Verfahren verwendet wird	100
--min_sentence_size SIZE	verwende SIZE als minimale Satzlänge	34
--first_nn_layer_size SIZE	verwende SIZE Neuronen in der ersten verdeckten Schicht des RNNs	50
--second_nn_layer_size SIZE	verwende SIZE Neuronen in der zweiten verdeckten Schicht des RNNs	None
--delayed_words_size SIZE	verwende die SIZE letzten ausgegebene Wörter als Eingabe für das RNN	2
--alpha ALPHA	verwende ALPHA als α bei der SC-LSTM	0,5

--lstm_structure STRUCTURE	verwende STRUCTURE = [cell|hidden|input] für den Typ der LSTM	input
--data_dir DIR	verwende Datensätze gespeichert in DIR	mem2/data/dialog-bAbI-tasks
--task_id ID	trainiere bzw. evaluiere Aufgabe ID der Dialog bAbI Tasks, falls die Option data_name nicht spezifiziert wurde	1
--data_name NAME	trainiere bzw. evaluiere den Datensatz NAME	None
--save	speichere Gewichte beim Training	False
--backend BACKEND	verwende BACKEND = [candidates|nn|lstm|sc-lstm] für die Antworterzeugung	candidates
--evaluate_type EVALUATE_KIND	verwende EVALUATE = [train|evaluate|evaluate_detailed] als Betriebsmodus	train

A.4. Anwendungen

train.py	Ermöglicht das Trainieren und das Evaluieren des Dialogsystems.
Dialogsystem als Django-Webanwendung	Weboberfläche für die Nutzung des trainierten Dialogsystems.
separate_hyperopt.py	Führt die voneinander getrennte Hyperparameteroptimierung durch.
separate_hyperopt_visualizer.py	Visualisiert die Ergebnisse der voneinander getrennten Hyperparameteroptimierung.
tpe_hyperopt.py	Führt die Hyperparameteroptimierung mit dem TPE durch.
tpe_hyperopt_visualizer.py	Visualisiert die Ergebnisse der Hyperparameteroptimierung mit dem TPE.
corpus_generator.py	Bibliothek, die verwendet werden kann, um Musterdialoge mit Parameterwerte zu befüllen.
robot_corpus_generator.py	Generiert die Robotersteuerungsaufgaben.
accuracy_time_checker.py	Führt für die Evaluation mehrere Durchläufe mit den verschiedenen Antwortgenerierungsarten durch.
accuracy_time_visualizer.py	Visualisiert die Ergebnisse des accuracy_time_checker.py.

accuracy_difference_checker.py	Berechnet den Unterschied zwischen der minimalen und maximalen Genauigkeit, bei Durchläufe von gleichen Hyperparameterwerte in der Hyperparameteroptimierung mit dem TPE.
api_call_comparator.py	Berechnet die Anzahl an Dialogen des Validations- und Testdatensatzes, die im Trainingsdatensatz sind und die nicht im Trainingsdatensatz sind.
dialogs_lengths_calculator.py	Berechnet die minimale, durchschnittliche, mediane und maximale Länge der Dialoge der Dialog bAbI Tasks.
restaurant_name_replacer.py	Ersetzt die Restaurantnamen der Dialog bAbI Tasks mit einem einheitlichen Namen gefolgt von einer durchgängigen Nummerierung.
user_vocab_extractor	Berechnet die Länge des Vokabulars des Benutzers von Aufgaben aus den Dialog bAbI Tasks. Optional kann das Vokabular auch ausgegeben werden.

A.5. Musterdialoge des Robotersteuerungsaufgaben

1. move to the LOCATION↹api_call move LOCATION
2. stop↹api_call stop
3. open the OPENABLE↹api_call open OPENABLE
4. give me the MOVABLE↹api_call give MOVABLE
5. close the OPENABLE↹api_call close OPENABLE

1. pour the POURABLE into the POUR↹api_call pour POURABLE POUR
2. stir the STIRABLE↹api_call stir STIRABLE
3. wipe the WIPEABLE↹api_call wipe WIPEABLE

1. hello↹hello
2. put the MOVABLE on the LOCATION↹api_call put MOVABLE LOCATION
3. grasp the MOVABLE from the LOCATION↹api_call grasp MOVABLE LOCATION

1. hello↹hello
2. see you the MOVABLE?↹<silence>
3. move it to the LOCATION↹api_call move MOVABLE LOCATION

1. hello↹hello
2. what a great MOVABLE↹<silence>
3. give me it↹where is it?
4. it is on the LOCATION↹api_call give MOVABLE LOCATION

1. hello↹hello

2. i want a drink↹do you prefer a cold or hot drink?
3. i prefer a hot drink↹do you want coffe or cappuccino or espresso or latte or ristretto or chocolate?
4. i want a HOT_DRINK↹do you want it tiny or small or medium or large or giant?
5. i want it SIZE↹do you want cream?
6. yes↹api_call drink HOT_DRINK hot SIZE cream

1. hello↹hello
2. i want a drink↹do you prefer a cold or hot drink?
3. i prefer a hot drink do you want coffe or cappuccino or espresso or latte or ristretto or chocolate?
4. i want a HOT_DRINK↹do you want it tiny or small or medium or large or giant?
5. i want it SIZE↹do you want cream?
6. no↹api_call drink HOT_DRINK hot SIZE no_extra

1. hello↹hello
2. i want a drink↹do you prefer a cold or hot drink?
3. i prefer a cold drink↹do you want water or lemonade or cola or milk or slush?
4. i want a COLD_DRINK↹do you want it tiny or small or medium or large or giant?
5. i want it SIZE↹do you want a straw?
6. yes↹api_call drink COLD_DRINK cold SIZE straw

1. hello↹hello
2. i want a drink↹do you prefer a cold or hot drink?
3. i prefer a cold drink↹do you want water or lemonade or cola or milk or slush?
4. i want a COLD_DRINK↹do you want it tiny or small or medium or large or giant?
5. i want it SIZE↹do you want a straw?
6. no↹api_call drink COLD_DRINK cold SIZE no_extra

COLD_DRINK = ['water', 'lemonade', 'cola', 'milk', 'slush']
HOT_DRINK = ['coffe', 'cappuccino', 'espresso', 'latte', 'ristretto', 'chocolate']
LOCATION = ['table', 'stove', 'dishwasher', 'fridge', 'sideboard']
MOVABLE = ['chair', 'juice', 'cola', 'lemonade', 'coffee']
OPENABLE = ['fridge', 'cereal']
POUR = ['glass', 'cup', 'bowl', 'mug']
POURABLE = ['water', 'juice', 'cola', 'lemonade', 'coffee']
SIZE = ['tiny', 'small', 'medium', 'large', 'giant']
STIRABLE = ['salad', 'drink']
WIPEABLE = ['sideboard', 'table']

A.6. Modifizierte Dialoge der Dialog bAbI Tasks

1. hi↹hello what can i help you with today
2. can you make a restaurant reservation for four↹i'm on it
3. <SILENCE>↹any preference on a type of cuisine
4. with spanish↹food where should it be
5. madrid↹please which price range are looking for
6. in a expensive price range please↹ok let me look into some options for you
7. <SILENCE>↹api_call spanish madrid four expensive

Äußerung in Zeile 2 ersetzt mit:

- can you make a restaurant reservation for four please ✓ ✓
- can you make a restaurant reservation for four for tonight X ✓
- make a restaurant reservation for four ✓✓
- please make a restaurant reservation for four X ✓

1. hello↹hello what can i help you with today
2. i'd like to book a table↹i'm on it
3. <SILENCE>↹any preference on a type of cuisine
4. with british cuisine↹where should it be
5. in paris↹how many people would be in your party
6. for two people please↹which price range are looking for
7. i am looking for a moderate restaurant↹ok let me look into some options for you
8. <SILENCE>↹api_call british paris two moderate

Äußerung in Zeile 5 ersetzt mit:

- in paris please ✓ ✓
- in the beautiful city paris ✓ ✓
- in the capital of france paris ✓ ✓
- paris ✓ ✓

1. hi↹hello what can i help you with today
2. can you make a restaurant reservation for eight people↹i'm on it
3. <SILENCE>↹any preference on a type of cuisine
4. with spanish food↹where should it be
5. in london↹which price range are looking for
6. in a cheap price range please↹ok let me look into some options for you
7. <SILENCE>↹api_call spanish london eight cheap

Äußerung in Zeile 4 ersetzt mit:

- with spanish food please X ✓
- with tasty spain food X X
- spain food X X
- tasty spain food X X

1. hi↹hello what can i help you with today
2. can you book a table↹i'm on it
3. <SILENCE>↹any preference on a type of cuisine
4. i love italian food↹where should it be
5. in paris↹how many people would be in your party
6. we will be two↹which price range are looking for
7. in a cheap price range please↹ok let me look into some options for you
8. <SILENCE>↹api_call italian paris two cheap

Äußerung in Zeile 7 ersetzt mit:

- cheap price range please ✓ ✓
- i prefer a cheap price range ✓ X
- cheap X X
- cheap price range ✓ ✓

A.7. Trainingsergebnisse

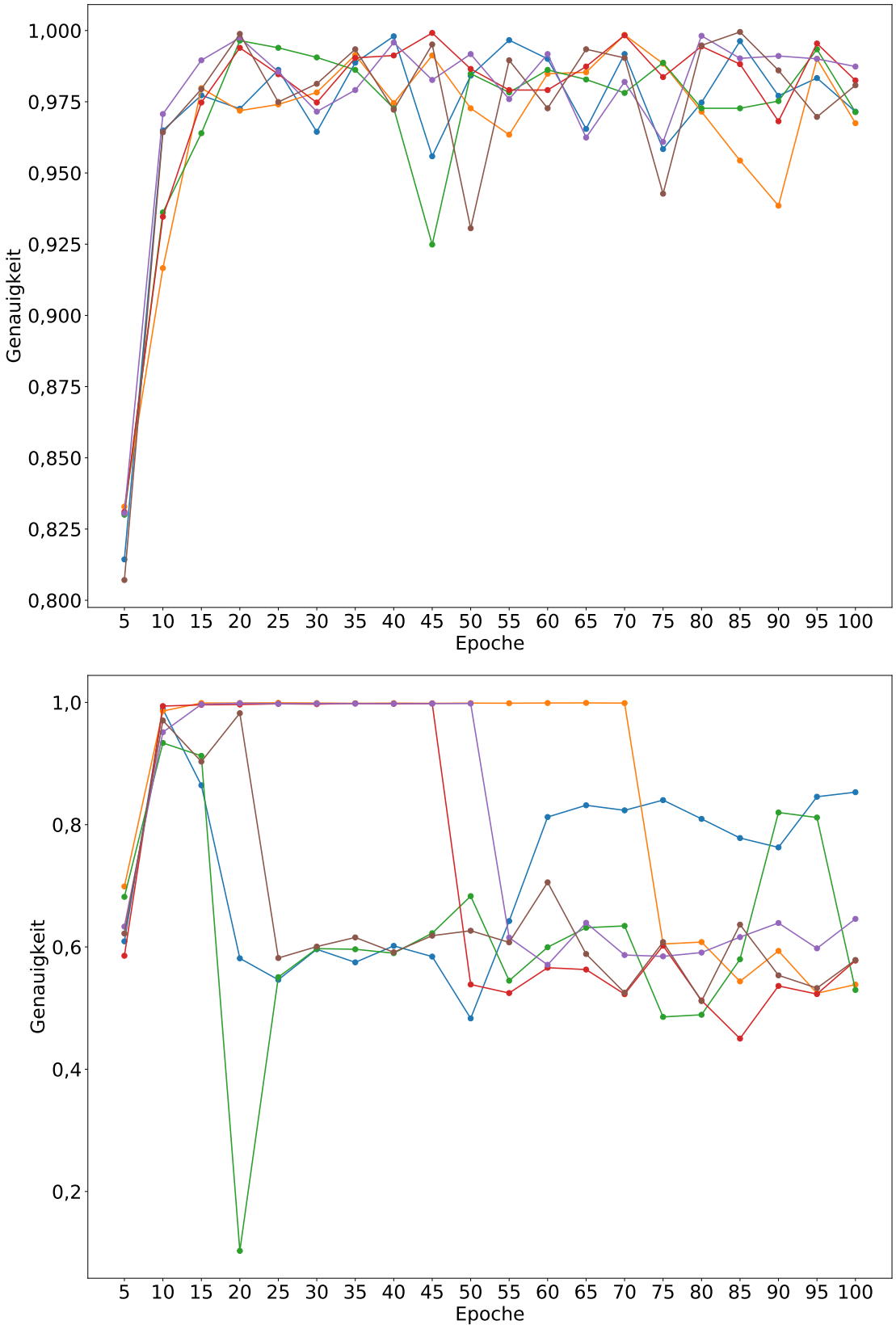

Abbildung A.1.: Aufgabe 1 mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

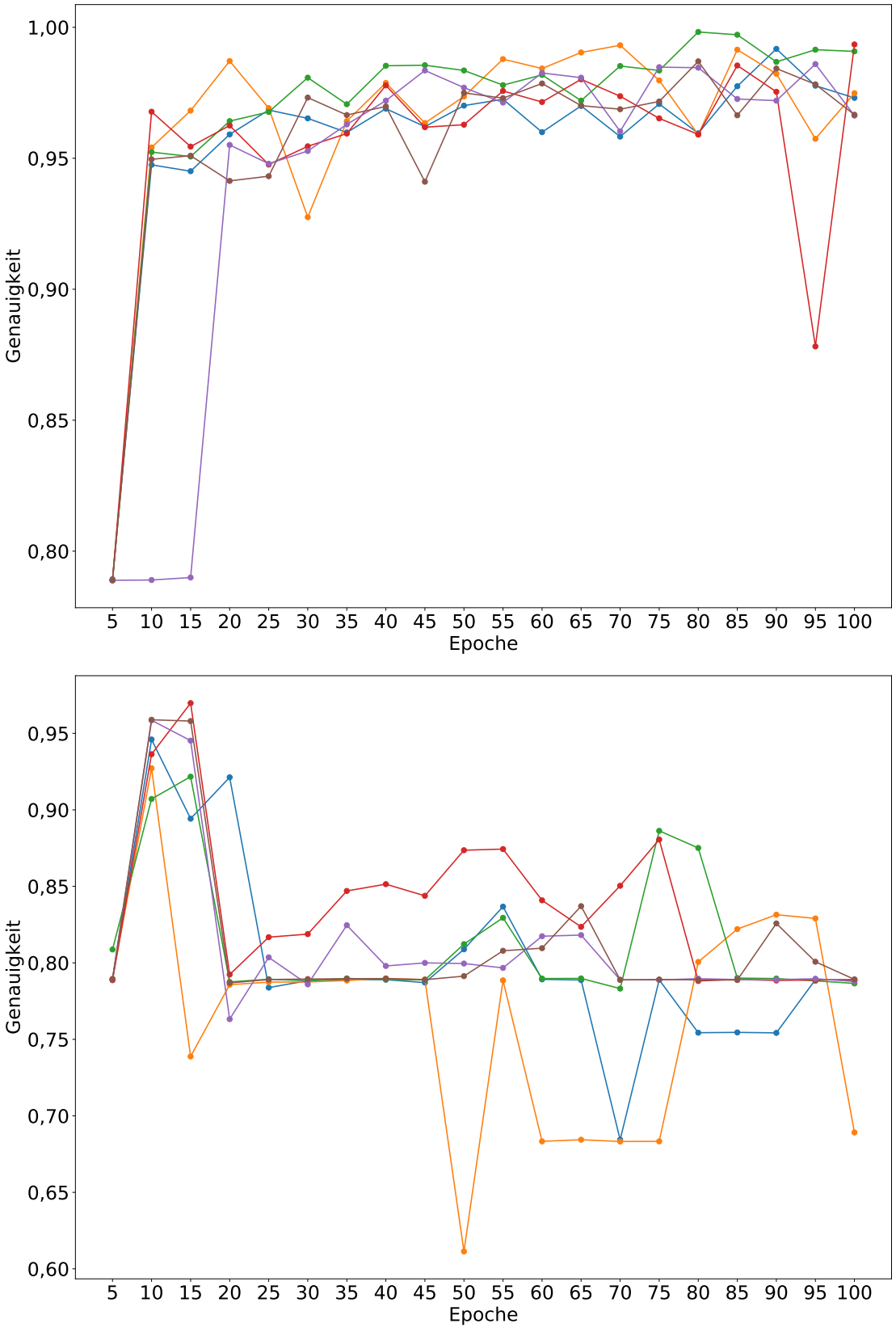

Abbildung A.2.: Aufgabe 2 mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

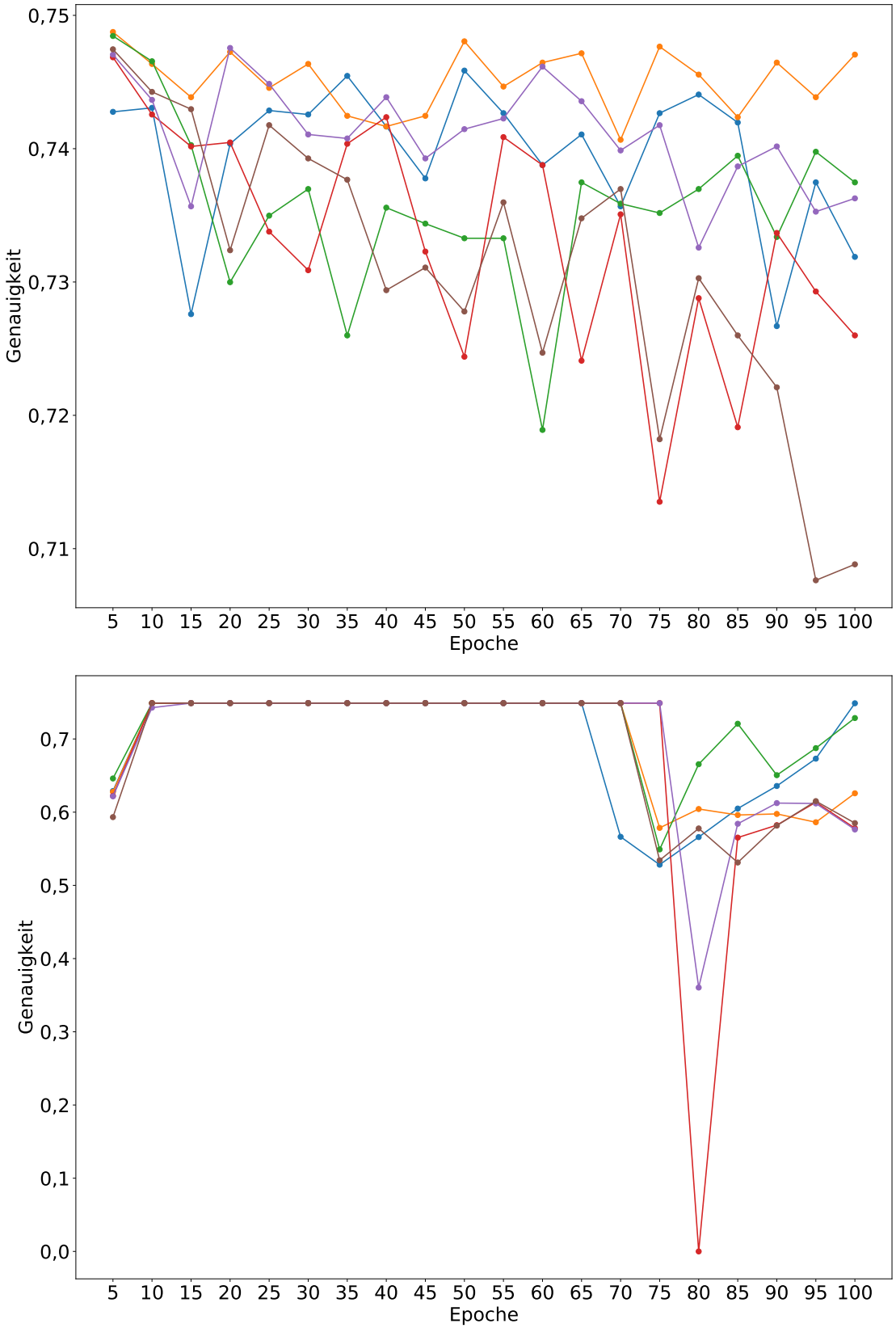

Abbildung A.3.: Aufgabe 3 mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

Abbildung A.4.: Aufgabe 4 mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

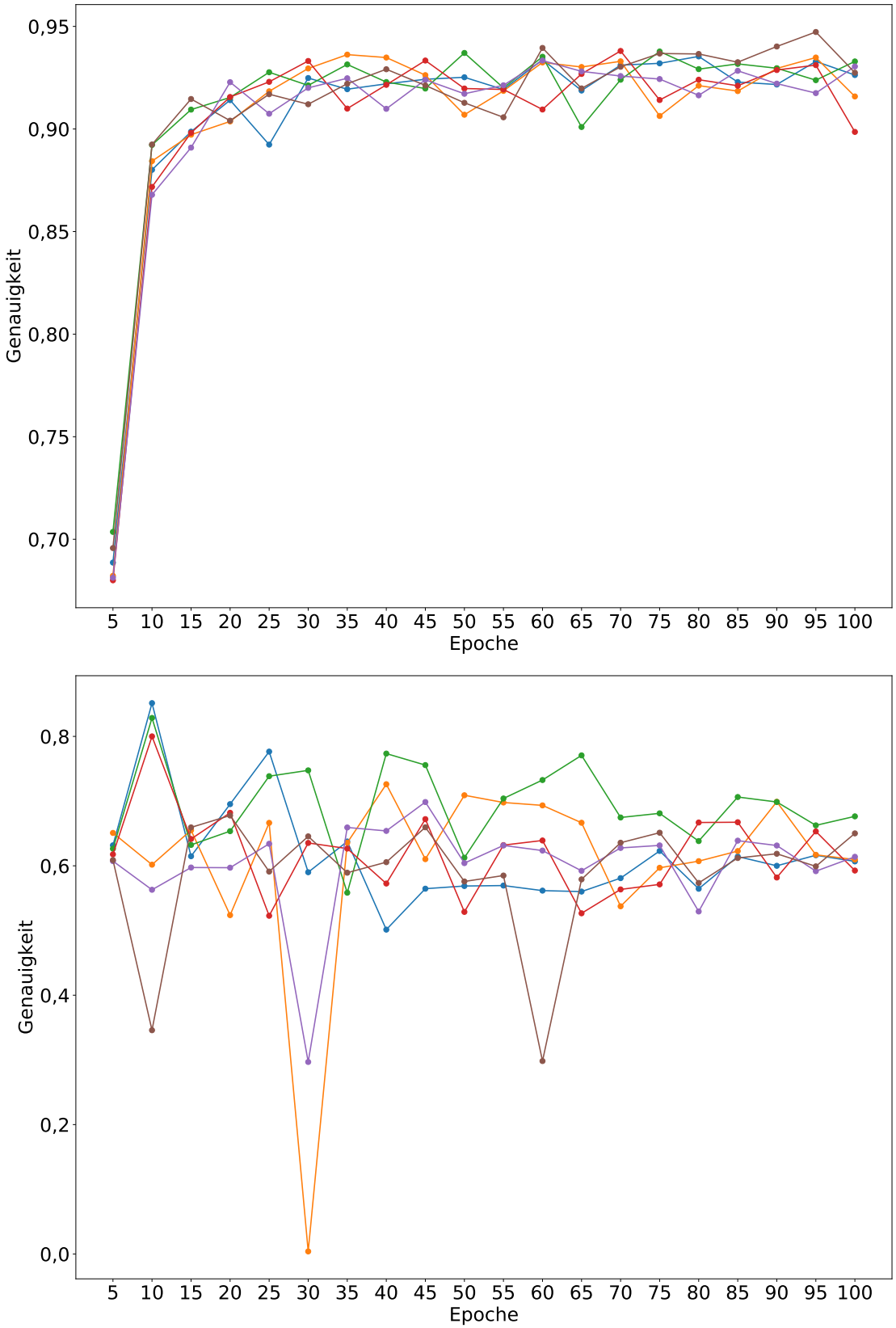

Abbildung A.5.: Aufgabe 5 mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

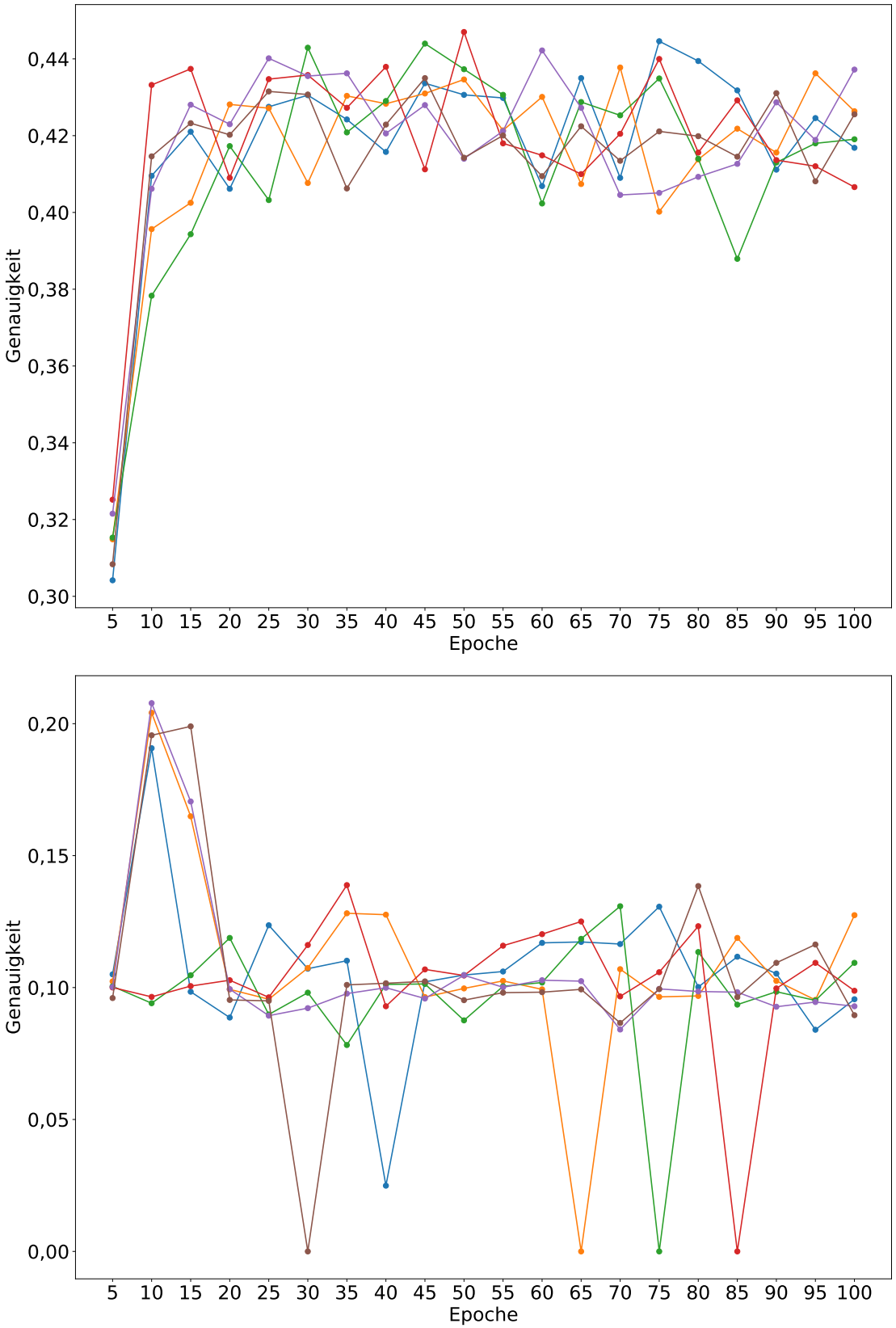

Abbildung A.6.: Aufgabe 6 mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

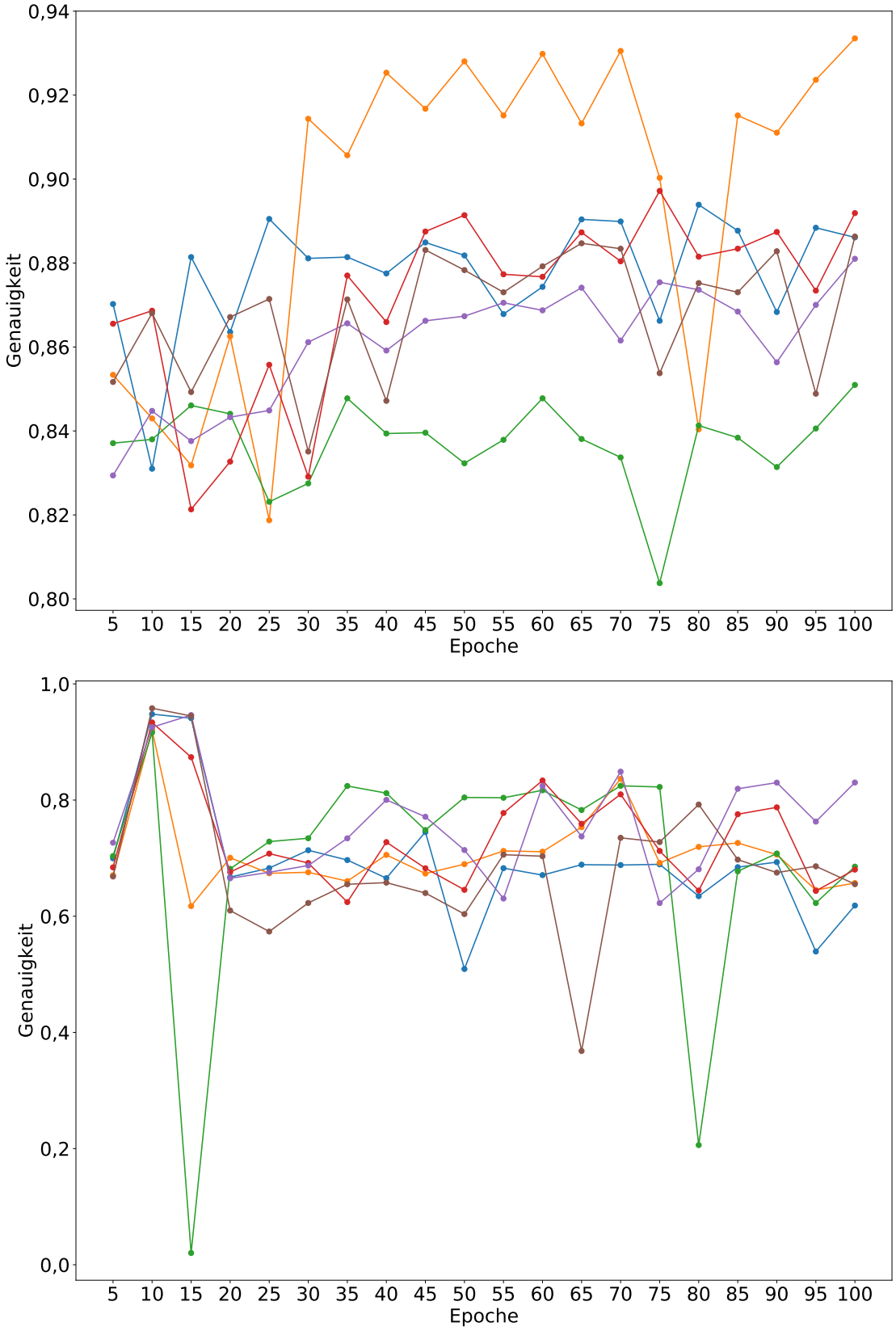

Abbildung A.7.: Aufgabe 3 vereinfacht mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

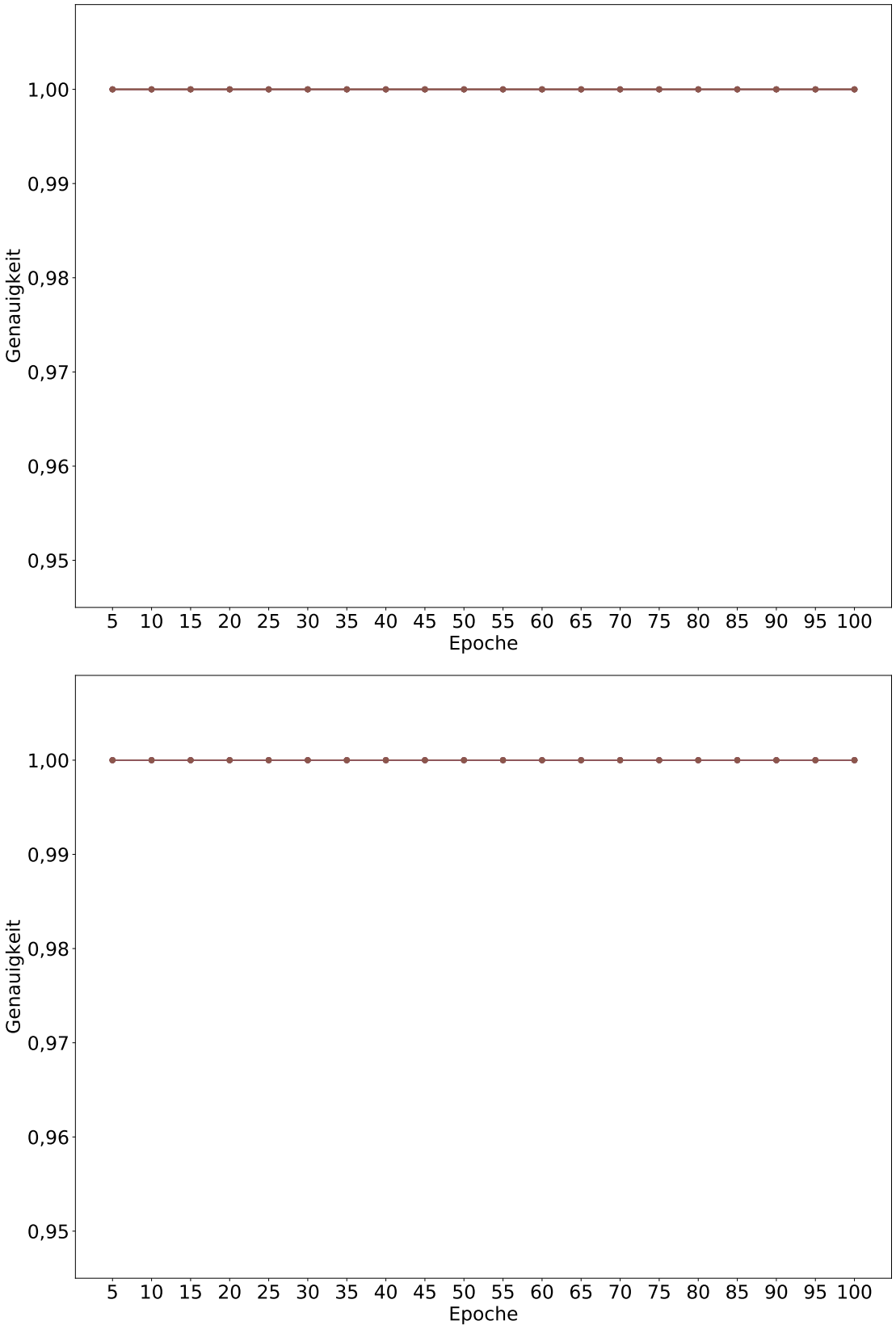

Abbildung A.8.: Aufgabe 4 vereinfacht mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)

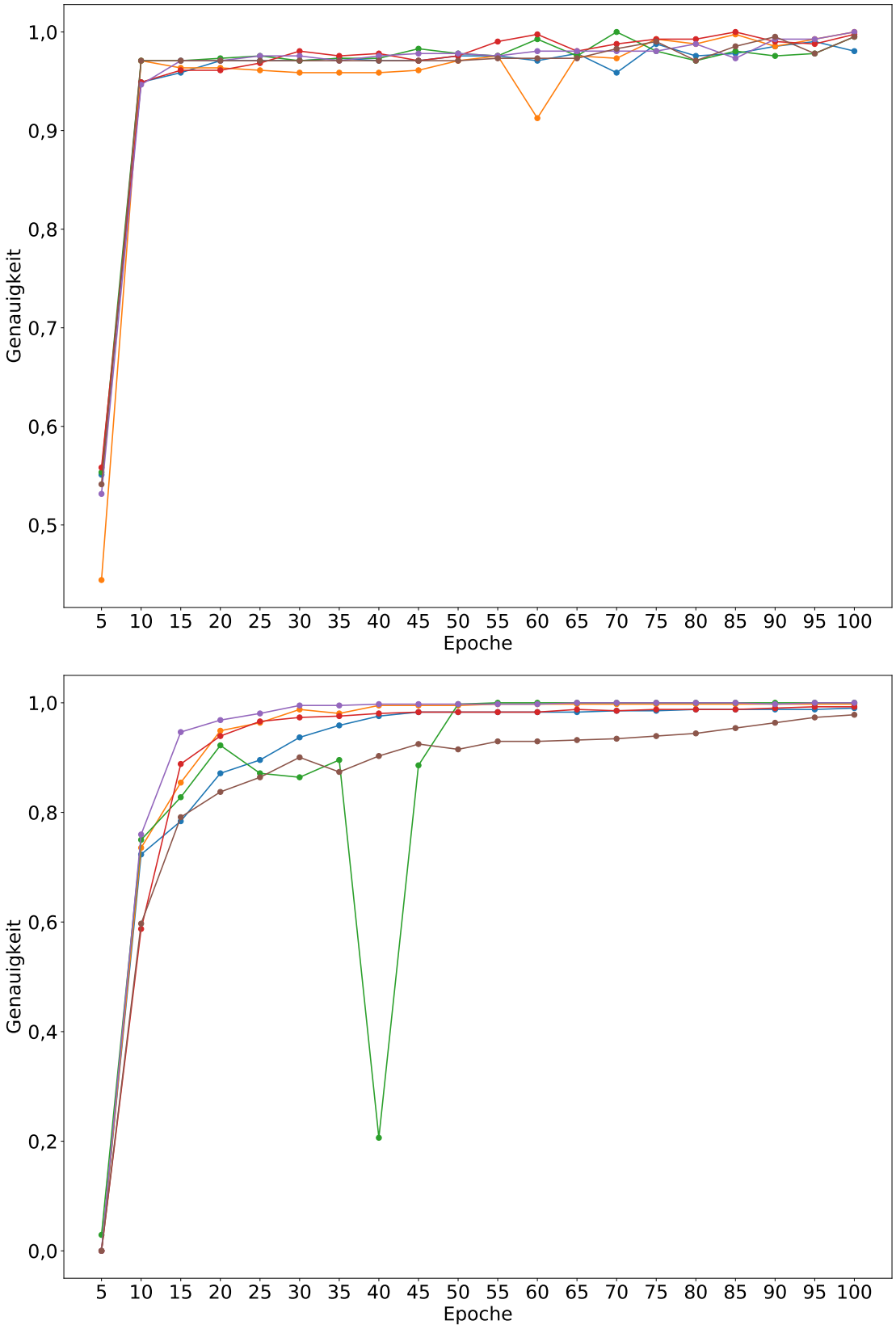

Abbildung A.9.: Robotersteuerungsaufgaben mit der NLG-Komponente Kandidatenauswahl (oben) und klassisches RNN (unten)