



InformatiCup 2020: Pandemie Dokumentation

Team: Nicolas Schaber, Daniel Schulz, Max Schiffer
Hochschule: Duale Hochschule Baden-Württemberg Karlsruhe
Studiengang: Informatik 2017
Kontakt: mail@nicolas-schaber.de

Inhaltsverzeichnis

Einleitung	3
Konzept	4
Grundidee	4
Komponenten	5
Randomizer	6
Datentypen	9
Implementierung	10
Verwendete Software	10
Projektstruktur	10
Verwendung	15
Development	15
Handbuch	17
Fazit	18

Einleitung



Diese Dokumentation beschreibt den Lösungsvorschlag für den InformatiCup 2020 der Gesellschaft für Informatik des Teams Nicolas Schaber, Daniel Schulz und Max Schiffer der DHBW Karlsruhe. Das Repository mit dem Quellcode der Lösung befindet sich hier: <https://github.com/nicosc77/pandemie>.

Aufgabenstellung

Die Aufgabe ist in diesem Repository abrufbar: <https://github.com/informatiCup/informatiCup2020>.

Im Grunde geht es darum, eine Software als **zustandslosen** Webservice zu implementieren, die in einem rundenbasierten Spiel gegen die Auslöschung der Menschheit (Pandemie) kämpft in dem sie versucht mit Aktionen den Spielverlauf einer Simulation zu lenken. Die Simulation dieser Pandemie wird durch ein gegebenes Kommandozeilen-Programm erzeugt.

Konzept



Grundidee

Bevor wir Lösungsansätze für das Problem aufstellen konnten haben wir zunächst die Funktionsweise des Kommandozeilen-Tools und damit des Spielablaufs genauer angesehen. So konnten wir ein Gefühl für die Spieldynamik entwickeln. Durch gezielte Anwendung von Aktionen auf aufgetretene Events haben wir außerdem die Auswirkungen von Aktionen und Events nachvollziehen können.

Da es sich quasi um ein textbasiertes Spiel handelt kann man als menschlicher Benutzer durch Beobachtung Entscheidungen treffen und die Aktion selbst auswählen. Um eine solche Intelligenz in eine Software zu implementieren muss diese als Künstliche Intelligenz in die Anwendungslogik eingebaut werden. Man könnte diese Software dann als einen GameBot bezeichnen, der versucht das Spiel bestmöglich zu spielen.

Wir haben uns dann schließlich dazu entschlossen das vorliegende Problem mit Machine Learning bzw. Deep Learning zu lösen. Deep Learning greift auf künstliche neuronale Netze zurück, die wie das biologische Vorbild der Neuronen im Gehirn angelernt werden können. Ein menschlicher Benutzer mit der Zeit lernt das Spiel immer besser zu Spielen. Bei unserer Software muss dieser Lernprozess durch Training mit sehr großen Datenmengen ebenfalls erfolgen.

Die Anwendung bzw. der Bot muss dafür immer den aktuellen Spielstand wissen. Dieser wird uns vom Kommandozeilen-Tool geliefert. Dieser Spielstand wird als JSON-Objekt ausgegeben. Auf Grundlage von diesem Spielstand muss der Bot eine Aktion auswählen und ausführen.

Das Konzept hierfür besteht darin, zunächst den eingehenden JSON Datensatz der aktuellen Spielrunde zu analysieren. Wir haben dabei an einen Algorithmus gedacht der jeder einzelnen Stadt aufgrund der vorliegenden Merkmale und Events einen numerischen Wert als Score zuweist, der ausdrückt wie gefährlich diese Stadt für den weiteren Ausbruch und Verbreitung von Viren ist. Mit diesem Score lässt sich im Spielstand die Stadt mit dem schlechtesten Score finden, also die Stadt mit der größten Gefahr. Genau diese Stadt könnte dann vom neuronalen Netz bearbeitet werden, in dem eine Aktion gefunden wird, die diesen Score verbessert und die Gefahr verringert.

Bevor das neuronale Netz der Anwendung jedoch diese Entscheidung treffen kann muss es trainiert werden. Für das Training wiederum müssen Trainingsdaten gesammelt werden. Hierbei ist uns der Score ebenfalls behilflich. Nach einer zufälligen Anwendung einer Aktion auf eine Stadt lässt sich danach die Differenz des Scores berechnen. Die Differenz liefert eine Rückmeldung darüber, ob die ausgewählte Aktion gut oder schlecht war. War diese Aktion tatsächlich erfolgreich kann die Aktion verknüpft mit der erfolgreichen Aktion den Trainingsdaten hinzugefügt werden. Erkennt das neuronale Netz dann später eine Stadt mit ähnlichen Merkmalen können dadurch vereinfacht ausgedrückt Rückschlüsse auf die nun vermutlich passende Aktion gezogen werden.

Deswegen bietet das Konzept unserer Software zwei grobe Funktionen: Das Trainieren des Netzes und das Testen des Netzes. Für die eigentliche Produktivversion unserer Anwendung ist nur die letztere Funktion wichtig, da für die Abgabe der Lösung bereits ein trainiertes Netz zur Verfügung gestellt wird.

Komponenten

Im folgenden sind die wichtigsten Komponenten unseres Konzepts etwas detaillierter beschrieben.

Kommandozeilen-Tool

Das für den Wettbewerb zur Verfügung gestellte Kommandozeilen-Tool repräsentiert als Client die Spiellogik welche von der zu implementierenden Anwendung getroffene Aktionen ausführt und dazu den nächsten Spielzustand berechnet. Das Tool sendet die Spielzustände an unsere Software und wartet auf die Rückgabe einer Aktion. Ist das Spiel gewonnen oder verloren terminiert das Tool. Eine Spielrunde kann sich jedoch über mehrere Iterationen erstrecken, da eine Spielrunde von der Software beispielsweise bei unzureichender Anzahl an Punkten aktiv beendet werden muss. Es können also mehrere Aktionen in einer Runde durchgeführt werden.

Webservice-Endpoints

Die Anwendung soll als Web-Service zur Verfügung gestellt werden. Ein- und Ausgabe von Daten soll also über ein Netzwerk erfolgen. Die von der Anwendung im Netzwerk veröffentlichen HTTP-Endpunkte sind hierfür als Kommunikationsbrücke mit dem Tool zuständig. Beim Eintreffen des Spielzustands wird dieser außerdem vom Transportformat JSON in Python Objekte umgewandelt.

In der Produktivversion der Software sind die HTTP-Endpoints für das Sammeln von Daten und das Trainieren des neuronalen Netzes gesperrt. Nur der Endpoint mit der Root-URL „/“ ist verfügbar, welche die eigentliche Funktionalität zur Verfügung stellt.

Sowohl beim Training als auch beim Testen wird anschließend von hier aus der gesamte Spielzustand an das Scoring übermittelt.

Scoring

Das Scoring bewertet jede Stadt des Spielzustands. Dabei werden sowohl die eigenen Werte der Stadt und deren Events als auch weltweite Werte und die Beziehungen zwischen den Städten analysiert und in Werte zusammengefasst. Am Ende des Scorings steht eine sortierte Liste der Städte zur Verfügung. Aus dieser Liste wird später das erste Objekt und damit die Stadt mit dem schlechtesten Score entnommen.

Hygiene, Wirtschaft, Politische Stabilität und Achtsamkeit der Einwohner einer Stadt werden durch „--“, „-“, „o“, „+“ oder „++“ dargestellt. Diese Werte lassen sich deshalb relativ einfach in einen numerischen Wert ausdrücken. Ebenfalls kann die Anzahl der Flugverbindungen und die Größe der Population einfach abgezählt werden.

Etwas schwieriger, jedoch für die sinnvolle Bestimmung des Scores unabdingbar ist die Analyse der Events einer Stadt. Wir sind dazu verschiedene Strategien durchgegangen, haben uns jedoch dann dazu entschieden auf Grundlage von vorliegenden Events die gerade genannten Werte zu modifizieren. Eine Ausnahme sind dabei Events die mit Viren zu tun haben, also bspw. Ausbruch oder Bio-Terrorismus. Dafür wurde ein extra Wert eingeführt. Schließlich werden alle einzelnen Scores der Ausprägungen einer Stadt addiert. Je höher dieser Score, desto gefährlicher ist die Stadt für direkte und indirekte Nachbarstädte und hat damit unserem Konzept zur Folge die größte Relevanz für den Ausgang der Spielrunde.

Nachdem alle Städte des Spielstands einen Score erhalten haben kann die Stadt mit dem schlechtesten Score (z.B. siehe Abbildung 1) herausgenommen werden. Diese Stadt (`top_city`) ist von nun an sowohl beim Training als auch bei der Benutzung des Netzes Gegenstand der weiteren Bearbeitung.

City: Kinshasa

Score: 7.8181694453895885

Population: 2939

Events: Outbreak of Plorps since round 6 with ~83.8% distribution

Hygiene: -

Government: -

Awareness: -

Abbildung 1: Ausgewählte Stadt mit dem schlechtesten Score einer zufälligen Runde

Ab diesem Punkt unterscheidet sich der weitere Programmablauf von der aktuellen Phase. Bei der Sammlung von Trainingsdaten geht es wie folgt weiter:

Collection(Datensammlung)

Beim Training wird die gewählte Stadt nun an die Collection gegeben. Aktiv wird hier nur etwas unter bestimmten Bedingungen unternommen.

1. In der vorangegangenen Runde muss eine Aktion ausgeführt worden sein, die nicht EndRound ist.
2. Diese Aktion hat den Score der bearbeiteten Stadt verbessert

Liegt also eine Aktion der letzten Runde vor, die nicht Endround ist und den Score verbessert kann der Datensatz zu den Trainingsdaten aufgenommen werden.

Nicht nur die Aktion wird festgehalten, sondern natürlich auch noch die bearbeitete Stadt. Nur so kann das neuronale Netz lernen welche Eigenschaften einer Stadt sich am besten mit welcher Aktion behandeln lassen kann. Hat eine Stadt zum Beispiel viele Einwohner und eine schlechte Wirtschaft könnte es vielleicht effektiv sein, die Wirtschaft zu verbessern.

Wichtig ist jedoch, dass eine Runde nach einer Aktion beendet wird. Die Umrechnung der ausgewählten Aktion auf den Spielstand erfolgt beim Kommandozeilen-Tool nämlich erst nach Beendigung der Runde. Damit die Auswirkungen von Aktionen also isoliert getestet werden können darf bei der Collection nur eine Aktion pro Runde getätigt werden. Deshalb gibt es zu den zwei Bedingungen zusätzlich noch Mechanismen, die dies überwachen.

Unabhängig davon ob ein Datensatz gespeichert wurde muss für die nächste Runde eine Dazu wird anschließend der Randomizer mit der aktuellen `top_city` aufgerufen.

Im Gegensatz zum Funktionsumfang der Produktivversion ist die die Kommunikation zwischen Tool und Anwendung bei der Collection nicht zustandslos.

Randomizer

Hier wird für diese Stadt eine zufällige Aktion ausgewählt. Dabei werden die erforderlichen Parameter bestimmt und die Machbarkeit der Aktion überprüft, um Fehler des Tools auszuschließen. Der Randomizer gibt die Aktion schließlich an den Endpunkt zurück. Umgewandelt in JSON wird die Aktion schließlich an das Tool zurückgesendet.

Sind genug Trainingsdaten gesammelt worden können Predictions (Vorhersagen) mithilfe eines Modells getroffen werden. Dazu der weitere Programmablauf:

Net

Nachdem mithilfe der Collection genug Trainingsdaten gesammelt wurden kann ein neuronales Netz gebaut werden und mit den Trainingsdaten trainiert werden.

Das neuronale Netz gibt für eine ausgewählte Stadt (`top_city`) dann eine Liste zurück, die für jede verfügbare Aktion beschreibt, wie wahrscheinlich es ist dass diese Aktion erfolgreich sein wird. Da diese Aktion jedoch nur als Zahl codiert zurückgegeben wird, muss mithilfe des Parsers ein vollwertiges Aktions-Objekt erzeugt werden.

Parser

Hierfür übersetzt der Parser die als Zahl (0-9 für 10 Aktionencodierungen) codierte Aktion nicht nur einfach in ein Aktions-Objekt, sondern überprüft Randbedingungen, wie die Machbarkeit dieser Aktion oder die verfügbare Punktzahl. Falls es nicht möglich oder sinnvoll ist die gegebene Aktion auszuführen wird je nach Situation die Aktion mit der nächst höchsten Wahrscheinlichkeit überprüft oder die Runde wird beendet.

Ein Sonderfall stellen die Aktionsgruppen für Medizin und Gegenmittel dar. Es gibt zu diesen beiden Gruppen jeweils eine Aktion Deploy und eine Aktion Develop. Bei der Speicherung der Trainingsdaten bekommen beide Aktionen das gleiche Label, sind also für das Netz nicht zu unterscheiden. Wird dieses Label also ausgewählt wird überprüft ob die Medizin bzw. das Gegenmittel bereits zur Verfügung steht, schon in Entwicklung ist oder keins von beiden. Abhängig davon wird die jeweilige anwendbare Aktion ausgeführt.

Außerdem werden die Parameter für die Aktionen ermittelt wie Anzahl der Runden oder zu behandelte Flugverbindung oder Virus.

Auf der nächsten Seite ist das Schaubild (Abbildung 2) der Komponenten dargestellt. Das Schaubild besteht aus einer linken und einer rechten Seite. Die linke Seite (Training) befasst sich mit der Sammlung von Trainingsdaten. Die rechte Seite (Testing) stellt die produktive und reguläre Verwendung der Software nach dem Training (Siehe roter Pfeil) dar.

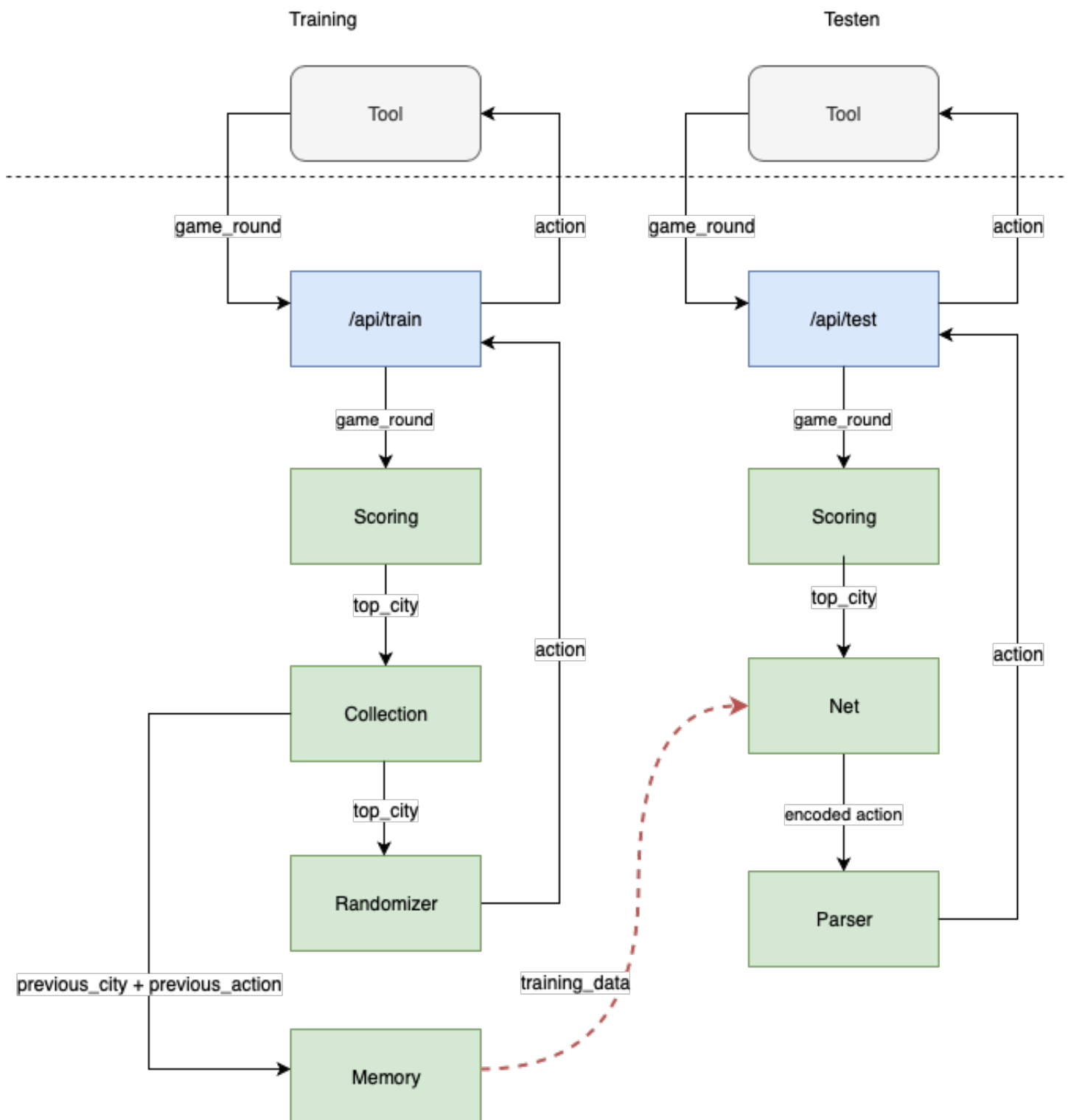


Abbildung 2: Übersicht über die Komponenten der Software

Datentypen

Im Schaubild ist außerdem der Informationsfluss der wichtigsten Datentypen während dem Programmablauf eingezeichnet. Dieser werden hier noch einmal kurz erläutert.

game_round

Ein `game_round` Objekt stellt einen Spielzustand dar. Dieser Spielzustand wird vom Tool im Laufe des Spiels immer wieder erzeugt. Dabei wird vom Tool ein JSON Objekt erzeugt und ausgegeben. Von unserer Software wird dieses Objekt zur internen Verarbeitung in ein Objekt der Python-Klasse `GameRound` umgewandelt.

action

Ein `action` Objekt ist die Darstellung einer ausgewählten Aktion aus der Liste der verfügbaren Aktionen. Diese Objekte werden von unserer Software mit den notwendigen Parametern erzeugt und zuletzt in ein JSON Objekt codiert und zurück an das Tool gesendet.

encoded_action

Das neuronale Netz kann nur die Art der Aktion bestimmen. Diese Art wird in `encoded_action` als Zahl repräsentiert.

top_city

Das `top_city` Objekt wird nach dem Scoring ermittelt und stellt eine Stadt aus der Liste der Städten des aktuellen `game_round` Objekt dar. Diese Stadt ist die am gefährlichsten eingestufte Stadt des Spielstands. Für das neuronale Netz wird das Objekt in ein Array aus numerischen Werten umgewandelt.

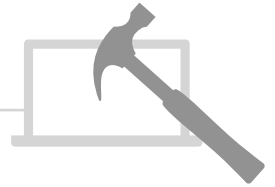
previous_city + previous_action

Diese beiden Objekte haben den gleichen Aufbau wie `top_city` bzw. `action`, sind jedoch dazu da, die Werte des jeweilig vorhergehenden Spielzustand zu speichern. Das ist bei der Sammlung von Trainingsdaten für die Bewertung eines Spielzugs essentiell.

training_data

Die gesammelten Trainingsdaten werden in einer Map gespeichert. Der vorhergehende Zustand der bearbeiteten Stadt und die ausgeführte Aktion werden darin festgehalten.

Implementierung



Verwendete Software

Wir haben uns für Python als Grundlage entschieden. Darüber hinaus verwenden wir eine Kombination aus dem Web-Service Framework Flask¹ und der Bibliothek Tensorflow² für die Implementierung des neuronalen Netzwerks. Mithilfe dieser Komponenten lässt sich ein robuster Web-Service für die Lösung des Problems mit Deep Learning bauen.

Zur einfacheren Implementierung von Tensorflow haben wir auf Keras³ gesetzt.

Projektstruktur

Dockerfile

Das Dockerfile enthält die für Docker notwendigen Informationen zum Bauen eines Containers.

cloudbuild.yaml und kubernetes directory

Diese Dateien enthalten die Konfiguration für das automatische Deployment auf der Kubernetes Engine auf der Google Cloud Plattform.

Pipfile

Das Pipfile hält alle notwendigen Abhängigkeiten des Python Projekts fest. mithilfe von pipenv, einem alternativen Paketmanager für Python können diese Abhängigkeiten aufgelöst werden.

app.py

Hierbei handelt es sich um das Modul mit dem die Anwendung gestartet wird. Mit der Funktionalität von Flask werden hier außerdem die HTTP-Endpunkte zur Verfügung gestellt. Je Endpunkt werden in der zugehörigen Methode die weiteren Komponenten der Software angesprochen.

bin directory

Die Binärdateien des Kommandozeilen-Tools befinden sich in diesem Verzeichnis. Diese werden während dem Training und dem Testen während der Entwicklung von der Anwendung automatisch ausgeführt. In der Produktivversion sind diese nicht benutzbar.

model package

Dieses Python package enthält die Klassen mit denen aus den eingehenden JSON-Daten das Objekt für den Spielzustand erzeugt werden. Das erleichtert intern den Umgang mit den Datensätzen. Das Package ist seinerseits in 4 Teile gegliedert.

actions.py beinhaltet die Klassen der Aktionen die von der Anwendung ausgelöst werden können. Neben dem Konstruktor haben diese Klassen Methoden zur Erzeugung der JSON-Ausgabe wie sie das Kommandozeilen-Tool versteht und zur Ausgabe des Labels der Aktion für das neuronale Netz. (Siehe Abbildung 3) Zudem eine Methode, die jeweils zurück gibt wie viele Punkte für die Aktion erforderlich sind. Bei Aktionen, bei denen diese Punktzahl variabel ist, werden diese mithilfe der Rundenanzahl berechnet.

Im Modul events.py sowie in gameround.py und city.py sind die notwendigen Datenstrukturen angelegt, um aus dem JSON Datensatz Objekte zu generieren.

¹ <https://www.palletsprojects.com/p/flask/>

² <https://www.tensorflow.org/>

³ <https://keras.io>

```

class LaunchCampaign:

    def __init__(self, city=None):
        if city is not None:
            self.city = city.name

    def getMessage(self):
        return '{"type":"launchCampaign", "city":"' + str(self.city) + '"}'

    def getLabel(self):
        return [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

    def getPoints(self):
        return 3

```




Abbildung 3: Klasse der Aktion „LaunchCampaign“

net package

Dieses Package ist für den Deep Learning Teil zuständig. Datensammlung (Collection), das Bauen, Trainieren und Verwenden des neuronalen Netzes für die Vorhersage der auszuführenden Aktion bzw. die Anwendungslogik von Keras und Tensorflow ist hier zu finden. Gebaut und Trainiert wird das Netz mithilfe der Keras Sequential⁴ API. (Abbildung 4)

```

model = Sequential()
model.add(Dense(128, input_dim=input_size, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(output_size, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

x = np.array([i[0] for i in training_data]).reshape(-1, len(training_data[0][0]))
y = np.array([i[1] for i in training_data]).reshape(-1, len(training_data[0][1]))

model = build_model(input_size=len(x[0]), output_size=len(y[0]))

model.fit(x, y, batch_size=6000, validation_split=0.25, epochs=epochs)

```

Abbildung 4: Workflow zum Bauen und Trainieren mit dem Keras Sequential Modell

⁴ <https://keras.io/models/sequential/>

Anschließend kann wie im Konzept bereits dargestellt das neuronale Netz (In diesem Fall die Variable `model`) mit der Methode `predict` benutzt werden, um Vorhersagen für die bestmögliche Aktionen zu treffen. (Abbildung 5)

```
observation = cityprocessor.process_city(top_city)
result = model.predict(observation.reshape(-1, len(observation)))[0]
```

Abbildung 5: Funktionsweise von `model.predict`

Das Ergebnis ist die Liste mit den Wahrscheinlichkeiten zu jeder Aktion, die aussagt wie sicher das neuronale Netz ist, dass diese Aktion am sinnvollsten ist.

Zur Einfachheit wird das aktuelle vortrainierte Modell für die Produktiv-Version lokal in einem Unterverzeichnis gesichert. Dabei wird die Konfiguration des Modells in einer JSON Datei abgelegt und die serialisierten Gewichte als h5 Datei. Bei Neustart der Anwendung werden diese Dateien beim ersten Aufruf der Anwendung wieder eingelesen wenn kein neues Training gestartet wird.

processors package

Neben den Dateien für das neuronale Netz befinden sich im processors package die wichtigsten Komponenten der Anwendung. Hier befinden sich unter anderem die Skripte zur Vor- und Aufbereitung der Daten. Dazu zählt auch das Scoring, welches jeder Stadt wie bereits erwähnt einen Wert zuordnet.

Ebenfalls wichtig ist der ActionRandomizer, der für die zufälligen Aktionen für die Sammlung der Trainingsdaten generiert. Abgesehen von der zufälligen Auswahl einer Aktion wird überprüft, ob diese Aktion überhaupt auf die Stadt anwendbar ist. Das ist zum Beispiel dann nicht der Fall, wenn eine dauerhaft Aktion bereits auf diese Stadt angewendet wurde, wenn die Punkte nicht ausreichen, oder kein Virus ausgebrochen ist, für das man beispielsweise ein Gegenmittel austeilen wollte.

Zur Datensammlung ebenfalls wichtig ist der CityProcessor, der mit einem ähnlichen Algorithmus wie das Scoring die Merkmale einer Stadt in ein Array zusammenfasst, welches von Tensorflow verstanden werden kann.

Schließlich ist der ActionParser für die Auswertung der Ergebnisse zuständig. Während das sortierte Array mit den vorhergesagten Aktionen abgearbeitet wird, werden ähnlich wie beim Randomizer die Anwendbarkeit der Aktionen überprüft. Eher unsinnige Vorschläge werden dabei ignoriert, beispielsweise die Schließung einer Flugverbindung wenn der Flughafen bereits geschlossen ist. In dieser Weise wird die Intelligenz des neuronalen Netzes zusätzlich mit hart codierten Bedingungen erweitert. Zudem werden bei Aktionen welche die Angabe einer Rundenanzahl erfordern die maximal mögliche Anzahl auf Grundlage der verfügbaren Punkte berechnet. (Siehe beispielsweise Abbildung 6)

```
elif action_number == 7:
    # Action: Close Airport

    if any(isinstance(x, AirportClosed) for x in city.events):
        # Pass if airport already closed
        pass
    else:
        # Calculating maximal amount of rounds
        rounds = CloseAirport.calculateRounds(points)

        # Preparing action
        action = CloseAirport(city, rounds)

        if action.getPoints() < points:
            # Return the action if enough points available
            return action
        else:
            # Saving the points if not enough available
            return EndRound()
```




Abbildung 6: Auswertung der vorhergesagten Aktion „Close Airport“

In der folgenden Abbildung ist die gesamte Projektstruktur des Projekts dargestellt:

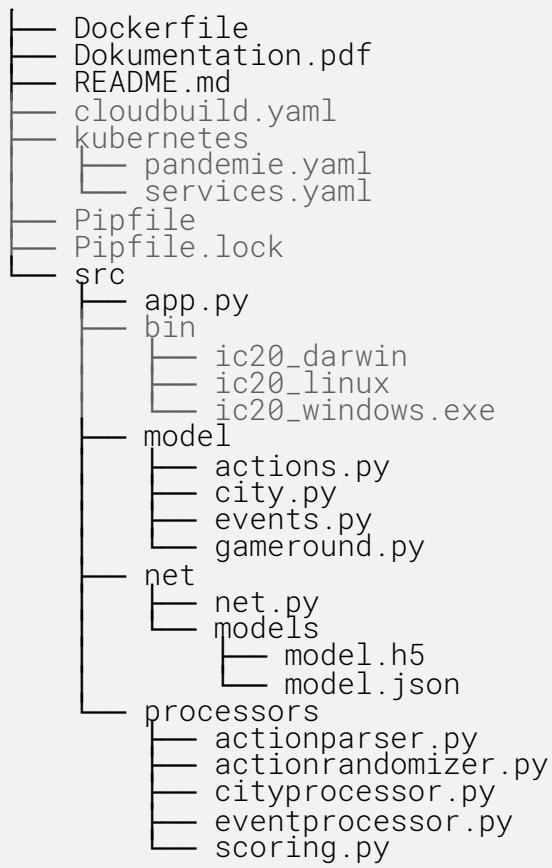
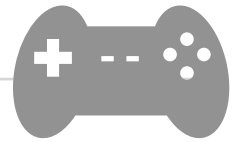


Abbildung 7: Projektstruktur des Programmverzeichnis

Verwendung



In diesem Kapitel wird zunächst beschrieben, wie wir die Software während der Vorbereitung genutzt haben, um eine Datensammlung durchzuführen und das Modell zu trainieren, welches in der Produktivversion bereitgestellt wird.

Anschließend folgt das kurze Handbuch zur Verwendung der Produktivversion, wie es auch im README zu sehen ist.

Development

Sammeln von Trainingsdaten und Trainieren des Netzes

Mithilfe von zusätzlichen HTTP-Endpoints hat man als Entwickler die Möglichkeit per POST das Training zu starten.

```
curl -X Post http://localhost:5000/dev/train?count=100&epochs=250
```

Durch die Angabe der Parameter „count“ und „epochs“ lässt sich während der Laufzeit festlegen wie viele Spiele zum Sammeln verwendet werden sollen und mit wie vielen Epochen das Netz anschließend trainiert werden soll. Für das Sammeln der Trainingsdaten wird das Kommandozeilen-Tool mittels einer Schleife so oft wie festgelegt ausgeführt.

Im Folgenden Auszug ist sichtbar, wie 3 Spielrunden in der Datensammlung durch die einzelnen Komponenten bearbeitet werden.

```
Receiving request to perform model training
Collecting data with 100 rounds...
0 of 100 games played

Receiving request to collect round 1 of a game
First round detected, passing
127.0.0.1 - - [08/Jan/2020 22:42:22] "POST /api/collect HTTP/1.1" 200 -

Receiving request to collect round 2 of a game
New round detected
No valid action data of last round available
Randomizing next action
Action is: {"type":"callElections", "city":"منعاء"}
127.0.0.1 - - [08/Jan/2020 22:42:22] "POST /api/collect HTTP/1.1" 200 -

Receiving request to collect round 2 of a game
Same round detected, ending round to complete collection of data
127.0.0.1 - - [08/Jan/2020 22:42:22] "POST /api/collect HTTP/1.1" 200 -

Receiving request to collect round 3 of a game
New round detected
Valid action data of last round available, checking if positive effect
Action {"type":"callElections", "city":"منعاء"} had positive effect
(3.835369570739142), adding to training data
Randomizing next action
Action is: {"type":"developMedication", "pathogen":"ϕthisis"}
127.0.0.1 - - [08/Jan/2020 22:42:22] "POST /api/collect HTTP/1.1" 200 -
```

Abbildung 8: Auszug des Logs bei der Datensammlung

Beim letzten Request in Abbildung 8 zum Beispiel hat die Anwendung einen Datensatz zur getätigten Aktion der letzten Spielrunde gefunden. Die Überprüfung auf Verbesserung des Scores hat ergeben, dass sich der Score um ca. 3.8 Punkte verbessert hat. Die Kombination aus angewendeter Aktion und beobachteter Stadt wird also gespeichert.

Nach Abschluss der Datensammlung wird das Netz zuerst gebaut und dann trainiert (Abbildung 9). Anschließend wird das Modell gespeichert, damit bei einem Neustart der Anwendung das trainierte neuronale Netz wieder eingelesen werden kann.

```
Collecting complete
Starting training with 250 epochs...
Training model
Building model
Fitting model with 250 epochs...
```

```
Epoch 250/250
26/26 [=====] - 0s 49us/step - loss: 0.0233
Training complete
Saved model to disk
```

Abbildung 9: Auszug des Logs bei Bau und Training des Netzes

Testen des Netzes

Ist das Netz trainiert lässt sich die eigentliche Funktionalität der Anwendung testen. Hierfür gibt es für die Entwicklung ebenfalls einen Endpoint mit dem sich gleich mehrere Spiele hintereinander spielen lassen.

```
curl -X Post http://localhost:5000/dev/test?count=20
```

```
Receiving request to perform model testing
Model already loaded
Testing with 20 rounds...
Receiving request to play round 1 of a game
Picking top city
Processing city
Predicting action
Processing results from prediction
Predicted actions: [8, 0, 5, 4, 9, 3, 2, 6, 1, 7]
Best action possible is: {"type": "callElections", "city": "دمشق"}
127.0.0.1 - - [08/Jan/2020 22:56:35] "POST /api/test HTTP/1.1" 200 -
Receiving request to play round 1 of a game
Picking top city
Processing city
Predicting action
Processing results from prediction
Predicted actions: [8, 0, 5, 4, 3, 1, 6, 9, 2, 7]
Best action possible is: {"type": "launchCampaign", "city": "دمشق"}
127.0.0.1 - - [08/Jan/2020 22:56:35] "POST /api/test HTTP/1.1" 200 -
```

Abbildung 10: Auszug des Logs bei Verwendung des Netzes

Handbuch



In der Produktversion der Anwendung sind die Developer-Endpoints gesperrt, da das bereits von uns trainierte Modell verwendet werden sollte.

Deployment

Die Aufgabenstellung bietet zwei Möglichkeiten, wie die Lösung bereitgestellt werden kann. Einerseits gibt es die Möglichkeit ein Dockerfile zum Bauen eines Containers bereitzustellen. Andererseits kann man die Software auf einer Cloud Plattform der Wahl deployen. Wir haben beide Möglichkeiten realisiert.

Dockerfile

Mithilfe des Dockerfile im Root-Verzeichnis des Projekt lässt sich ein Image bauen, das als Container mit Docker ausgeführt werden kann. Die Anwendung ist dann unter <http://localhost:5000> zu erreichen.

```
docker build -t pandemie .  
docker run -p 5000:5000 -d pandemie
```

Cloud-Plattform

Zudem besteht die Möglichkeit die Anwendung über das Internet zu verwenden. Wir lassen dazu den Docker-Container auf einem Google Cloud Kubernetes Engine Cluster laufen.

Die Anwendung hier ist unter der URL <http://pandemie.nicolas-schaber.de> zu erreichen.

Alternativ lassen sich die Kubernetes Konfigurationsdateien auch verwenden, um die Anwendung auf einem lokalen Cluster selbst zu starten.

Testen

Mithilfe der aktuellsten Version des Kommandozeilen-Tools aus dem Repository des Wettbewerbs kann mit der Software ein Spiel gespielt werden. Dazu führt man das Tool mit Angabe der URL (Je nach Deployment-Methode) der Anwendung aus. Auf Unix-System muss das Tool eventuell zuerst ausführbar gemacht werden.

Hier z.B. für das lokale Deployment:

- **Windows:** `ic20_windows.exe -u „http://localhost:5000“`
- **Linux:** `./ic20_linux -u „http://localhost:5000“`
- **macOS:** `./ic20_darwin -u „http://localhost:5000“`

Anschließend spielt das Tool gegen unsere KI der Software ein Spiel. Nachdem entweder gewonnen oder verloren wurde wird das Tool beendet. Im Feld "outcome" lässt sich der Ausgang des Spiels durch "win" oder "loss" feststellen.

Fazit

Wir haben mit Deep Learning ein einfaches neuronales Netz aufgestellt und eine vollständige Anwendung in Form eines Web-Service darum gebaut. Das neuronale Netz funktioniert und liefert oft gute Ergebnisse, könnte jedoch noch weiter verbessert werden.

Dafür gibt es verschiedene Ansätze, wie zum Beispiel Deep Reinforcement Learning. Wir haben uns bewusst dagegen entschieden solche Ansätze von Anfang an zu implementieren, da wir hierfür nicht die notwendige Erfahrung mit Machine Learning aufbringen konnten. Schließlich war dies für alle von uns das erste richtige Projekt mit dieser Thematik.

Allerdings haben wir dadurch einen guten Einstieg in das Thema bekommen und können unsere Lösung durch Verbesserung bzw. Implementierung anderer Ansätze im Rahmen der Studienarbeit auch nach Abschluss des Wettbewerbs erweitern.