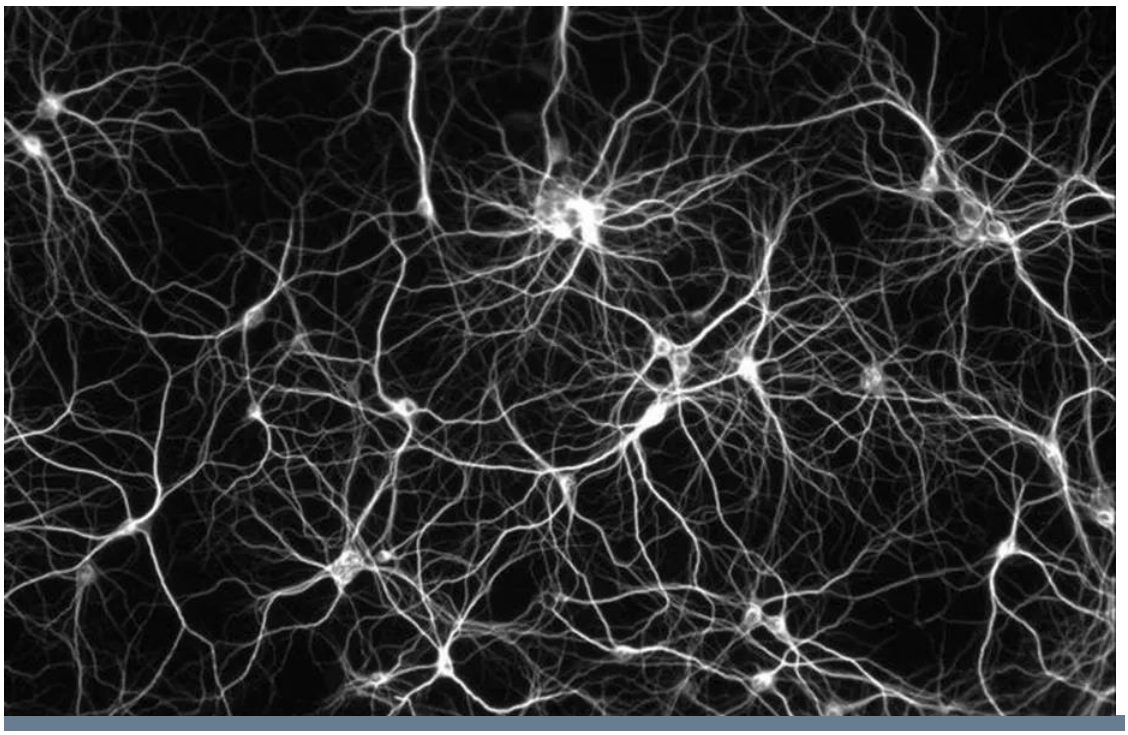


Technische Dokumentation: Optimierung von neuronalen Netzwerken

Projektteam 17

Projektbetreuer: Jürgen Vogel

Lukas Zoller

Remo Hofmann

Berner Fachhochschule

Departement für Technik und Informatik

Inhaltsverzeichnis

1 Zweck des Dokuments

Im Rahmen des Projekts 1 des Studiengangs Informatik an der Berner Fachhochschule wurde die Applikation hyperOptimize entwickelt. Dieses Dokument beschreibt diese Applikation zum einen in technischer Hinsicht. In diesem Zusammenhang werden die verwendete Technologie, die Systemarchitektur mit den einzelnen Komponenten, sowie das GUI beschrieben. Ebenfalls wird am Ende des Dokuments eine Bedienungsanleitung für die Applikation zur Verfügung gestellt. Im Glossar finden sich Erläuterungen zu den wichtigsten technischen Ausdrücken.

Da dieses Dokument auch projektmanagementspezifische Punkte enthalten soll, werden zum anderen auch einige Punkte angeführt, die eigentlich nicht in eine technische Dokumentation gehörten. Insbesondere Kapitel 2 Umsetzung, in welcher die Zusammenarbeit und organisatorische Belange während des Projekts beleuchtet werden, sowie Kapitel 3 Zielerreichung, in welchem die Anforderungen an das Projekt dem tatsächlichen Resultat gegenübergestellt werden, treten aus dem Rahmen einer technischen Dokumentation. Wenige weitere punktuelle Abweichungen von dieser Vorgabe sind vorhanden, werden hier jedoch nicht einzeln aufgeführt.

2 Umsetzung

2.1 Zusammenarbeit

Die Zusammenarbeit wurde mittels des Versionskontrollsystems GitHub gewährleistet. Unter dem Git Repository hyperOptimize (https://github.com/lukaszoller/hyperOptimize) wurde der jeweilige Stand des Projektes nach einer Änderung zwischengespeichert. Markierte To-Do's halfen, im Code auf noch offene Punkte einzugehen.

2.2 Organisatorisches

2.2.1 Organisation mit Projektbetreuer

Die Organisation mit dem Projektbetreuer, Herrn Vogel fand in wöchentlichen bis zweiwöchentlichen Statusmeetings statt. Dabei wurde jeweils eine Agenda von aufgetauchten Fragen unsererseits abgearbeitet, sowie verschiedene Lösungsansätze von kritischen Codeteilen besprochen. Daneben wurden Abgaben und dringendere Fragen per Mail ausgetauscht.

2.2.2 Organisation der Zusammenarbeit

Nach der Erarbeitung des Requirement-Dokuments wurde ein agiles Vorgehen im Projekt gewählt. Mindestens wöchentlich kam es zum gemeinsamen Abtausch, wer wieviel Zeit zur Verfügung hat, und was als nächstes erledigt werden soll. Erfolgreiche Änderungen wurden auf Git geladen und per Chat kommuniziert. Anfangs hat sich eine Person hauptsächlich auf die Logik konzentriert, die andere mehrheitlich auf den Aufbau des GUI. Bei der Zusammenführung von Codefragmenten wurde auch Pair Programming angewendet. So ist es gelungen, alle im Requirement Engineering definierten Ziele auch umzusetzen. Hierbei wurden zuerst die "Muss"-Ziele abgearbeitet, danach nach zuvor definierter Priorität.

3 Zielerreichung

In diesem Kapitel werden die Zielsetzungen, die in den Anforderungsspezifikationen definiert wurden, der tatsächlich implementierten Applikation gegenübergestellt.

3.1.1.1 Use Case 1 (Datensätze einlesen)

ID	Prio	Beschreibung	Bemerkung Zielerreichung
F1.1	Muss	Ein User hat die Möglichkeit, standardisierte Daten einzulesen. Diese Datei kann per Suche über die gängige Ordnerstruktur ausgewählt werden.	Erreicht: LoadDataView ermöglicht Einlesen per von csv-Dateien mittels Auswahl in einer Ordnerstruktur.
F1.2	Optional (P1)	Falls ein Datensatz nicht eingelesen werden kann, wird dies dem User gemeldet. (Mit einem Hinweis, weshalb es nicht funktioniert hat). Der Satz wird übersprungen, mit dem Einlesen wird fortgesetzt	Erreicht: Detaillierte Fehlerbehandlung implementiert
F1.3	Optional (P1)	Vorschaudaten des eingelesenen Datensatzes werden zur Kontrolle angezeigt.	Erreicht Aber: Nur minimale Vorschau (keine benutzerfreundlichen Features wie z.B. Scrollen vorhanden)
F1.4	Muss	Der User bestätigt, dass mit den eingelesenen Daten gearbeitet werden soll.	Erreicht: Pop-Up-Window für die Bestätigung erscheint

3.1.1.2 Use Case 2 (Modell automatisch parametrisieren und trainieren)

ID	Prio	Beschreibung	Bemerkung Zielerreichung
F2.1	Muss	Der User kann die Funktion wählen, ein optimales Modell zu seinen Trainingsdaten erstellen zu lassen. Hierbei gelangt er in ein neues Menü.	Erreicht: Optimize Model Button in der ModelView

F2.2	Muss	Der User kann bei den verschiedenen Hyperparametern entscheiden, wie gross die Range zum Testen sein soll.	Erreicht: Die Wahl von Ranges für sechs Hyperparameter ist möglich
F2.3	Muss	Der User kann abfragen, wie lange die Berechnung der optimalen Parameter mit den geladenen Training- und Testdaten sowie den Ranges für die Hyperparameter voraussichtlich dauert. Eine Schätzung der Laufzeit wird angezeigt.	Erreicht: Schätzung der Laufzeit in OptimizeModelView über Button möglich
F2.4	Muss	Der User wird vor Start der Berechnung darauf hingewiesen, falls die vorberechnete Dauer höher als ein definierter Grenzwert ist und kann dann entweder die Ranges anpassen oder muss erneut bestätigen, dass alles so durchgeführt werden soll.	Erreicht: Pop-Up-Fenster mit Laufzeitschätzung und Warnung, falls Laufzeit über 3h geschätzt wird.
F2.5	Muss	Das System liefert dem User ein mit optimalen Parametern versetztes und bereits trainiertes Modell, sowie die genauen Angaben zu den gefundenen optimalen Hyperparametern zurück.	Erreicht: Optimierung endet mit der Anzeige der Hyperparameter des besten Modells in der ModelView

3.1.1.3 Use Case 3 (Vergleich von Ergebnissen der Parametrisierung)

ID	Prio	Beschreibung	Bemerkung Zielerreichung
F3.1	Optional (P1)	Nach der Hyperparameteroptimierung kann der User Ergebnisse der Auswertung anzeigen lassen. Diese Anzeige visualisiert nach Möglichkeit die verschiedenen Hyperparameterkombinationen und die damit erzeugte Trefferrate.	Erreicht: Nach Optimierung wird eine Grafik mit Fehler pro Model pro Hyperparameter angezeigt
F3.2	Optional (P3)	Der User kann bei dieser Visualisierung nach Hyperparameter filtern.	Teilweise erreicht: Keine Filterung aber automatische

			Visualisierung pro Hyperparamer

3.1.1.4 Use Case 4 (Modell manuell parametrisieren und trainieren)

ID	Prio	Beschreibung	Bemerkung Zielerreichung
F4.1	Optional (P2)	Ein User soll auch in der Lage sein, ein Modell nach eigenen Kriterien zu trainieren. Anstatt dass die optimalen Hyperparameter ausgewertet werden, kann der User die Parameter selbst eingeben.	Erreicht: ModelView erlaubt manuelle Eingabe von Hyperparametern
F4.2	Optional (P2)	Nach Bestätigung wird auch hier das Modell mithilfe der eingelesenen Datensätze trainiert	Erreicht: Nach manueller Eingabe der Hyperparameter lässt sich das Modell mit diesen Hyperparametern trainieren

3.1.1.5 Use Case 5 (Auswertung nicht klassifizierter Daten durch ein bestehendes Modell)

ID	Prio	Beschreibung	Bemerkung Zielerreichung
F5.1	Optional (P2)	Zu einem bereits trainierten Modell können unklassifizierte Daten hinzugeladen werden.	Erreicht: In der ProjectView können unklassifizierte Daten geladen und mit einem passenden Modell klassifiziert werden.
F5.5	Optional (P2)	Die Daten werden anhand dieses Modells klassifiziert. Als Output wird wieder ein Standardisiertes File (csv) generiert.	Erreicht: Nach der Klassifizierung werden die Resultatdaten auf dem Filesystem gespeichert

3.2 Qualitätsanforderungen

ID	Prio	Beschreibung	Bemerkung Zielerreichung
Q1.1	Muss	Usability: Alle notwendigen Schritte zum Erstellen, Optimieren und testen eines Neuronalen Netzes sind in einem einheitlichen GUI implementiert.	Erreicht Aber: Einige Bugs vorhanden
Q1.2	Muss	Zuverlässigkeit: Die gelieferten Daten sollen deterministisch und auch reproduzierbar sein.	Erreicht: Test haben keine differierenden Resultate ergeben
Q1.3	Muss	Es muss eine Laufzeitabschätzung mit einer angestrebten Genauigkeit von +/- 25% der geschätzten Zeit als Warnungsgrundlage vor jeder Berechnung gemacht werden.	Erreicht: Die bisher ausgegebenen Schätzungen liegen in einem Fehlerbereich von ca. 10% Aber: Bei einer grossen Anzahl von zu trainierenden Models ist die Schätzung zu tief (Keine Trainingdaten für diese Fälle gesammelt). Schätzung wird mit der Zeit jedoch besser.

4 Technologie

4.1 Verwendete Technologie

- Programmiersprache: Python
- Library für die Erstellung und Manipulation von neuronalen Netzwerken: Keras
- GUI: TkInter (Interface für Tk library)
- Entwicklungsumgebung: PyCharm
- Datenbank: Sqlite

4.2 Eignung Technologie für die gewählte Problemstellung

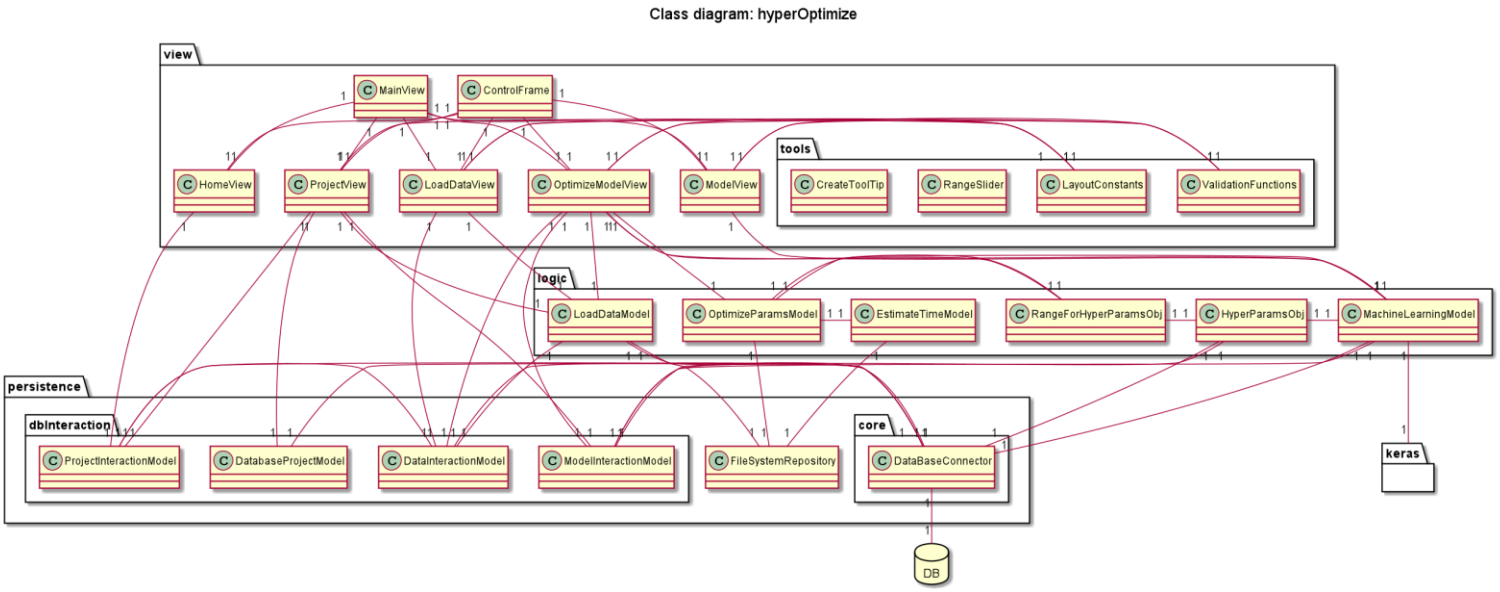
- Python: Die Applikation mit Python zu schreiben war eine gute Wahl. Python hat alle unsere Ansprüche (objektorientierte Programmierung, einfache und mächtige Libraries für Machine Learning Probleme) erfüllt.
- Keras: Keras als Library für den Umgang mit neuronalen Netzwerken war ebenfalls eine gute Wahl. Keras bietet umfangreiche Möglichkeiten für die Parametrisierung von neuronalen Netzwerken, sowie ein einigermassen einfaches Interface für die Erstellung, das Training, etc.
- Tkinter: Die GUI-Erstellung mit Tkinter war mühsam, da auf tiefem Level programmiert wird. Zudem entspricht die optische Erscheinung nicht mehr heutigen Standards.
- Pycharm bietet eine exzellente Entwicklungsumgebung, die insbesondere bezüglich der Interaktion mit Git, dem Debugging und dem Autovervollständigen höchsten Ansprüchen genügt.

- Sqlite: Sqlite hat als Einstiegsdatenbank genügend Funktionen geliefert, um allen Ansprüchen der Applikation (Projekte speichern, Speicherung von serialisierten Klasseninstanzen sowie Speicherung von JSON-Modellen der trainierten Neuronalen Netzen) gerecht zu werden.

5 Architektur

5.1 Systemübersicht

5.1.1 UML-Diagramm



5.1.2 Rechtfertigung von zwei oberen Layers und nicht MVP

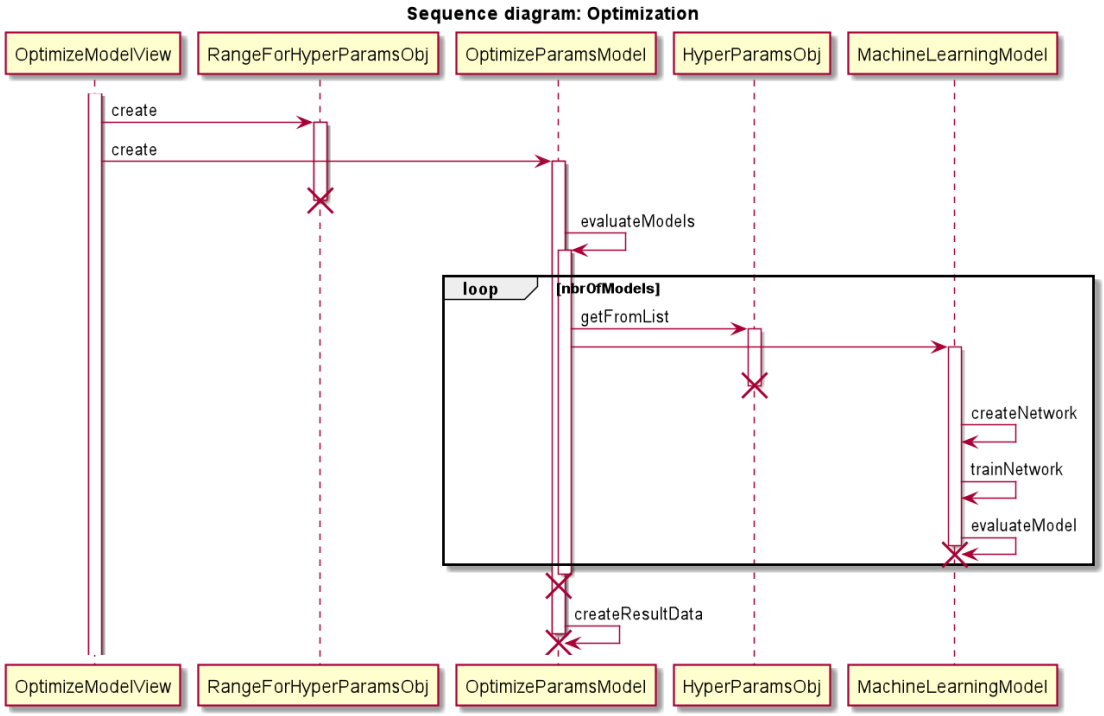
Zunächst wurde die Architektur gemäss dem MVP-Pattern entworfen mit vier Packages: View, Presenter, Model und Persistence. Während der Implementierung schien es, dass die Presenter-Ebene zu viel Programmieraufwand für zu wenig Nutzen bedeutete. So war beispielsweise nicht geplant, verschiedene Views zu einem Model zu implementieren. Die Presenter-Ebene wurde nach diesen Überlegungen gestrichen.

Gegen Ende der Implementierung wurde jedoch klar, dass die Views einige Aufgaben besitzen, die nichts mit der Darstellung von Informationen zu tun haben (z.B. bestimmte Datenprüfungen und Datenbankabfragen). Somit wurde deutlich, dass eine weitere Ebene zwischen View und Logic dennoch Sinn gemacht hätte. Dieses Refactoring wurde jedoch auf Grund des nahenden Projektendes nicht mehr vorgenommen.

5.1.3 Sequenz-Diagramm: Optimierung

Die folgende Grafik zeigt den Kern der Applikation – die Optimierung der Parameter für ein bestimmtes Modell: Die Applikation geht in folgenden Schritten vor:

- In der OptimizeModelView werden die Werte erfasst um die Ranges für die HyperParameter abzustecken. Anschliessend wird das RangeForHyperParamsObj gebildet.
- Das OptimizeParamsModel wird gebildet.
- Basierende auf den Ranges des RangeForHyperParamsObj wird eine Liste von HyperParamsObj gebildet (Für jedes HyperParamsObj in dieser Liste wird später ein neuronales Netzwerk gebildet wobei das HyperParamsObj die HyperParameter dieses Netzwerks bestimmt).
- Im OptimizeParamsModel wird die evaluateModels-Methode aufgerufen. Diese geht wie folgt vor:
    - Loop über alle HyperParamsObj in der oben genannten Liste
    - Netzwerk bilden, trainieren und testen
- Schlussendlich werden die Ergebnisse der Tests für die Visualisierung sowie das beste Modell gespeichert.



5.2 Komponenten

In diesem Unterkapitel werden die einzelnen Komponenten, meistens sind diese als eigene Klassen implementiert, erklärt.

5.2.1 GUI

5.2.1.1 MainView

Die MainView ist wie im Klassendiagramm ersichtlich die Hauptansicht. Sie erstellt beim Start des Programms alle weiteren Ansichten und auch die Kontrollklasse ControlFrame. Alle Ansichten sind eine Implementation des tkinter-Frames. Je nach Aktion wird dann das jeweilige Frame, also die jeweilige Ansicht in den Vordergrund geladen. So ist die ganze Applikation in einem Fenster angeordnet und kann auch als Ganzes geschlossen werden. Teil der MainView ist auch der Menüblock oberhalb der Frames, sowie der Buttonblock mit variablen Schnellelementen unterhalb der Frames.

5.2.1.2 HomeView

Die HomeView ist das erste Frame, das nach dem Start geladen wird. Hier kann ein Projekt aus einer Liste ausgewählt oder gelöscht, sowie ein neues Projekt erstellt werden.

5.2.1.3 ProjectView

Hat man ein Projekt ausgewählt oder neu erstellt kommt man in das Projektframe. Hier können spezifische Änderungen am Projekt vorgenommen werden. Zum einen können die Daten geladen werden (man gelangt in die LoadData-Ansicht), aber auch verschiedene Modelle im Zusammenhang mit diesem Projekt ausgewählt oder gelöscht, sowie neue Modelle erstellt werden.
Bereits Trainierte Modelle können neue Daten klassifizieren.

5.2.1.4 LoadDataView

Die LoadDataView kann auf dem Filesystem eine csv-Datei auswählen. Diese wird per Vorschau angezeigt. Die csv-Datei dient entweder als Set für Trainings – und Testdaten für alle im Projekt beinhalteten Modelle. Oder als Klassifizierungsset für bereits trainierte Modelle.
Das Laden der Daten generiert optional ein Fehlerhandling und zeigt dem Benutzer visuell, wenn ein Datensatz oder eine Eingabe nicht passt.

5.2.1.5 ModelView

In der ModelView können zu einem ausgewählten Model manuelle Hyperparameter eingetragen werden. Mit diesen kann das Modell dann mithilfe der Trainings – und Testdaten des Projekts trainiert werden. Ausserdem befindet sich auch der Button zur automatischen Optimierung des Modells in diesem Frame. Hier gelangt man zur OptimizeModelView.

5.2.1.6 OptimizeModelView

Die OptimizeModelView hat für alle möglichen Hyperparameter, welche durch die Applikation getestet werden, eine Range oder Auswahlboxen, damit der Benutzer eingeben kann, welcher Umfang an Hyperparametern getestet werden soll, und wie viele verschiedene Modelle getestet werden soll (nach Zufallsprinzip aus den eingegebenen Ranges). Man kann dann mit den Eingaben das optimale Modell herausfinden lassen. Hierbei wird verifiziert, ob die Eingaben des Users korrekt sind, und eine Zeitabschätzung gemacht, welche in einem Warn-Popup resultiert, falls die Zeit zur Berechnung auf mehr als drei Stunden geschätzt wird. Sobald ein Berechnungslauf durchgelaufen wird, kann der Benutzer auswählen, ob er eine Visualisierung der Daten in einem eigenen Popup sehen will und kommt dann zurück auf die ModelView.

5.2.2 Logic

5.2.2.1 EstimateTimeModel

Diese Klasse ist für die Laufzeitschätzung der Optimierung der Hyperparameter zuständig. Die Schätzung wird durch lineare Regression mit polynomialen Features vorgenommen (Python scikit-Library). Das Modell, welches dafür verwendet wird, wird bei jeder Schätzung neu trainiert. Die Trainingdaten werden nach jeder durchgeführten Optimierung mit den Werten dieser Optimierung ergänzt. Somit steigt die Zuverlässigkeit der Schätzung mit fortschreitender Benutzung der Applikation.

Die Schätzung für die Laufzeit der Optimierung wird berechnet, indem für jedes HyperParamsObj in der HyperParamsObj-Liste, welche dem OptimizeParamsModel-Objekt übergeben wird, die Laufzeit berechnet wird und schlussendlich die Summe über alle HyperParamsObj-Objekte gebildet wird. Die für die Berechnung verwendeten Features sind:

- Anzahl Layers
- Anzahl Knoten pro Layer
- Learning Rate
- CPU-Geschwindigkeit

5.2.2.2 HyperParamsObj

Diese Klasse wird verwendet, um die Hyperparameter zu speichern, die für ein einzelnes neuronales Netzwerk notwendig sind. Folgende Hyperparameter werden gespeichert:

- Anzahl Knoten pro Array (Alle Hidden Layers besitzen dieselbe Anzahl Knoten)
- Anzahl hidden Layers
- Aktivierungsfunktion
- Drop Out Rate
- Loss Function
- Model Optimizer
- Learning Rate
- Learning Rate Decay

5.2.2.3 LoadDataModel

Diese Klasse dient als Zwischenglied zwischen FileSystemRepository, mit welchem die Training- und Testdaten eingelesen werden und der LoadDataView, die dem Benutzer das Laden von Daten erlaubt.

5.2.2.4 MachineLearningModel

Diese Klasse stellt die Methoden für das Bilden, Trainieren und Testen der neuronalen Netzwerke bereit. Die createNetwork-Methode der MachineLearningModel-Klasse verwendet die Sequential-Klasse von Keras. Keras.Sequential besitzt sehr umfangreiche Funktionalität und ist dementsprechend komplex. Die createNetwork-Methode schränkt diese Komplexität bewusst ein. Folgende Einschränkungen, bzw. Funktionalität wurden umgesetzt:

- Der createNetwork-Methode wird ein HyperParamsObj mitgegeben, nach dessen Vorgaben das neuronale Netzwerk gebaut werden soll.
- Alle hidden Layers besitzen dieselbe Anzahl Nodes.
- Nach jedem hidden Layer wird ein Drop-out-Layer eingebaut.
- Abgesehen von den im HyperParamsObj mitgegebenen Hyperparametern werdend die Defaulteinstellungen von Keras.Sequential verwendet.

Die weiteren Methoden dieser Klasse (z.B. trainNetwork, etc.) sind reine Wrapper für Keras-Methoden.

5.2.2.5 OptimizeParamsModel

Diese Klasse ist für die Auswahl des besten Modells bezüglich einem Datensatz und einem bestimmten Hyperparameterbereich zuständig. In der evaluateModels-Methode wird die eigentliche Optimierung durchgeführt. Sie erhält eine Liste mit HyperParamsObj-Objekten, bildet für jedes dieser Objekte ein neuronales Netzwerk und testet dieses auf einem Testdatensatz. Das Modell, welches die besten Resultate bei der Klassifizierung eines Testdatensatzes erzielt, wird für die spätere Verwendung gespeichert.

5.2.2.6 RangeForHyperParamsObj

Diese Klasse wird einerseits dazu verwendet, um den Bereich abzustecken, in welchem das optimale neuronale Netzwerk gesucht werden soll. Dafür werden für jeden Hyperparameter, der im HyperParamsObj gespeichert werden kann, ein Bereich definiert.

Andererseits wird basierend auf diesem Bereich und der Angabe der Anzahl zu prüfender Modelle mit der createHyperParamsListRandom eine Liste von HyperParamsObj generiert. Für jedes HyperParamsObj-Objekt in dieser Liste wird für jeden Hyperparameter ein zufälliger Wert aus dem entsprechenden Bereich des RangeForHyperParamsObj-Objektes gewählt.

5.2.3 Persistence

5.2.3.1 Core Package: DatabaseConnector

5.2.3.2 dbInteraction Package

- DatabaseProjectModel
- DataInteractionModel
- ModelInteractionModel
- ProjectInteractionModel

5.2.3.3 FileSystemRepository

Diese Klasse ist für das Speichern und Laden von Dateien von und auf das Filesystem zuständig. Folgende Methoden sind erwähnenswert:

- loadDataForTrainingOrPrediction: Lädt Training- oder Testdaten in die Applikation
- getEstTimeData: Lädt die Daten für die Laufzeitschätzung in die Applikation
- saveTimeMeasurement: Speichert die Daten für die Laufzeitschätzung auf dem Filesystem
- hyperParamsToData: Diese Methode konvertiert eine Liste mit HyperParamsObj-Objekten in einen 2D-Array, der für die Laufzeitschätzung der Optimierung verwendet wird.

6 Lessons Learned

6.1 Architektur

Bei der Architektur waren wir sehr zufrieden mit der Wahl von Python als Programmiersprache und mit der Wahl von Keras als Machine Learning-Paket für Python. Eine Visuelle Implementierung mittels TkInter würden wir in einem zukünftigen Projekt nicht mehr machen. (Allenfalls wäre dann die Applikation eine reine Command Line (mehr Funktionalität in der begrenzten Entwicklungszeit) Applikation oder ein Webinterface (Zeitgemässer Auftritt, die Applikation könnte auf einem leistungsstarken Server laufen)).
Zusätzlich kamen beim Code Review Punkte zum Vorschein, welche in der Projekt – und Codestruktur noch verbesserungswürdig wären. Die gewählte Trennung von View und Model (ohne Presenter) hat zur Folge, dass teilweise Views zu viel Logik beinhalten. Ein ausgeklügeltes Interface mit allen Möglichkeiten, die eine View braucht, wäre hier vonnöten (entweder wird dort also die Datenbank angesteuert oder die verschiedenen Modell-Klassen).
Der zweite Punkt betrifft die Datenbankimplementation. Die Datenbank sollte möglichst auswechselbar sein. Jedoch wurden bei unserer Datenbankklasse zum Teil bereits Klassenobjekte erstellt oder deserialisiert, was zu vermeiden wäre.

6.2 VCS

Von beiden Programmierern wurden Code-Changes jeweils in den master-Branch des Versionskontrollsystems gepusht. Dies führte teilweise zu delikaten Merge-Konflikten. Um aber immer eine lauffähige Applikation zu haben, wäre es in zukünftigen Projekten sinnvoll, pro Feature, oder pro Use-Case einen eigenen Branch zu erstellen und diesen dann, wenn der Use Case oder das Feature funktioniert in den Master-Branch des Git zu mergen.

6.3 Use Cases Ausblick

Obwohl die Anforderungsspezifikationen die Use Cases sinnvoll beschreiben, sind während dem Gebrauch der Applikation einige Ideen aufgetaucht, wie die Applikation noch verbessert werden könnte. Folgende Use Cases wären ebenfalls interessant umzusetzen:

- Speicherung der Optimierungsgrafik im Modell.
- Export eines trainierten Modells mittels JSON oder XML, damit es von anderen Applikationen verwendet werden kann.

- Verlagerung der Berechnung auf einen leistungsfähigen Server und Verlagerung des GUI auf eine Website.

7 Installationsanleitung

7.1 Mittels VCS und IDE (Windows / Linux)

1. Voraussetzung: Python 3.5 bis 3.7. Benötigte Module werden je nach Entwicklungsumgebung nach laden des Projekts automatisch bereitgestellt.
2. Pullen des Git-Repository: https://github.com/lukaszoller/hyperOptimize.git direkt in die Entwicklungsumgebung.
3. Benötigte Packages installieren (mit pip, siehe unten) oder je nach Entwicklungsumgebung automatisch installieren lassen.
4. Starten der Main-Datei (unter /src/hyperOptimizeApp/Main.py).

7.2 Windows (nur möglich unter 64-bit, mit installer executable)

Anmerkung: Eine rein lauffähige Executable mit Python-Code für alle Windows-Versionen hat sich als schwierig herausgestellt. Hier ist die Installationsanleitung mittels eines Windows-Installers. Dabei müssen aber trotzdem Python und die benötigten Module im Vorfeld installiert werdne.

1. Python 3.7 Installieren (Die Machine Learning Komponenten benötigen Version 3.5 bis 3.7).
   Wichtig: Unbedingt die 64-Bit-Version installieren.
   Wichtig: Unbedingt Python zu den Umgebungsvariablen hinzufügen.
   https://www.python.org/ftp/python/3.7.6/python-3.7.6-amd64.exe
2. Benötigte Module: (lässt sich mit Powershell mit «pip install [modulname]» installieren (Diese Module sind der jeweiligen Python-Version angepasst, deshalb müssen sie lokal installiert werden).

- pandas
- sklearn
- urllib
- keras
- future
- jsonpickle
- cpuinfo
- numpy
- matplotlib
- pytz
- tensorflow

3. Die Datei **hyperOptimize-1.0.0.win-amd64.exe** als Administrator ausführen.
4. Unter $pythonhome(=Pfad zur Python-Installation)\Lib\site-packages\hyperOptimizeApp kann nun die Datei HyperOptimize geöffnet werden.

8 Bedienungsanleitung GUI

8.1 Hauptkomponenten der Applikation HyperOptimize

Bevor die möglichen Aktionen von HyperOptimize erklärt werden, ist es sinnvoll, die von der Applikation verwendeten Komponenten (Projekt und Modul) kurz vorzustellen:

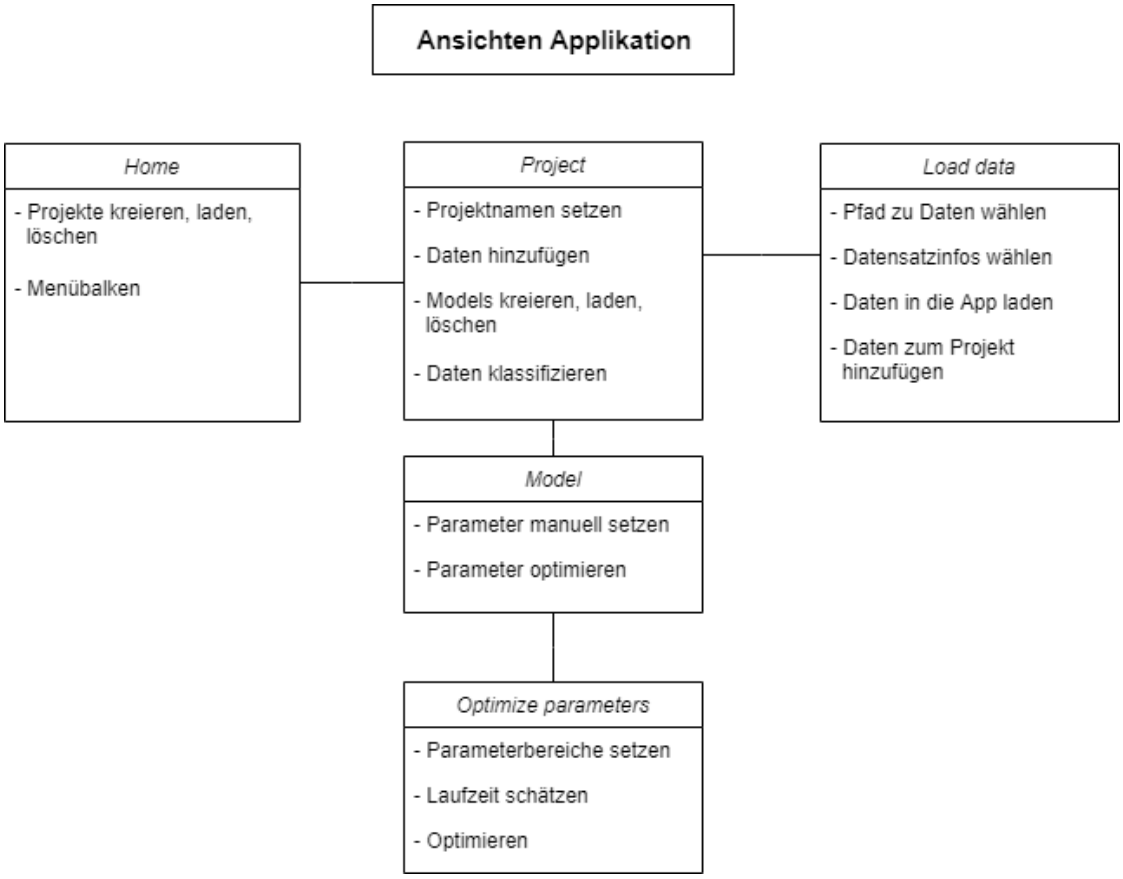
- Projekt: In einem Projekt werden verschiedene Modelle zu einem bestimmten Datensatz vereint.
- Model: Ein Modell ist gehört zu einem bestimmten Projekt und bezieht sich auf den, zu diesem Projekt zugehörigen Datensatz. Es kann mithilfe eines neuronalen Netzwerks trainiert werden und erlaubt die Klassifikation von Daten mit derselben Form wie der Projektdatensatz.

8.2 Ansichten

8.2.1 Ansichten gemäss Anforderungsspezifikationen

8.2.2 Übersicht Ansichten nach Umsetzung

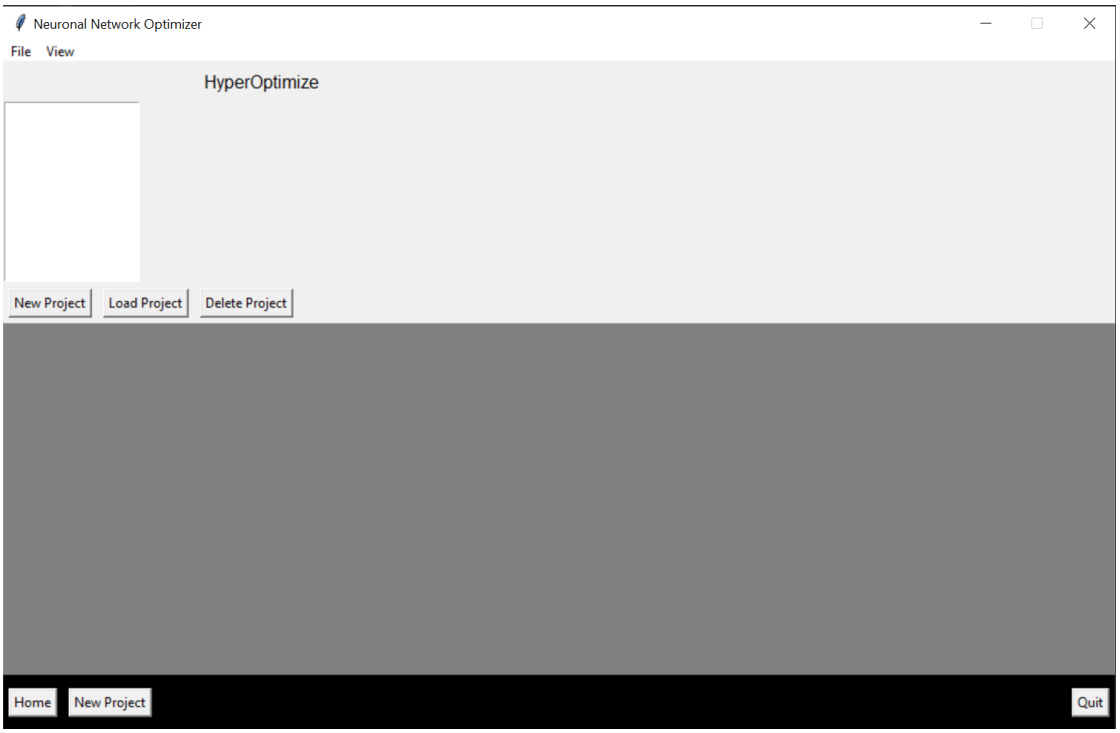
Die folgende Grafik zeigt, wie die einzelnen Views zusammenhängen. Bevor die möglichen Handlungen der Applikation genauer beschrieben werden, wird grob auf die einzelnen Ansichten und ihre Beziehung eingegangen.



Die Applikation startet mit der Home-Ansicht in welcher Projekte verwaltet werden. Von dort gelangt man in die Projekt-Ansicht, die die Informationen zu einem einzelnen Projekt enthält (z.B. trainierte Modelle, Trainingdaten, etc.). An diese Projekt-Ansicht sind einerseits die Load-data-Ansicht, sowie die Modell-Ansicht angehängt. Mit der Load-data-Ansicht werden Training- oder Daten für die Klassifizierung hochgeladen. Mit der Modell-Ansicht können Modelle trainiert werden. Das Training kann entweder mit manuell eingegebenen Parametern oder über eine Optimierung vorgenommen werden. Letzteres wird über die Optimize-parameters-Ansicht erreicht.

8.2.3 Detaillierte Beschreibung der Ansichten

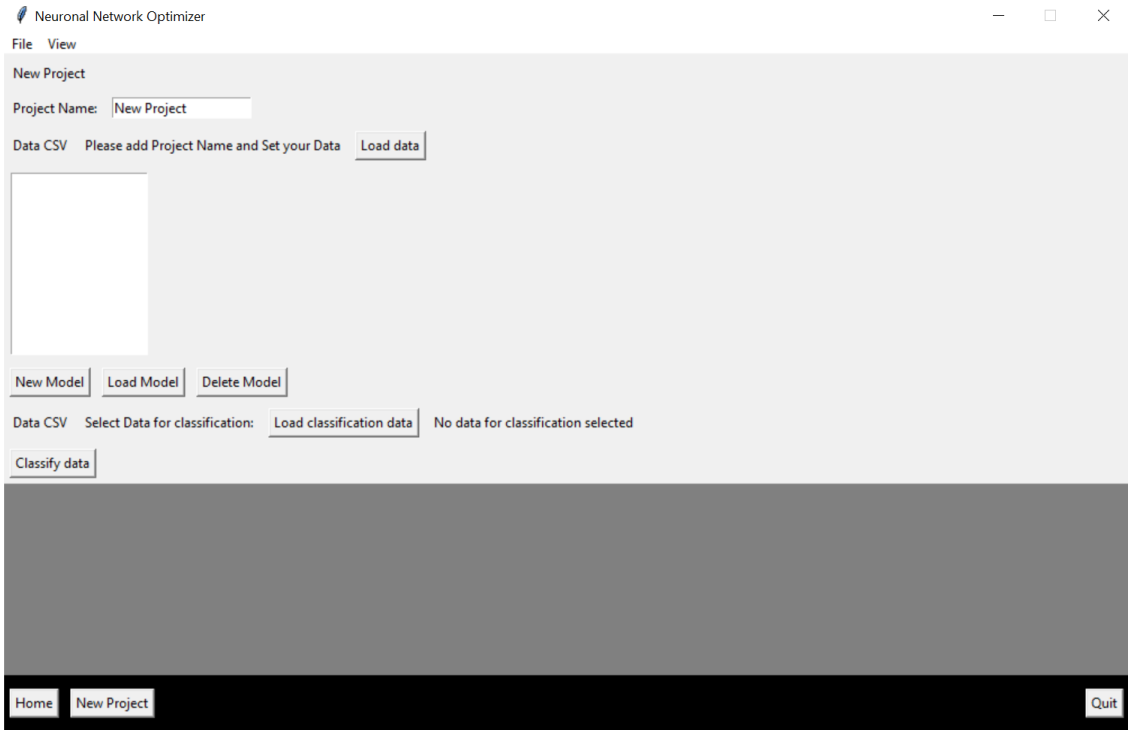
8.2.3.1 Home-Ansicht



- New Project: Erstellt ein neues Projekt und wechselt zur Projekt-Ansicht

- Load Project: In der obigen Grafik sind noch keine Projekte vorhanden. Andernfalls würden im Feld oberhalb der Buttons Einträge erscheinen. Mit Klick auf einen solchen Eintrag und den Load-Project-Button wird das Projekt geladen und zur Projekt-Ansicht gewechselt.
- Delete Project: Mit Klick auf einen Eintrag in der Liste mit Projekteinträgen sowie auf den Delete-Button wird das ausgewählte Projekt gelöscht.

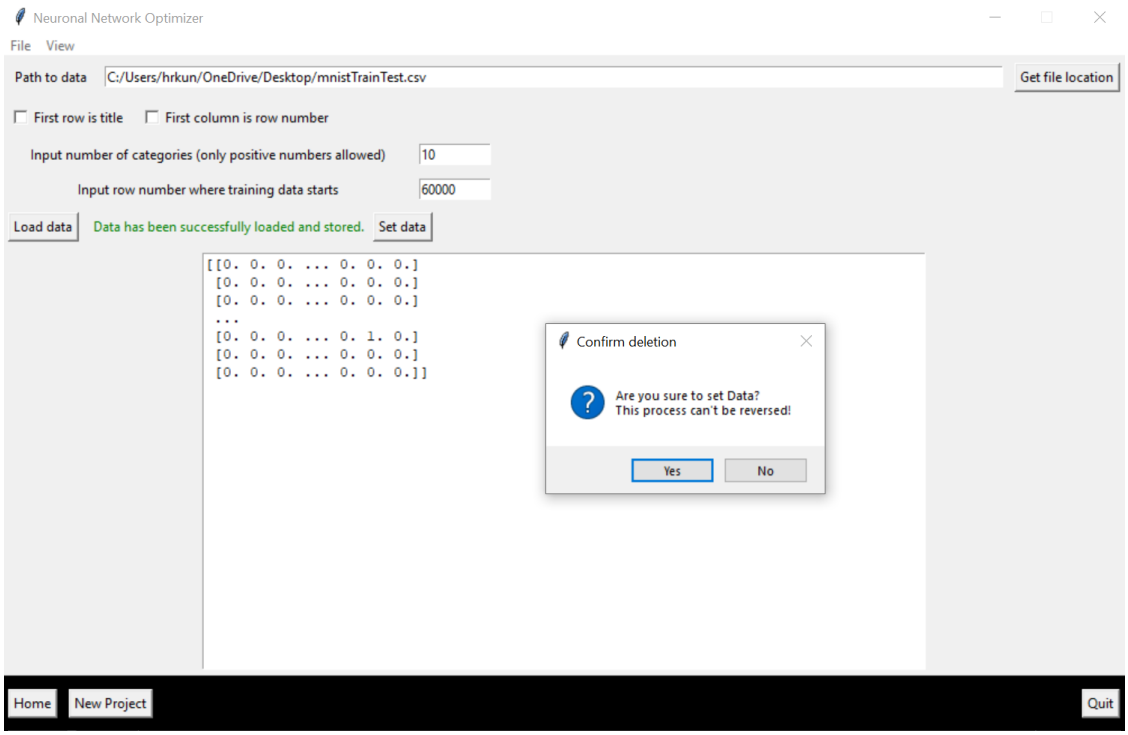
8.2.3.2 Projekt-Ansicht



- Project Name: In diesem Feld kann der Projektname eingegeben werden.
- Load data: Mit diesem Button wird zur Load-data-Ansicht gewechselt, in welcher Trainingdaten zum Projekt hinzugefügt werden können.
- New Model: Mit Klick auf diesen Button wird ein neues Model erstellt. Es erscheint ein Pop-Up-Fenster, in welchem der Modellname eingegeben werden kann. Dieser Modellname erscheint im Feld, welches in der oberen Grafik noch leer ist (weil noch keine Modelle erstellt wurden)
- Load Model: Mit Klick auf ein erstelltes Modell in der Modellliste, sowie auf den Load-Model-Button wird zur Model-Ansicht gewechselt, in welcher das Modell trainiert werden kann.
- Delete Model: Mit Klick auf ein erstelltes Modell in der Modellliste, sowie auf den Delete-Model-Button kann das ausgewählte Modell gelöscht werden.
- Load classification data: Mit Klick auf diesen Button wird zur Load-Data-Ansicht (für Klassifizierungsdaten) gewechselt, in welcher Daten für die Klassifizierung geladen werden können.
- Classify data: Wenn Klassifizierungsdaten geladen wurden, können diese mit Klick auf ein trainiertes Modell aus der Modellliste sowie mit Klick auf den Classify-data-Button klassifiziert werden. Die klassifizierten Daten werden automatisch an derselben Stelle wie die zu klassifizierenden Daten gespeichert.

8.2.3.3 Load-Data-Ansicht

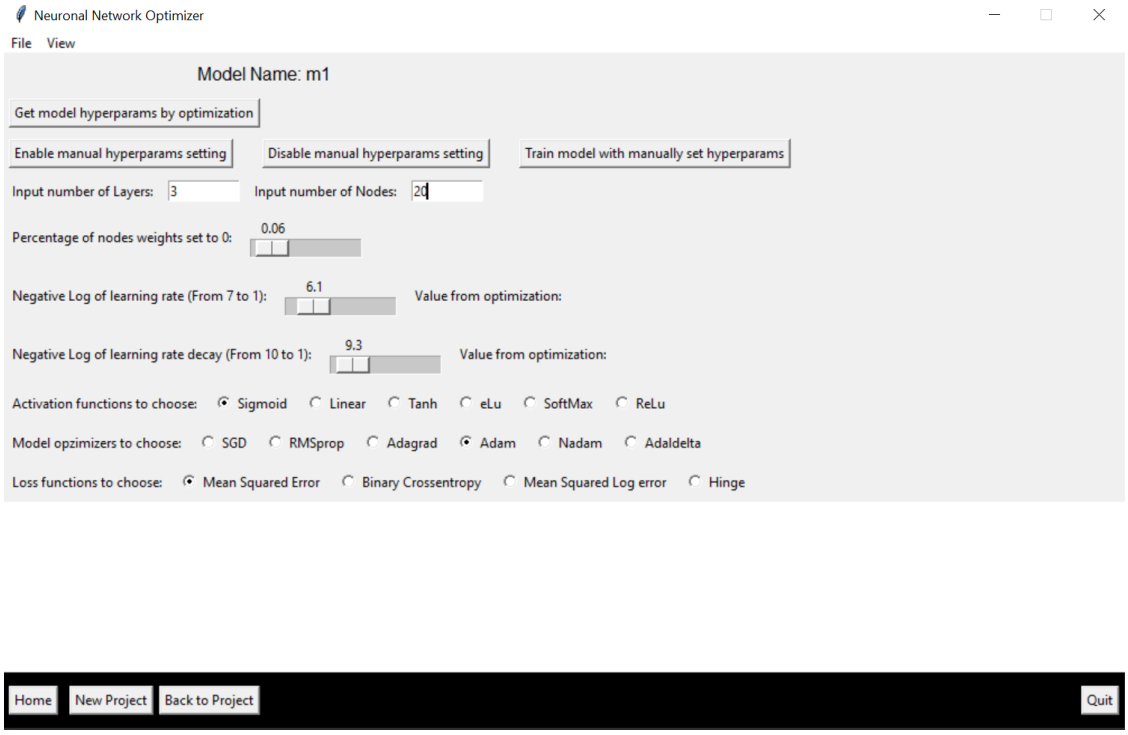
Die Load-Data-Ansicht wird ein zwei Ausführungen von der Applikation verwendet: Erstens für Trainingdaten, zweitens für Klassifizierungsdaten. In der Load-Data-Ansicht für Klassifizierungsdaten werden die zwei Felder "Input number of categories" und "Input row number where training data starts" nicht angezeigt.



- Path to data: In diesem Feld muss der Pfad zu den Daten angegeben werden.
- Get file location: Der Pfad kann entweder manuell oder über den Get-file-location-Button eingetragen werden.
- First row is title: Diese Checkbox muss gesetzt werden, wenn die erste Zeile im Datensatz Spaltennamen enthält (diese Zeile wird in den hochgeladenen Daten entfernt).
- First column is row number: Diese Checkbox muss gesetzt werden, wenn die erste Spalte im Datensatz Zeilennummern enthält (diese Spalte wird in den hochgeladenen Daten entfernt).
- Input number of categories: In diesem Feld muss die Anzahl der Kategorien, die der Trainingdatensatz enthält, angegeben werden.
- Input row number where Training data starts: In diesem Feld muss die Zeilennummer eingegeben werden, ab welchem die Testdaten beginnen.[1]
- Load data: Mit Klick auf diesen Button werden die Daten, sofern sie geeignet sind, in die Applikation geladen. Im grossen Feld wird eine Vorschau der geladenen Daten angezeigt.
- Set data: Mit Klick auf diesen Button erscheint ein Pop-Up-Window, welches fragt, ob die Daten tatsächlich zum Projekt hinzugefügt werden sollen. Bei Bestätigung werden die Daten hinzugefügt. Mit diesem Schritt sind die Daten für dieses Projekt gesetzt und können nicht mehr geändert werden.

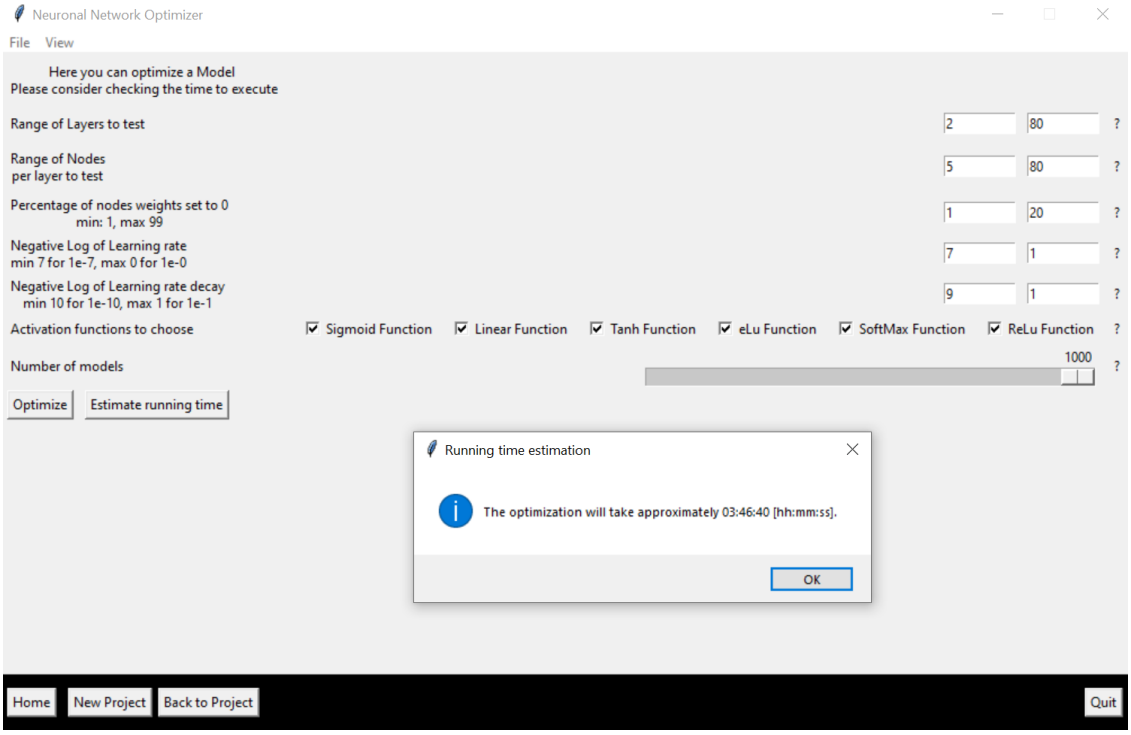
[1] Der Text in der Load-Data-Ansicht zu diesem Punkt ist fehlerhaft. Die Eingabe bestimmt nicht, ab wo die Trainingdaten beginnen (diese beginnen bei Zeile 1), sondern ab wo die Testdaten beginnen.

8.2.3.4 Model-Ansicht



- Get model hyperparams by optimization: Mit Klick auf diesen Button wird zur Optimize-parameters-Ansicht gewechselt.
- Enable manual hyperparams setting: Mit Klick auf diesen Button wird die manuelle Eingabe von Hyperparametern via Inputfeldern, Sliders und Radiobuttons ermöglicht.
- Disable manual hyperparams setting: Mit Klick auf diesen Button werden die Inputfelder für die manuelle Hyperparametereingabe ausgegraut und die Eingabe verunmöglicht.
- Train model with manually set hyperparams: Mit Klick auf diesen Button werden die manuell angegebenen Hyperparameter verwendet um ein Modell zu trainieren.
- Inputfelder für Hyperparameter:
    - Number of Layers: Die Anzahl hidden Layers die das neuronale Netzwerk haben soll (Drop-out Layers nicht eingerechnet).
    - Node weights set to 0: Für jeden hidden Layer wird dem neuronalen Netzwerk zusätzlich ein Drop-out-Layer hinzugefügt. Der Slider spezifiziert, bei wie vielen Nodes die Gewichte auf Null gesetzt werden sollen.
    - Negative Log of learning rate: Die Learning rate bezeichnet die Grösse der Schritte bei der Optimierung des neuronalen Netzwerks. Minimus ist 7 (d.h. eine Learning Rate von 1e7) Maximum ist 1 (d.h. eine Learning rate von 1).
    - Negative Log of learning rate decay: Der Learning rate decay bezeichnet die Grösse, mit welcher die Learning rate bei jeder Optimierungsiteration kleiner werden soll. Minimum ist 10 (d.h. ein Learning Rate Decay von 1e-10) Maximum ist 1 (d.h. eine Learning Rate Decay von 1).
    - Activation Function: Dies bezeichnet die Aktivierungsfunktion der Knoten des neuronalen Netzwerks.
    - Model optimizers: Hier kann ausgewählt werden, mit welchem Optimierer das Netzwerk trainiert werden soll.
    - Loss function: Hier kann die Loss function für das Training des neuronalen Netzwerks angegeben werden.

8.2.3.5 Optimize-Parameters-Ansicht



- Inputfelder: Mit den vorhandenen Inputfeldern wird der Bereich definiert, aus welchem das optimale Model gesucht werden soll (Für die Erklärung der einzelnen Inputfelder siehe oben bei Model-Ansicht.
- Number of Models: Mit diesem Slider lässt sich festlegen, wie viele neuronale Netzwerke trainiert werden sollen, um dann aus diesen das Model mit der geringsten Fehlerquote auf den Testdaten auszuwählen.
- Estimate running time: Mit Klick auf diesen Button wird eine Schätzung vorgenommen, wie lange das Training und Testing der Models dauern wird.[2]
- Optimize: Mit Klick auf diesen Button definiert die Applikation zunächst für jeden der sechs Hyperparameter einen Bereich. Anschliessend wird die mit dem "Number of models" definierte Anzahl von neuronalen Netzwerken aus diesem 6-dimensionalen Raum gebildet, trainiert und getestet. Die Applikation schätzt zuerst die Laufzeit für diese Schritte. Falls die Schätzung einen Wert von drei Stunden überschreitet, wird per Pop-Up-Fenster eine Warnung ausgegeben. In diesem Pop-Up-Fenster kann man wählen ob die Optimierung fortgesetzt werden soll.

8.3 Use Cases

8.3.1 Neues Projekt erstellen

- Home-Ansicht → Klick auf "New Project"
- Dem Projekt einen Namen geben

[2] Die Genauigkeit der Schätzung nimmt zu, wenn mehr Optimierungen durchgeführt wurden. Dies liegt daran, dass die anfänglichen Trainingdaten für diese Schätzung nicht mit dem System erstellt wurden, auf welchen die HyperOptmize beim Benutzer installiert ist.

8.3.2 Trainingdaten für ein Projekt laden

- Datenformat: Die Trainingdaten müssen eine bestimmte Form haben, um von HyperOptimize verarbeitet werden zu können und um sinnvolle Ergebnisse zu erzeugen. Folgende Kriterien sind einzuhalten:
    - Dateiformat: .csv
    - Daten müssen in einer zweidimensionalen Tabelle sein.
    - Die Zeilen der Tabelle müssen den Fällen entsprechen.
    - Die Spalten der Tabelle müssen von links beginnend den Features entsprechen und anschliessend den Klassifikationskategorien.
    - Für jede Kategorie muss eine eigene Spalte mit den Werten 0 oder 1 (0 = gehört nicht zu dieser Kategorie, 1 = gehört zu dieser Kategorie) existieren. Pro Zeile sollte in den Kategoriespalten nur eine einzige 1 enthalten sein.
    - Die Training- und Testdaten müssen in demselben Datensatz mitgegeben werden. Zuerst kommen die Training-, dann die Testdaten. In der Load-Data-Ansicht kann spezifiziert werden, ab welcher Zeile die Testdaten beginnen.[3]
- Projekt auswählen oder neues Projekt erstellen
- Projekt-Ansicht → Klick auf "Load data"
- Load-Data-Ansicht → Pfad zu den Daten angeben
- Weitere Inputfelder ausfüllen
- "Load data" klicken → Fehlermeldungen informieren über fehlerhafte Daten.
- "Set data" klicken → Nach der Bestätigung mit einem Pop-Up-Fenster werden die Daten unwiderruflich dem Projekt hinzugefügt.

8.3.3 Klassifizierungsdaten laden

- Wie unter Trainingdaten für Projekt laden vorgehen (Inputfelder für die Anzahl Kategorien sowie die Zeilennummer ab welcher die Testdaten beginnen, sind im Gegensatz zur Load-data-Ansicht für Trainingdaten nicht vorhanden).
- Kriterien für Klassifizierungsdaten:
    - Dateiformat: .csv
    - Daten müssen in einer zweidimensionalen Tabelle sein.
    - Die Zeilen der Tabelle müssen den Fällen entsprechen
    - Die Spalten müssen den Features entsprechen.
    - Die Anzahl Spalten muss gleich der Anzahl Features des Trainingdatensatzes sein.

8.3.4 Neues Modell erstellen

- Model-Ansicht → Klick auf "New Model" → Name des Model eingeben

8.3.5 Modell trainieren (manuelle Eingabe der Hyperparameter)

- Model-Ansicht → Klick auf ein Model aus der Modellliste
- Klick auf "Load model" → Wechsel zur Model-Ansicht
- Model-Ansicht → Klick auf " Enable manual hyperparams setting"
- Hyperparameter manuell eingeben
- Klick auf " Train model with manually set hyperparams"

8.3.6 Modell trainieren (Optimierung)

- Wie beim Trainieren mit manuell eingegebenen Parametern vorgehen um zur Model-Ansicht zu kommen.

[3] Der Text in der Load-Data-Ansicht zu diesem Punkt ist fehlerhaft. Die Eingabe bestimmt nicht, ab wo die Trainingdaten beginnen (diese beginnen bei Zeile 1) sondern ab wo die Testdaten beginnen.

- Klick auf " Get model hyperparams by optimization"
- Optimize-parameters-Ansicht → Ranges für den Hyperparameterbereich angeben.
- Klick auf "Optimize" → Optimierung wird durchgeführt
- Nach der Optimierung wird eine Grafik angezeigt, die die Verteilung der getesteten Modelle pro Hyperparameter und deren Fehlerrate anzeigt. Obwohl nach dem Schliessen der Grafik zurück zur Modellansicht gewechselt wird und dort die Hyperparameter des neuronalen Netzwerks mit der kleinsten Fehlerrate angezeigt werden, kann diese Grafik bei folgenden Schritten helfen: Erstens kann eine weitere Optimierung vorgenommen werden und der Hyperparameterraum mithilfe der Grafik eingeschränkt werden (z.B. macht die Grafik vielleicht deutlich, dass bestimmte Optimierer sehr schlechte Modelle liefern und nicht verwendet werden sollten). Zweitens kann die Grafik helfen, beim Training eines Modells mit manueller Eingabe der Parameter sinnvolle Parameter zu wählen.
- Nach Schliessen der Grafik wird zurück zur Model-Ansicht gewechselt. Das Modell ist nun trainiert und kann für die Klassifikation verwendet werden.

8.3.7 Daten klassifizieren

- Um Daten zu klassifizieren müssen zwei Bedingungen erfüllt sein: Erstens muss ein trainiertes Modell in der Modellliste existieren. Zweitens müssen Klassifizierungsdaten geladen worden sein.
- Klick auf ein Model aus der Modellliste und anschliessend klick auf "Classify" → Die Klassifizierungsdaten werden mit dem ausgewählten Modell klassifiziert. Der neue Datensatz mit den Klassifizierungen wird automatisch am selben Ort wie die Klassifizierungsdaten als csv-Datei gespeichert.

9 Glossar

Drop out rate	Die Drop out rate bestimmt, für wie viele Knoten eines Layers die Gewichte auf Null gesetzt werden sollen. Die Anwendung einer Drop out rate kann Overfitting verringern.
GitHub	Versionskontrollsystem, um Projekte in Repositiories auf Webservern zu speichern. Dies ermöglicht unkomplizierte bearbeitung von Code von mehreren Programmierern.
Hyperparameter	Ein Hyperparameter ist ein Parameter, der zur Steuerung des Trainingsalgorithmus verwendet wird und dessen Wert im Gegensatz zu anderen Parametern vor dem eigentlichen Training des Modells festgelegt werden muss.4 Beispiele für Hyperparameter sind: Anzahl hidden Layers, Anzahl Knoten pro Layer, Learning Rate (für den Optimierungsalgorithmus), etc.
Klassifizierte Daten	(Englisch auch labelled data genannt) Klassifizierte Datensätze enthalten eine Anzahl von Fällen, die mit bestimmten Labels versehen sind. Ein Beispiel wäre ein Datensatz in welchem pro Zeile die Pixelwerte eines Fotos enthalten sind (Spalten 1-256) und in einer weiteren Spalte der Wert 0 oder 1 ob es sich um ein Katzenfoto oder nicht handelt. Klassifizierte Datensätze werden für das Training und das Testen von Modellen verwendet.
Layer	Die *Neuronen* (auch *Knotenpunkte*) eines künstlichen neuronalen Netzes sind schichtweise in sogenannten *Layers* angeordnet und in der Regel in einer festen Hierarchie miteinander verbunden. Die Neuronen sind dabei zumeist zwischen zwei Layers verbunden (*Inter-Neuronlayer-Connection*), in selteneren Fällen aber auch innerhalb eines Layers (*Intra-Neuronlayer-Connection*).

[4] https://de.wikipedia.org/wiki/Hyperparameteroptimierung (Zuletzt abgerufen am 20.10.2019)

	Zwischen den Layers oder Schichten ist jedes Neuron der einen Schicht immer mit allen Neuronen der nächsten Schicht verbunden. Beginnend mit der *Eingabeschicht* (*Input Layer*) fließen Informationen über eine oder mehrere *Zwischenschichten* (*Hidden Layer*) bis hin zur *Ausgabeschicht* (*Output Layer*). Dabei ist der Output des einen Neurons der Input des nächsten.[5]
Learning rate	Beim Training von neuronalen Netzwerken durch den Optimierungsalgorithmus werden für jeden Knoten Gewichte gesucht, die eine möglichst gute Vorhersage in Bezug auf den Testdatensatz ergeben. Während des Trainings werden die aktuellen Gewichte Schritt für Schritt so verändert, dass eine bessere Vorhersage resultiert. Die Schrittgrösse dieser Veränderung nennt sich Learning Rate.
Learning rate decay	Während dem Training eines neuronalen Netzwerks wird versucht, das Optimium einer Funktion zu finden (die Funktion, die die Testinputdaten mit möglichst geringem Fehler auf die Testoutputdaten abbildet). Es kann Sinn machen, die Annäherung an das Optimum bei fortschreitender Optimierung bei jedem Trainingsschritt zu verkleinern, damit das Optimum nicht übersprungen wird. Diese Verringerung der Schrittgrösse Richtung Optimum wird durch den sogenannten Learning rate decay erreicht.
Loss function	Das Training eines neuronalen Netzwerks passiert mit einem Algorithmus, der versucht, die Vorhersage möglichst den tatsächlichen Werten anzugleichen. Die Loss function gibt an, wie gross der Fehler der Vorhersage ist. Der Algorithmus versucht während des Trainings, die Loss function zu minimieren.
Model Optimizer	Der Model optimizer definiert, welcher Algorithmus für die Optimierung des neuronalen Netzwerks bzw. für die Minimierung der Loss function verwendet wird.
Neuronales Netzwerk	Das künstliche neuronale Netz (KNN) ist bis zu einem gewissen Grad dem Aufbau des biologischen Gehirns nachempfunden. Es besteht aus einem abstrahierten Modell miteinander verbundener Neuronen, durch deren spezielle Anordnung und Verknüpfung sich Anwendungsprobleme aus verschiedenen Bereichen wie der Statistik, der Technik oder der Wirtschaftswissenschaften computerbasiert lösen lassen. Das neuronale Netz ist ein Forschungsgegenstand der Neuroinformatik und Teilgebiet der künstlichen Intelligenz. Neuronale Netze müssen, bevor sie Problemstellungen lösen können, trainiert werden.[6]
Testdaten	Testdaten sind klassifizierte Datensätze und werden dazu verwendet, ein bereits trainiertes Modell zu prüfen. Dabei wird mit dem neuronalen Netzwerk eine Klassifikation der Daten durchgeführt. Diese Klassifizierung wird dann mit der wahren Klassifikation der Fälle aus dem Testdatensatz verglichen. Es kann somit ein Fehlerquote angegeben werden, wie viele Fälle das neuronale Netzwerk eine korrekt klassifiziert hat.
Trainingdaten	Klassifizierte Daten, mit welchen das neuronale Netzwerk trainiert wird.

[5] https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/ (Zuletzt abgerufen am 20.10.2019)

[6] https://www.bigdata-insider.de/was-ist-ein-neuronales-netz-a-686185/ (Zuletzt abgerufen am 20.10.2019)

10 Quellenverzeichnis

Code Snippets aus folgenden Quellen:

Machine Learning

Ng, Andrew, Coursera-Kurs: https://www.coursera.org/learn/machine-learning

Neuronale Netze selbst Programmieren – ein verständlicher Einstieg mit Python

Rashid, Tariq, Neuronale Netze selbst Programmieren – ein verständlicher Einstieg mit Python, O'Reilly Media Inc., übersetzt aus dem Englischen von dpunkt.verlag GmbH, Heidelberg (D), 1. Ausgabe, 2017

11 Versionskontrolle

Version	Datum	Beschreibung	Autor
X0.1	08.01.2020	Dokument erstellt	L.Z.
X0.2	13.01.2020	Dokument mit weiteren Kapiteln ergänzt	L.Z., R.H.
X0.3	14.01.2020	Glossar hinzugefügt, Einleitung, Zielerreichung geändert	L.Z.
X0.4	15.01.2020	Überarbeitung aller Kapitel	L.Z.
X0.5	15.01.2020	Kapitel 2 bis 5 überarbeitet	R.H.
X0.6	16.01.2020	Installationsanleitung, kleine Changes	R.H.
X0.7	17.01.2020	Letzte Änderungen	L.Z., R.H.