FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lehrstuhl für Informatik 3

Rechnerarchitektur

Marian Horn

Hyperdimensional Computing Framework in C

Computer Science Project (Master Computational Engineering)

16. Dezember 2024

Please cite as:
Marian Horn, “Hyperdimensional Computing Framework in C,” Abschlussbericht, University of Erlangen, Dept. of Computer Science, December 2024.

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik
Rechnerarchitektur

Martensstr. 3 · 91058 Erlangen · Germany

www3.cs.fau.de

Hyperdimensional Computing Framework in C

Computer Science Project (Master Computational Engineering)

vorgelegt von

Marian Horn

geb. am 15. April 2002
in München

angefertigt am

Lehrstuhl für Informatik 3
Rechnerarchitektur

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuender Professor: **Prof. Dr.-Ing. Dietmar Fey**

Einreichungsdatum: **16. Dezember 2024**

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Zielsetzung

Ziel dieses Projektes ist die Implementierung eines Frameworks für Hyperdimensional Computing (HDC) in der Programmiersprache C. Es soll ähnlich zur Pythonbibliothek TorchHD Methoden zur Generierung, Manipulation und Verwaltung von Hypervektoren sowie Strukturen für Training und Inferenz nach dem HDC-Konzept bereitstellen.

Der genutzte Encoding-Algorithmus für Zeitserien basiert auf dem Paper [1] und wird im Rahmen des Projekts *NOE-EMY* am Lehrstuhl für Rechnerarchitektur (INF3) für die Klassifizierung der Sprunggelenkposition anhand von EMG-Messungen am Bein adaptiert und evaluiert.

Zudem bietet das Framework eine anwendungsunabhängige Vorlage, die den Aufbau weiterer HDC-Modelle ermöglicht und erleichtert.

Das Framework ist verfügbar im zugehörigen Repository unter:
[GitLab Repository]

1.2 Vorgehen

Zunächst wurde das grundlegende Framework für HDC aufgebaut, das in Kapitel 2 beschrieben ist. Dabei wurde beachtet, dass das Framework generisch für verschiedenste HDC-Anwendungen genutzt werden kann und nicht nur spezifisch auf die EMG-Klassifikation ausgelegt ist.

Zum Testen des Frameworks konnte die Klassifikation von Hand-EMG-Daten analog zum Paper [1] in C implementiert und die Ergebnisse validiert werden.

Da diese Arbeit in das Forschungsprojekt *NOE-EMY*, welches eine Orthese zur Unterstützung von teilgelähmten Sprunggelenken entwickelt, eingebettet ist, sollte das funktionierende Modell der Hand für die Klassifikation von EMG-Daten, die die Position des Sprunggelenks repräsentieren, adaptiert werden. Dafür wurden Datensätze verwendet, die vom Lehrstuhl "Neuromuscular Physiology and Neural Interfacing" aufgezeichnet und mittels Butterworth-Filter, Root-Mean-Squared-Value, Skalierung auf einen Durchschnittswert von 0 mit Standardabweichung 1 und anschließender Projektion in den Bereich [-1,1] vorverarbeitet worden sind. Nach Anpassung des Systems und seiner Konstanten konnten einige Systemvariablen (wie z.B. die Länge der Zeitserien) für die spezifische Anwendung optimiert werden.

Des Weiteren wurde das Programm dahingehend erweitert, dass auch eine Online-Klassifikation durchgeführt beziehungsweise simuliert werden kann. Hierfür wird ein Strom von Testdaten, die gruppiert in Batches eintreffen, klassifiziert, indem pro Batch das wahrscheinlichste Label über alle enthaltenen Zeitseriensequenzen gewählt wird. Die Fähigkeit zur Verarbeitung der Daten in Echtzeit ist essenziell, da in der Orthese immer nur kleine, live gemessene Datenmengen und nicht ein großer Testdatensatz auf einmal klassifiziert werden.

Um die Anwendung des Frameworks in anderen Bereichen als der Zeitserienklassifikation von EMG-Daten zu erleichtern, wurde das Programm um Funktionen zum allgemeinen Encoding von Daten, die aus Feature-Wert-Paaren bestehen aber zeitunabhängig sind, erweitert und eine Vorlage zur Erstellung neuer HDC-Modelle hinzugefügt.

Kapitel 2

Programmstruktur

2.1 Konfiguration

Zur Nutzung des Frameworks müssen zunächst folgende systemspezifische Konstanten in einer Konfigurations-Datei wie `/customModel/configCustom.h` definiert werden, die von den übrigen Programmteilen importiert wird:

Name	**Bedeutung**
BIPOLAR_MODE	Definiert, ob bipolare (-1/1) oder binäre (0/1) Vektoren verwendet werden
VECTOR_DIMENSION	Dimension der Hypervektoren (Anzahl der Elemente pro Vektor)
NUM_LEVELS	Anzahl der Diskretisierungsstufen für den Datensatz
MIN_LEVEL	Minimalwert im Datensatz
MAX_LEVEL	Maximalwert im Datensatz
NUM_CLASSES	Anzahl der Klassen/Labels im Datensatz
NUM_FEATURES	Anzahl der Features (z. B. EMG-Kanäle)
N_GRAM_SIZE	Anzahl der Zeitschritte in einer Zeitserie
DOWNSAMPLE	Downsampling-Rate im Preprocessing
NORMALIZE	Normalisierung der Associative Memory (nur für bipolare Vektoren)
CUTTING_ANGLE_THRESHOLD	Schwellwert für den Winkel während des Trainings (bipolarer Fall)
PRECOMPUTED_ITEM_MEMORY	Aktiviert eine für alle Feature-Wert-Kombinationen vorberechnete Item Memory

WINDOW	Größe des Fensters für Zeitserienklassifikation im fensterbasierten Modus
OUTPUT_MODE	Steuerung der Konsolenausgaben (keine, basic, detailed, debug)

Tabelle 2.1 – Konfigurationsparameter im HDC-Modell

Standardwerte für diese Systemkonstanten können `/foot/configFoot.h` oder `/customModel/configCustom.h` entnommen werden.

2.2 Vektoren

Die Basis des gesamten Modells bilden hyperdimensionale Vektoren. Diese besitzen üblicherweise 10,000 Dimensionen und enthalten die Werte 0 und 1 im binären und -1 und 1 im bipolaren Modus.

In `/hdc_infrastructure/vector.c` wird ein Datenstruktur für Vektoren definiert und Methoden für die sowohl vorinitialisierte `create_vector()` als auch uninitialisierte `create_uninitialized_vector()` Erstellung, sowie die Ausgabe (`print_vector()`) und Deallokation (`free_vector()`) bereitgestellt.

2.3 Operationen

Zur Verarbeitung der Vektoren existieren drei verschiedene Operationen, die in `/hdc_infrastructure/operations.c` implementiert sind.

Binding (`bind()`) wird durch elementweise Multiplikation (bipolar) oder XOR (binär) realisiert. Der Ergebnisvektor kombiniert die Eingangsvektoren auf eine Weise, die eine einzigartige Verknüpfung darstellt. Damit schafft Binding es, verschiedene Merkmale oder Kontextinformationen miteinander zu assoziieren.

Bundling funktioniert im bipolaren Fall durch elementweise Addition. Im Binären hingegen muss ein Majority Voting durchgeführt werden. Der Ergebnisvektor erhält dabei den Wert, der am öftesten bei den zu bundelnden Vektoren am jeweiligen Element vorkommt. Daher kann Bundling im binären Modus nicht inkrementell funktionieren, sondern muss für alle Vektoren gleichzeitig in einem Schritt geschehen. Daher wird zusätzlich zur einfachen Funktion (`bundle()`) eine erweiterte Möglichkeit (`bundleMulti()`) eingeführt, die als Argument anstatt zwei Vektoren beliebig viele Vektoren in Form einer Matrix erwartet.

Permutation (`permute()`) ist eine zyklische Verschiebung der Vektoren um eine definierte Anzahl von Elementen und kann zur Kodierung temporaler Abhängigkeiten genutzt werden.

Zum Vergleich von Vektoren (`similarityCheck()`) wird die Hamming-Distanz (binärer Fall) oder der Cosinus verwendet. Das Ergebnis wird auf ein Intervall zwischen -1 (diametral entgegengesetzte Vektoren) und 1 (identische Vektoren) abgebildet.

2.4 Item Memory

Die Item Memory speichert alle Basisvektoren und kann diese generieren, exportieren (`store_item_mem_to_bin()`), importieren (`load_item_mem_from_bin()`), deallokieren (`free_item_memory()`), abfragen (`get_item_vector()`) oder ausgeben (`print_item_memory()`).

Die Initialisierung und Generierung der Basisvektoren kann entweder als diskrete Item Memory mit zufälligen orthogonalen Vektoren (`init_item_memory()`) oder als kontinuierliche Item Memory mit unidistanten Vektoren durch Interpolation (`init_continuous_item_memory()`) erfolgen. Ersteres sollte zur Kodierung der Features, zweiteres zur Kodierung der diskretisierten Werte genutzt werden.

Als experimentelle Erweiterung kann in der Konfigurationsdatei durch Setzen von `PRECOMPUTED_ITEM_MEMORY = 1` die Nutzung einer einzigen, größeren Item Memory aktiviert werden. Diese enthält bereits Basisvektoren für jede mögliche Feature-Wert-Kombination und ersetzt die getrennten Item Memories für Features und Werte (`init_precomp_item_mem()`).

Je nach Anzahl der Features (`NUM_FEATURES`) und Diskretisierungsschritte (`NUM_LEVELS`) kann dies jedoch zu erheblichem Speicherverbrauch führen. Gleichzeitig werden die elementaren Binding-Operationen zu Beginn des Encodings eingespart, was den Rechenaufwand reduziert.

Die Option `PRECOMPUTED_ITEM_MEMORY` stellt somit einen Trade-off zwischen Speicherverbrauch und Rechenaufwand dar und kann je nach Anwendungsfall sinnvoll eingesetzt werden.

2.5 Associative Memory

Die Associative Memory speichert für jede zu unterscheidende Klasse einen Klassenvektor, der in der Trainingsphase gelernt wird. Es existieren Methoden für die Initialisierung (`init_assoc_mem()`), Deallokation (`free_assoc_mem`), Export (`store_assoc_mem_to_bin()`), Import (`load_assoc_mem_from_bin()`) und Normalisierung nach dem Training im bipolaren Modus (`normalize()`). Für die Klassenvek-

toren bestehen Möglichkeiten zum Hinzufügen (`add_to_assoc_mem()`), Abfragen (`get_class_vector()`) und Ausgeben (`print_class_vector()`).

Im bipolaren Modus erwartet die Funktion `add_to_assoc_mem()` einen Vektor, der eine einzelne Zeitserie repräsentiert. Dieser wird mit dem bisherigen Klassenvektor verglichen und auf diesen gebundled, wenn der Vergleichswinkel größer als `CUTTING_ANGLE_THRESHOLD` ist. Dagegen müssen im binären Modus alle Zeitserien zuerst gebundled werden, `add_to_assoc_mem()` speichert den erhaltenen Klassenvektor dann ohne weitere Überprüfungen direkt ab.

Die Funktion `classify()` ist essentiell für die Evaluation des Modells und vergleicht einen Testvektor mit allen gespeicherten Klassenvektoren. Die Klasse mit dem kleinsten Differenzwinkel wird dann als Klassifizerungsergebnis zurückgegeben.

2.6 Encoder

Der Encoder enthält die entscheidende Logik des HDC-Modells und den Algorithmus, der die Trainings- und Testdaten in den hyperdimensionalen Raum abbildet. Hierfür bietet er je nach Anwendungsfall die Funktionen `encode_timeseries()` oder `encode_general_data()` an.

Zur Initialisierung (`init_encoder()`) erwartet der Encoder Pointer auf die bereits initialisierten Item Memories.

Der Algorithmus, der hinter dem Encoding von Zeitserien (`encode_timeseries()`) steckt, ist im Paper [1] detailliert beschrieben. `encode_general_data()` funktioniert als vereinfachte Version dieses Algorithmus mit einer Zeitserienlänge von 1, um auch zeitunabhängige Probleme encoden zu können.

2.7 Trainer

Im Trainer wird sämtliche übrige Logik, die zum Errechnen der Klassenvektoren aus den Trainingsdaten benötigt wird, gebündelt. Der Datensatz wird durchiteriert und jedes Sample oder jede Zeitserie vom Encoder in einen Vektor transformiert. Diese Vektoren werden dann klassenweise zu Klassenvektoren gebundled und zur Associative Memory hinzugefügt. Die Funktionen sind für Zeitserien (`train_model_timeseries()`) oder allgemeine Klassifikation (`train_model_general_data()`) verfügbar.

2.8 Evaluator

Der Evaluator übernimmt die Klassifikation der Testdaten, indem er sie mit demselben Encoder wie im Training kodiert. Anhand der tatsächlichen Labels wird anschließend die Klassifizierungsgenauigkeit sowie eine Verwechslungsmatrix berechnet.

Der Evaluator verfügt über verschiedene Modi:
`evaluate_model_timeseries_direct()` klassifiziert alle Zeitserien aus den Testdaten direkt.
`evaluate_model_timeseries_with_window()` teilt die Testdaten in Zeitfenster der Länge `WINDOW_SIZE` auf. Innerhalb jedes Fensters, das mehrere Zeitserien umfasst, wird für jede Zeitserie ein Label und ein Zuverlässigkeitslevel (Confidence) berechnet. Das Label der Zeitserie mit der höchsten Zuverlässigkeit entscheidet schließlich über das Label des gesamten Fensters.
Zusätzlich zur allgemeinen Testgenauigkeit berechnen beide Evaluationsmethoden für Zeitserien eine hypothetische Genauigkeit. Hierbei werden Zeitserien ignoriert, deren Label sich in ihrem Verlauf verändert.

Für zeitunabhängige Modelle kann `evaluate_model_general_direct()` genutzt werden.

2.9 Online Klassifizierung

Alternativ zum Evaluator kann für Zeitserien auch `/hdc_infrastructure/online_classifer.c` genutzt werden, womit sich die Klassifizierung von live eingehenden Testdaten simulieren oder durchführen lässt. Ein solcher Klassifizierer muss mit `init_online_classifier()` initialisiert werden. Dabei müssen Pointer zu einer trainierten Associative Memory, dem dafür genutzten Encoder und eine Batchsize, die definiert, wie viele Samples gleichzeitig zur Klassifikation eintreffen, angegeben werden.
Daraufhin kann das aufrufende Programm für jede eintreffende Batch an Testdaten mit `calculateUpdate()` deren Klasse bestimmen. Die Evaluation entspricht dabei `evaluate_model_timeseries_with_window()` wobei jede Batch als ein Fenster betrachtet wird.

2.10 Main

Das Hauptprogramm muss je nach Anwendungsfall individuell zusammengestellt werden, folgt aber immer einem ähnlichen Schema unter Verwendung der bisher vorgestellten Funktionalitäten.

Zuerst müssen die nötigen Item Memories, der Encoder und die Associative Memory initialisiert werden. Außerdem müssen alle Trainings- und Testdaten eingelesen werden, sodass sie über Pointer referenzierbar sind. Hierfür kann die Funktion `getData()` je nach Datensatz angepasst werden. Die Daten sollen als 2D-Array von `double`-Werten vorliegen, wobei jede Zeile ein Sample und jede Spalte ein Feature repräsentiert. Die zugehörigen Labels werden als 1D-Array von einem `int`-Element pro Sample erwartet.
Anschließend kann das Modell mit den Trainingsdaten trainiert und mit den Testdaten evaluiert werden.

Um Speicherprobleme zu vermeiden, sollten alle initialisierten Strukturen nach der Nutzung mit den entsprechenden Funktionen wieder freigegeben werden.

Kapitel 3

Anwendung

3.1 EMG-Klassifikation Fuß

Im Rahmen des Projektes *NOE-EMY* am Lehrstuhl für Rechnerarchitektur (INF3) soll unter anderen das HDC-Modell für die Klassifizierung von Fußbewegungen evaluiert werden. Dazu werden Datensätze von zwei gesunden und zwei beeinträchtigten Testpersonen genutzt. Diese enthalten vorverarbeitete EMG-Messungen, die mit 32 Elektroden (Features) am Unterschenkel aufgenommen wurden. In den Messungen wird zwischen Ruheposition, Fuß heben bzw. senken sowie Fuß nach innen und außen neigen unterschieden und jeder Position ein Label von 0 bis 4 zugewiesen. Jede Position wurde dabei für 10 Sekunden gehalten.

Das entsprechende HDC-Modell in `/foot/modelFoot.c` kann mit `make foot` kompiliert und anschließend mit `./modelFoot` ausgeführt werden. Es eignet sich auch als Beispielsystem, um den Einfluss verschiedener Systemparameter und Ausführungsmodi zu testen, die in `/foot/configFoot.h` definiert und modifiziert werden können.

Dabei können im Beispiel Genauigkeiten von bis zu 99% erreicht werden. Die Ergebnisse schwanken jedoch je nach Testperson und gewählten Parametern erheblich.

3.2 Individuelle Nutzung

Um das HDC-Framework auch für unabhängige Anwendungen nutzen zu können, wird im Ordner `/customModel` eine Vorlage zur Entwicklung eigener HDC-Modelle bereitgestellt. Das Modell kann mit `make custom` kompiliert und anschließend mit `./modelCustom` ausgeführt werden. Es ist bereits in der Rohfassung lauffähig und enthält ein einfaches Beispielproblem mit 3 Klassen, 4 Features sowie Trainings- und

Testdaten, die in `/customModel/dataReaderCustom.c` mit jeweils einem Sample pro Klasse und Datensatz definiert sind. Hierbei wird jeweils ein Sample pro Klasse verwendet.

Für eine eigene Nutzung muss die Methode `getData()` individuell implementiert werden, um die spezifischen Trainings- und Testdaten einzulesen. Die Daten sollen dabei als 2D-Array von `double`-Werten vorliegen, wobei jede Zeile ein Sample und jede Spalte ein Feature repräsentiert. Die zugehörigen Labels werden als 1D-Array mit einem `int`-Element pro Sample erwartet.

Anschließend müssen die entsprechenden Konstanten in `/customModel/configCustom.h` angepasst werden, und das neue Modell ist einsatzbereit.

Quellenverzeichnis

[1] A. Rahimi, S. Benatti, P. Kanerva, L. Benini und J. M. Rabaey, „Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition,“ in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, S. 1–8 (siehe S. 1, 2, 6).