DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Entwicklung einer DSL zum Rechnen mit mathematischen Formeln für Anwendungen im Maschinellem Lernen

Studienarbeit

für die Prüfung zum

Bachelor of Science

des Studienganges Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Sebastian Bernauer

Abgabedatum XX.XX.20XX

Bearbeitungszeitraum	XX Wochen
Matrikelnummer	7390071
Kurs	TINF16B5
Ausbildungsfirma	United Internet AG Karlsruhe
Betreuer	Oliver Rettig Marcus Strand

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: „Entwicklung einer DSL zum Rechnen mit mathematischen Formeln für Anwendungen im Maschinellem Lernen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum	Unterschrift

Zusammenfassung

TODO

Inhaltsverzeichnis

KAPITEL 1

Motivation

Immer öfter werden Probleme der realen Welt mittels neuronaler Netze gelöst. Allerdings kann man in den meisten Fällen nicht einfach das neuronale Netz auf die gemessenen Daten angewendet werden. Die Daten müssen vorher aufbereitet werden und gegebenenfalls unwichtige Daten entfernt werden. Dies geschieht mittels verschiedener Frameworks in verschiedenen Sprachen. Es soll eine Domain-specific language (DSL) entwickelt werden, die genau auf diesen Anwendungsfall zugeschnitten ist. Durch die DSL soll einen einheitliche Sprache geschaffen werden, welche das Vorprozessieren der Daten vereinfacht. Die Sprache soll plattformübergreifend sein und Central processing unit (CPU)- und Graphics processing unit (GPU)-Berechnungen ermöglichen.

KAPITEL 2

Überblick über Technologien

Das Aufbereiten der Daten für das neuronale Netz kann in mehreren Frameworks in mehreren Sprachen erfolgen. Oft wird für das Vorverarbeiten die gleiche Sprache wie für das neuronale Netz verwendet. Die gängigsten Frameworks sind in Tabelle 2.1 gelistet.

Framework	**Sprache**
Tensorflow	Python
DL4J	Java

Tabelle 2.1: Die gängigsten Frameworks für maschinelles Lernen

2.1 Unterschied Compiler, Transpiler und Interpreter

Compiler

Übersetzt von einer höheren Sprache in eine niedrigere Sprache.

Transpiler

Übersetzt zwischen zwei Sprachen mit ungefähr gleichem Abstraktionsgrad.

Interpreter

Führt Code einer höheren Sprache direkt aus.

2.2 ND4J

ND4J ist eine Framework für die Sprache Java, in welchem effiziente Matrizenoperationen durchgeführt werden können. Es kann auf der CPU oder GPU ausgeführt werden. In ND4J ist größtenteils nur das Konstrukt einer Matrix bekannt, Vektoren sind nur ein Spezialfall einer Matrix.

KAPITEL 3

Aufgabenstellung

3.1 Zu Grunde liegendes Framework

In dem Umfeld der Arbeit hat sich das Framework ND4J in der Programmiersprache Java für den Praxiseinsatz durchgesetzt. Daher soll die in dieser Arbeit entwickelte DSL auf diesem Framework aufbauen.

3.2 Compiler oder Interpreter?

Die DSL ist für das Vorprozessieren von Daten für maschinelles Lernen. Für die DSL kommt ein Compiler oder Interpreter in Frage. Ein Transpiler ist nicht möglich, da es keine in der Praxis verwendete Sprache für das Vorprozessieren der Daten gibt, in die übersetzt werden kann. In der Tabelle 3.1 werden die Implementierungsmöglichkeiten mittels Compiler und Interpreter gegenüber gestellt.

Implementierung	**Vorteile**	**Nachteile**
Compiler	Generierter Java-Code kann auf jeder Java virtual machine (JVM) ausgeführt werden, es wird kein Interpreter benötigt.	Debugging ist nur in dem generierten Java-Code möglich.
Interpreter	Debugging leichter möglich	Möglicherweise nicht so performant

Tabelle 3.1: Vor- und Nachteile einer Implementierung mittels Compiler oder Interpreter

TODO Antlr TODO Graal + Truffle

3.3 Verwendete Technologien

Die DSL wird mittels einem Interpreter ausgeführt. Dieser baut auf folgenden Technologien auf:

ND4J
Matrizen-Berechnungen werden mittels dem ND4J-Framework durchgeführt.

Java
Das ND4J-Framework ist in der Programmiersprache Java verfügbar. Damit der Interpreter es verwenden kann, wird in dieser Sprache geschrieben.

Graal
TODO

Antlr
Antlr v4 wird für das Parser der eingegebenen Programme verwendet.

KAPITEL 4

Design der DSL

Funktionen mit Parametern (copy parameter(?)). Die DSL soll dynamisch typisiert sein. Variablen müssen zuvor mit Typ definiert werden. Alle Variablen sind intern ein INDArray. Debugging ist wichtig.

KAPITEL 5

Implementierung der DSL

5.1 Interpreter oder Compiler?

=> Interpreter

5.2 Grundrechenarten

Bei der Darstellung der Zahlen im Speicher wird für jede Zahl ein Double verwendet. Das erhöht die Genauigkeit gegenüber einem float und erspart Konvertierungen zwischen Ganz- und Fließzahlen. Grundrechenarten sind die Addition, Subtraktion, Multiplikation und Division von Matrizen. Sie Operationen sind elementweise und werden direkt von ND4J zur Verfügung gestellt.

5.3 Kreuzprodukt

Anders als die vier Grundarten aus dem vorherigen Abschnitt wird das Kreuzprodukt von ND4J nicht als Operation angeboten. Das Kreuzprodukt wird in der Praxis allerdings zu Beispiel für das Aufspannen von Vektoren im dreidimensionalen Raum benötigt. Daher wird an dieser Stelle das Kreuzprodukt zweier Vektoren selber implementiert. Falls ND4J in Zukunft den Operator Kreuzprodukt bereit stellt, kann in zukünftigen Varianten auf ihre Implementierung zugegriffen werden, da diese wahrscheinlich effizienter sein wird. Die eigene Implementierung des Kreuzprodukts kann auf folgende Arten geschehen:

1. Implementierung in Plain Java mittels einer for-Schleife.

2. Implementierung in Plain Java mittels Subarrays

5.3.1 Implementierung Kreuzprodukt mittels For-Schleife

Bei der Implementierung mittels der For-Schleife werden die eigentlichen Berechnungen in Java durchgeführt. Als erstes wird ein Double-Array als Zwischenspeicher für das Ergebnis

angelegt. Es besitzt (Anzahl der Zeitelemente in den Eingabe-Vektoren) * drei Elemente. Die Anzahl der Elemente entspricht so der Anzahl der Elemente der Ergebnismatrix, die Daten können in dem Double-Array effizient gespeichert werden. Das Array wird im Anschluss in eine Matrix konvertiert. Für die Berechnung der Elemente wird mittels einer For-Schleife über alle Zeilen der Matrix iteriert. In jeder Zeile werden die 3 Werte des entstehenden Ergebnisvektors berechnet und in das Double-Array gespeichert. Abschließend wir das Double-Array in eine Matrix mit den Dimensionen [<Anzahl Zeitelemente> x drei] konvertiert und durch eine Vector3-Wrapper-Klasse als Vector mit 3 Werten gekennzeichnet. Der Algorithmus ist in Codefragment 5.1 dargestellt.

```
private Vector3 crossProduct(Vector3 left, Vector3 right) {
    INDArray a = left.getNdArray();
    INDArray b = right.getNdArray();

    int size = a.shape()[0];
    double[] result = new double[size * 3];

    for (int i = 0; i < size; i++) {
        result[i * 3 + 0] = a.getDouble(i, 1) *
           b.getDouble(i, 2) - a.getDouble(i, 2) *
           b.getDouble(i, 1);
        result[i * 3 + 1] = a.getDouble(i, 2) *
           b.getDouble(i, 0) - a.getDouble(i, 0) *
           b.getDouble(i, 2);
        result[i * 3 + 2] = a.getDouble(i, 0) *
           b.getDouble(i, 1) - a.getDouble(i, 1) *
           b.getDouble(i, 0);
    }
    return new Vector3(Nd4j.create(result, new int[]{size,
       3}));
}
```

Codefragment 5.1: Implementierung Kreuzprodukt mittels for-Schleife

5.3.2 Implementierung Kreuzprodukt mittels Subarrays

```
private Vector3 crossProductSubArray(Vector3 left, Vector3
   right) {
    INDArray a = left.getNdArray();
    INDArray b = right.getNdArray();

    INDArray a1 = a.getColumn(0);
    INDArray a2 = a.getColumn(1);
    INDArray a3 = a.getColumn(2);

    INDArray b1 = b.getColumn(0);
    INDArray b2 = b.getColumn(1);
    INDArray b3 = b.getColumn(2);

    INDArray c1 = (a2.mul(b3)).sub(a3.mul(b2));
    INDArray c2 = (a3.mul(b1)).sub(a1.mul(b3));
    INDArray c3 = (a1.mul(b2)).sub(a2.mul(b1));

    int size = a.shape()[0];
    INDArray result = Nd4j.create(size, 3);
    result.putColumn(0, c1);
    result.putColumn(1, c2);
    result.putColumn(2, c3);

    return new Vector3(result);
}
```

Codefragment 5.2: Implementierung Kreuzprodukt mittels for-Schleife

5.4 Vergleich der Implementierungen

Beide Impelemtierungsmöglichkeiten haben Vor- und Nachteile. Diese sind in Tabelle 5.1 auf der nächsten Seite aufgeführt.

Implementierungs-möglichkeit	**Vorteile**	**Nachteile**
Mittels For-Schleife	Leichter verständlich.	Wird direkt in Java ausgeführt. Mögliche Optimierungen von ND4J können nicht verwendet werden. Berechnungen finden nur auf der CPU statt!
Mittels Subarray	Durch die Verwendung von ND4J können die Optimierungen verwendet werden. Wenn ND4J so konfiguriert ist, dass es auf der GPU läuft, kann die eigentliche Berechnung weiterhin auf der GPU erfolgen.	Schwerer verständlich.

Tabelle 5.1: Vor- und Nachteile einer Implementierung mittels Compiler oder Interpreter

Für große Zeitreihen müsste sich die Implementierung mittels dem Subarray als effizienter erweisen, besonders wenn die Berechnungen auf der GPU durchgeführt werden. Als Nachweis und für das Effizienzverhalten bei kleinen Zeitreihen wurde ein Benchmark durchgeführt. Die Ergebnisse sind in Tabelle 5.2 festgehalten.

Anzahl Datensätze	**For-Schleife**	**Subarray**
100	6 (342)	4 (13)
1.000	20 (489)	7 (22)
10.000	53 (527)	5 (21)
100.000	333 (1159)	5 (15)
1.000.000	3205 (4129)	47 (64)
10.000.000	32358 (33594)	442 (453)

Tabelle 5.2: Benchmark-Ergebnisse in ms der verschiedenen Implementierungsmöglichkeiten für das Kreuzprodukt. Die Messung ist nach 5 Durchläufen gemessen, die Zahl in Klammern gibt die Zeit des ersten Durchlaufs an.

3 Varianten

Benchmarks! Klassifizierung CPU oder GPU-Workload.

Möglicherweise lohnt sich die eher GPU-betonte Variante erst ab gewisser Größe. => Dann mit konstantem Aufwand entscheiden, welches Verfahren. Vom Nutzer (während Laufzeit) auswählbar?

Glossar

Inhalt

Daten, welche auf den Portal-Homepages[1] angezeigt werden, z.B. Lottodaten, Wetterdaten, Bundesliga-Liveticker und das Horoskop.

Portal-Homepage

Die Startseite einer der Portale web.de, gmx.net, gmx.ch, gmx.at und home.1und1.de.

[1] Die Startseite einer der Portale web.de, gmx.net, gmx.ch, gmx.at und home.1und1.de

Abkürzungsverzeichnis

DSL Domain-specific language . . . 3

JVM Java virtual machine . . . 5

CPU Central processing unit . . . 3

GPU Graphics processing unit . . . 3

Abbildungsverzeichnis

Literatur

[1] Oracle. *Using Java Reflection*. 1988. URL: http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html [besucht am 04. 09. 2017].