Übungsblatt: 1

Abgabe bis spätestens 19. Mai 2024 23:59:59 GMT+1

Aufgabe 1: Ablauf der folgenden Übungen

Ziel dieses und des nächsten Übungsblattes ist die Implementierung Ihres eigenen Frameworks für Deep Learning in Python. Neben der Python standard Library dürfen sie ausschließlich *Numpy* verwenden, sowie eine Library ihrer Wahl, die nur dazu verwendet wird den MNIST Datensatz herunterzuladen.

Klausurbonus Sie können in Gruppen von bis zu drei Studierenden arbeiten, um einen Bonus für die Klausur zu erhalten. Hier sind die dafür notwendigen:

(a) Kontaktieren Sie Kirill Djebko per E-Mail (kirill.djebko@uni-wuerzburg.de), um ein Gitlab Repository zu erhalten.

(b) Laden Sie Ihren Code für *jede* Unteraufgabe (z.B. 2 b) mindestens einmal in das Repository hoch.

(c) Stellen Sie sicher, dass Sie alle Aufgaben der ersten beiden Übungsblätter funktionsfähig implementiert haben. Dazu gehört auch, dass das Framework in der Lage ist Training auf dem MNIST Datensatz in sinnvoller Zeit durchzuführen.

(d) Stellen Sie sicher, dass Sie den Code für jedes Übungsblatt fristgerecht ins Repository hochladen. Wenn Sie mit dem Blatt fertig sind, erstellen Sie einen *Tag* aus dem *Commit*, der das Ende ihrer Programmierarbeiten für das Übungsblatt markiert. Der Commit muss *vor Ablauf der Deadline* hochgeladen worden sein.

(e) Erstellen Sie im Repository eine kurze Anleitung als README.md, wie ihr Framework zu verwenden ist. Wenn Sie externe Bibliotheken verwenden, um die MNIST-Daten herunterzuladen, fügen Sie bitte auch eine requirements.txt-Datei hinzu, die den Namen der Bibliothek enthält.

(f) Im WueCampus-Kurs finden Sie Tests, die jedes implementierte Layer bestehen muss. Sie können die Struktur der Tests an Ihre Implementierung anpassen, müssen jedoch die Testwerte beibehalten.

(g) Um den Bonus zu erhalten, müssen Sie als Gruppe nach Abschluss der Programmierarbeiten Ihren Code erklären können. Dies wird durch ein Gespräch erfolgen, in dem Sie die Funktionsweise Ihres Codes und die Entscheidungen, die Sie bei der Implementierung getroffen haben, erläutern. Der Ablauf der Terminvergabe wird vorher rechtzeitig beschrieben.

Hinweis: Implementieren Sie zuerst die Funktionalität und optimieren Sie Ihren Code anschließend in Hinsicht auf Laufzeit. Verwenden Sie wo immer möglich slicing anstelle von Schleifen

Aufgabe 2: Ihr eigenes kleines Deep Learning Framework

(a) Modellieren Sie die Daten wie folgt:

1. Eine Tensorklasse, in der die **elements**, die **deltas** sowie ein **Shape** gespeichert ist.
2. Eine Klasse **Shape**, mit der Sie die Dimension Ihrer Daten kennzeichnen.

(b) Modellieren Sie Ihr Netzwerk wie folgt:

1. Ein **InputLayer** transformiert die Eingabedaten in für das Netz lesbare Tensoren.
2. Mehrere **Layer** verarbeiten die Daten entsprechend Ihrer Definition
3. Ein (oder mehrere) Lossfunktionen, die das Ergebnis des letzten Tensors bewerten
4. Das Netzwerk sollte über die Methoden **backprop()**, sowie über eine Methode **forward()** verfügen, in denen eingegebene Daten durch das Netzwerk geschickt werden.

(c) Modellieren Sie Ihre Layer mit mindestens den folgenden Methoden:

- **forward()**
- **backward()**
- **calculate_delta_weights()**, falls Parameter im **Layer** sind

(d) Zu implementierende **Layer**:

- FullyConnected
- Activation (Relu, Sigmoid, TanH)
- Softmax

(e) Zu implementierende **Losses**:

- Cross-Entropy
- Mean-Squared Error

(f) Optimizer

Implementieren Sie den Standard Stochastik Gradient Descent, wie in der Vorlesung gezeigt.

(g) Gewichtsinitialisierung

Initialisieren Sie alle Gewichte zufällig im Intervall von [-0.5, 0.5]

(h) Laden und Speichern

Implementieren Sie Funktionen, um die Gewichte ihres Netzwerkes zu speichern und zu laden. Erweitern Sie hierzu ihre Netzwerk-Klasse um die Methoden “save_params(self, folder_path)” und “load_params(self, folder_path)”, die einen Pfad zu einem Ordner erhalten, um die Gewichte zu speichern und zu laden. Bei Speichern sollen die Gewichte aller Layer mit Parametern in diesem Ordner abgespeichert werden (z. B. als .pkl-Dateien mithilfe des Pickle Python-Moduls). Beim Laden sollen die Gewichte korrekt geladen werden. Sie können hierfür vereinfacht jedem Layer ein Attribut “layer_type” und “num” zuweisen, aus denen sich ein eindeutiger Bezeichner für den Layer ergibt, den Sie als Dateinamen für

die gespeicherten Gewichte verwenden können. Verwenden Sie zum konkatenieren von Pfaden "os.path.join()". Sie können vereinfacht die Gewichte und den Bias jedes Layers als getrennte Dateien abspeichern. Beim Aufruf der Trainings-Methode soll mithilfe eines booleans ausgewählt werden können, ob das Netz trainiert werden soll (und in diesem Fall die neuen Gewichte gespeichert werden), oder ob stattdessen bestehende Gewichte geladen werden sollen.

Aufgabe 3: MNIST

(a) Testen Sie Ihre Implementierung mit dem MNIST-Datensatz. Lassen Sie Ihre Implementierung für 10-20 Epochen laufen und notieren Sie für jede Epoche die Epochennummer, die Laufzeit der Epoche und den durchschnittlichen Loss der Epoche. Notieren Sie außerdem die Accuracy des trainierten Netzwerks auf den Testdaten. Ihre Implementierung sollte eine Accuracy von $> 95\%$ erreichen (dies ist jedoch kein striktes Kriterium sondern dient nur als Hilfe). Welche Beobachtungen machen Sie? Schreiben Sie Ihre Notizen und Beobachtungen in die README.md-Datei und committen Sie sie zusammen mit den gespeicherten Netzwerk-Gewichten in das GitLab-Repository.