Mrgim Demku - https://mergimdemku.github.io/lstm-language-model/

Github Repository: https://github.com/mergimdemku/lstm-language-model

---

Projektdokumentation

Titel: LSTM Language Model mit TensorFlow.js

---

1. Einleitung

Im Rahmen des Mastermoduls „Deep Learning“ wurde die Aufgabe gestellt, ein Language Model (LM) zur Wortvorhersage mit einem rekurrenten Long Short-Term Memory (LSTM) Netzwerk zu entwickeln und dieses mittels TensorFlow.js (TFJS) als Webanwendung interaktiv bereitzustellen. Ziel war es, eine benutzerfreundliche Oberfläche zu schaffen, über die Texteingaben gemacht und Wortvorhersagen angezeigt werden können.

---

2. Projektbeschreibung

2.1 Daten und Training

- Trainingsdaten: ca. 200.000 Wörter, 39.774 Vokabeln, 209.158 Trainingsbeispiele
- Hardware: Training erfolgte auf einem leistungsfähigen Master-PC mit 128 GB RAM und NVIDIA RTX 1060 GPU (32 GB RAM im Laptop reichten nicht aus)
- Trainingsparameter:
    - Netzwerkarchitektur: 2 gestapelte LSTM-Schichten mit je 100 Units
    - Optimizer: Adam mit Lernrate 0,01
    - Loss: Categorical Cross-Entropy
    - Batch-Size: 32

- Trainingsdauer und Ressourcenverbrauch: ca. 10 Epochen, maximaler RAM-Verbrauch 93,3 GB, nach Epoch 4 Rückgang auf ca. 46 GB

- Endergebnis: finale Trainingsgenauigkeit von ca. 13,9 %, Loss 5,17

2.2 Modellkonvertierung und Webintegration

- Speicherung des Modells im Keras-Format (`.keras`)

- Mehrere Versuche zur Konvertierung in TensorFlow.js-Format mittels `tensorflowjs_converter`, zunächst lokal mit Problemen bei Abhängigkeiten und Kompatibilität

- Nutzung von Google Colab zur erfolgreichen Konvertierung

- Entwicklung einer Web-App mit HTML und JavaScript: Laden von Modell und Tokenizer, Verarbeitung von Texteingaben, Vorhersage des nächsten Wortes, automatische Mehrwortvorhersage und Reset-Funktion

2.3 Tokenizer

- Ursprünglicher Tokenizer enthielt nicht direkt den erforderlichen `word_index`, sondern nur Wortzählungen als JSON-String

- Entwicklung eines Python-Skripts zur Extraktion und separaten Speicherung von `word_index` zur Nutzung in der Web-App

---

3. Herausforderungen und Probleme

- Fehlende Definition des Input-Layers mit `inputShape` im Modell führte zu Lade- und Laufzeitfehlern in TensorFlow.js

- `model.predict()` konnte wegen fehlender Initialisierung nicht verwendet werden

- Tokenizer musste aufwendig konvertiert werden, um im Browser nutzbar zu sein

- Zeit- und Ressourcenrestriktionen erschwerten die Fertigstellung und umfassende Tests

- Die finale funktionierende Integration des LSTM-Modells im Web konnte wegen oben genannter Probleme nicht erreicht werden

---

4. Ergebnisse und Erkenntnisse

- Erfolgreiches Training und Konvertierung des LSTM-Modells mit großem Vokabular und umfangreichen Daten
- Vollständige Entwicklung einer Web-App zur Modellintegration mit Benutzerinteraktion
- Kritische Bedeutung der korrekten Modellarchitektur und Input-Definition für TensorFlow.js-Kompatibilität erkannt
- Die Arbeit verdeutlichte die komplexen Anforderungen an Modellexport und Webdeployment
- Trotz intensiver Bemühungen und Hilfestellungen konnte die Hauptfunktionalität aufgrund technischer Limitierungen nicht realisiert werden

---

5. Ausblick und Empfehlungen

- Sicherstellung einer expliziten Input-Layer-Deklaration mit `inputShape` im Keras-Modell vor dem Export
- Beginn mit kleineren Datensets zur effizienten Entwicklung und Fehlersuche
- Modularisierung von Tokenizer, Modell und Web-App für getrennte Tests
- Nutzung stabiler Softwareversionen und Dokumentation der Tool-Umgebungen
- Erweiterung der Nutzerinteraktion und Visualisierung in der Web-App bei zukünftigen Projekten

---

6. Quellen

- TensorFlow und TensorFlow.js Dokumentationen
- Literatur zu LSTM-Sprachmodellen und Next Word Prediction
- Offizielle Keras Tutorials und API-Dokumentation

GPT wurde für Texterstellung und Anpassung von Skripten verwendet