Wie funktioniert eigentlich eine LM?

Albérique und Ruben L.

June 6, 2025

1 Einführung

Wir kennen sie alle und haben sie mit Sicherheit schon einmal bentzt: LM(Language Model)s. Kommerzielle Modelle, die wir normalerweise benutzen, sind sogar LLM(*Large* Language Model)s. Sie sind weitaus leistungsstärker als kleine Modelle, sie funktionieren aber größtenteils gleich. In einem Video und einer Präsentation werden wir probieren, zu erklären, *wie*. Aber der Hauptteil unseres Projekts wird es sein, ein funktionierendes Modell zu programmieren, ohne große Libraries zu benutzen. Funkionierend heisst hier, dass wir ein Modell haben wollen, das dazu in der Lage ist, verständliche, deutsche Sätze zu bilden. Es ist gut möglich, dass wir das nicht schaffen, deshalb ist ein weniger ehrgeiziges Ziel, einfach deutsche Wörter zu bilden. Neben dem Programmieren und dem Recherchieren wird es auch wichtig sein, auf unseren nicht für das Trainieren von KIs spezialisierten Laptops dieses Modell zu trainieren und dafür eine realistische Größe zu finden.

2 Tokenizing, Dataset, Embedding Layer, Vocabulary

Der erste Schritt, den wir unternahmen, war, einen Tokenizer zu machen. Da wir nicht noch einen Algorithmus lernen wollten, der die Tokens als Wörter oder Subwörter repräsentiert, einigten wir uns auf Buchstabentokens. (Jeder Buchstabe ist also ein Token) Danach brauchten wir ein großes Dataset. Wir hatten nicht unglaublich viel Hoffnung, dass wir mit unseren bescheidenen Rechenleistungen ein sehr fähiges Modell auf die Beine bringen können, aber wir suchten dennoch ein relativ gutes Dataset. Wir holten uns ein Programm, das die Dumps in ein gigantisches txt-file umwandelt, das wir dann als Dataset nutzten. Als wir mit dem Dataset zufrieden waren (es brauchte noch ein wenig Aufräumen), machten wir uns daran, ein Vokabular für das Modell aufzustellen. Im Prinzip war es sehr simpel: Wir gingen durch alle Buchstaben im Dataset und die, die mehr als 1000-mal erscheinen (die wahrscheinlich also wahrscheinlich wichtig waren, für die Sprache, in der es geschrieben wurde) wurden zusammen mit ihrem Token in ein seperates File gepackt. Damit in Zukunft das Training schneller gehen kann, haben wir auch das gesamte Dataset, das noch aus Buchstaben bestand, in Tokens umgewandelt. Schließlich konnten wir uns der Embedding Layer widmen, unserer ersten layer. Simpel gesagt, verwandelt sie die Tokens in Vektoren, -

mehr oder weniger- den ersten Neuronen! (Sozusagen die Input Layer). Diese Vektoren wurden nicht nur gemäß ihres Tokens sondern auch ihrer Position codiert. Das heisst ein "l" in dritter Position hat nicht den gleichen Vektor wie ein "l" in vierter.

3 Das Modell

Nach der Embeddingschicht kommt das eigenliche Modell, das versucht, aus diesen Vektoren andere Vektoren zu machen, die, wenn sie wieder decodiert, für uns, nach genügend Training, verständlich sind. Es orientiert sich an der klassischen Transformerarchitektur, jedoch extrem vereinfacht und natürlich von Grund auf konstruiert (ohne Libraries, die uns groß helfen). Ein erwähnenswerteste Teil unseres Modells ist das Attention-Prinzip. Vereinfacht gesagt, kann ein Neuron mit den anderen "kommunizieren" (es wird speziell mit den anderen verrechnet, damit sie sozusagen Kontext haben). Wir haben eine extrem einfache Version davon konstruiert, doch trotzdem ist das file unser größtes.

4 Output

Damit wir das, was unser Modell da macht, auch sehen können, haben wir eine Outputschicht programmiert. Diese wandelt die Vektoren, die unser Modell am Schluss hat, in Logits um, die schon fast die Wahrscheinlichkeiten sind. Die müssen wir aber noch durch eine Softmax-Funktion jagen, um alles in den Bereich $[0, 1]$ zu bringen. Dann haben wir für jede Position einen Vektor, der angibt, wie wahrscheinlich es ist, dass ein bestimmter Buchstabe als nächstes kommt. Die Ergebnisse werden dann evaluiert (mit dem Buchstaben, der tatsächlich als nächstes kommt) und durch Backpropagation werden die Parameter aktualisiert.

5 Quellen

Ich habe keine Zeit mehr, die Quellen anzufügen, aber wir haben eine Menge. Wenn Sie den Code in seinem jetzigen Zustand sehen wollen (er spammt "o"s bei jeder Prediction, funkioniert also noch nicht, beim Zeitpunkt des

Schreibens), haben wir den Link zu unserer Github-repo hier:
www.github.com/DaCUtePotato/lm