Woche 12 Programmierung: Textverarbeitung

Skript

Erarbeitet von
Marc Feger

Lernziele

- Aufbau von BERT erklären
- BERT-Einbettungen erläutern
- Konzept der semantischen Ähnlichkeiten von Wörtern erklären

Inhalt

Einstieg

Hallo und herzlich willkommen. Im heutigen Tutorial wird es um die Textverarbeitung gehen.

Texte und Computer

Wenn wir Texte verarbeiten, denken wir selten darüber nach, wie wir sie in einer für Computer verständlichen Form darstellen können. Text-Repräsentation und Wort-Einbettungen sind jedoch entscheidende Konzepte in der natürlichen Sprachverarbeitung, die uns dabei helfen, genau das zu tun!

BERT (Bidirectional Encoder Representations from Transformers) ist ein fortschrittlicheres Modell zur Wort-Einbettung, das die Bedeutung von Wörtern anhand ihres Kontexts erfasst und sie in einem mehrdimensionalen Raum positioniert.

Quelle [1]

Das mag sich zwar kompliziert anhören, aber es ist eigentlich ziemlich cool. Denn im Gegensatz zu One-Hot-Encodings kann BERT die Bedeutungen von Worten und deren Kontext besser darstellen. Das führt zu einer höheren Genauigkeit in der Sprachverarbeitung und hilft uns, noch besser zu verstehen, was in Texten wirklich vor sich geht. Doch laden wir dafür erst einmal BERT.

```
transformer = TransformerWordEmbeddings('bert-base-uncased')
```

Wenn wir BERT geladen haben, können wir uns den Aufbau anschauen.

Eine weitere faszinierende Eigenschaft von BERT ist, dass es bereits vortrainiert ist. Das bedeutet, dass es mit einer großen Menge an Daten trainiert wurde, um eine allgemeine Sprachkompetenz zu erwerben. Das ist vergleichbar mit dem Erlernen einer Sprache durch einen Menschen, der eine Vielzahl von Texten liest und somit ein besseres Verständnis für die Sprache entwickelt. Dabei erlernt BERT ein Vokabular aus etwa 30.000 Wörtern und Subworten. Betrachten wir dazu ein Beispiel.

```
bert_vocabulary = transformer.tokenizer.vocab

print("Größe Vokabular:" , len(bert_vocabulary))
```

Wie du siehst, verfügt BERT über einen Wortschatz von ungefähr 30.000 Wörtern. Am Ende von BERT steht eine Ausgabeschicht, d. h. wir bekommen eine Repräsentation der Wörter zurück. Schauen wir uns an, wie diese Ausgaben aufgebaut sind.

```
print("Größe Einbettung:", transformer.model.embeddings.word_embeddings)
```

Wie du hier sehen kannst, erzeugt BERT aus seinen ungefähr 30.000 Wörtern im Wortschatz Einbettungen für einzelne Wörter der Größe von 768. Das heißt, am Ende von BERT steht ein Vektor der Größe 768, wobei jeder Eintrag in diesem Vektor eine andere Dimension des

Textes kodiert. Ebenso können wir uns ansehen, auf wie viele Schichten von Encodern BERT dafür zurückgreift.

```
transformer_layers = transformer.model.encoder.layer
print("Transformer Schichten:", len(transformer_layers))
```

Wie du hier siehst, besteht BERT aus 12 Encoder-Schichten. Jeder dieser Encoder sorgt dafür, dass die Informationen von Schicht zu Schicht über das Wort genauer werden.

Ein Beispiel für Textverarbeitung

Kommen wir nun zu einem Beispiel für Textverarbeitung. Ein klassisches Beispiel um das Konzept der Wort-Einbettungen und die Vorteile von BERT zu demonstrieren ist das Rätsel: “king - man = x - woman”. Dabei werden die Vektoren der Wörter *king, man* und *woman* sinnbildlich in Bezug zueinander gestellt, um das Wort *x* zu erhalten. Der Vorteil in der Verwendung von BERT-Einbettungen besteht darin, dass sie auf einem starken Sprachmodell basieren und so in der Lage sind, anspruchsvollere und nuanciertere Beziehungen (also die Semantik) zwischen Wörtern zu erfassen.

In unserem Beispiel beschränken wir uns auf einen Wortschatz von 80 Wörtern, die in die vier Kategorien Menschen, Zahlen, Fahrzeuge und Nationen eingeteilt werden.

```
royal = "royal monarchy crown queen king coronation throne palace majestic
imperial"
sex = "sex man woman diversity transgender masculinity femininity intersex
LGBTQ gender"

digits = "1 2 3 4 5 6 7 8 9 10"
numbers = "one two three four five six seven eight nine ten"

car = "car wheels engine speed road drive auto fuel vehicle emission"
aircraft = "aircraft airbus boeing B747 A380 pilot cockpit wings takeoff
landing"

cities = "Beijing Tokyo London Paris Berlin Rome Madrid Warsaw Bangkok
Bern"
nations = "China Japan UK France Germany Italy Spain Poland Thailand
Switzerland"

corpus = " ".join([royal, sex, digits, numbers, car, aircraft, cities,
nations])

print("Text of all words:", corpus)
```

Um vom Wort zum Vektor zu kommen, greifen wir auf *Flair* zurück. Das Sentence-Objekt in Flair hilft dabei, Texte und Sätze als Vektoren darzustellen. Das ist hilfreich, um Texte zu analysieren und verschiedene Aufgaben in der Sprachverarbeitung zu lösen. Dafür starten wir mit dem Sentence-Objekt.

```
sentence = Sentence(corpus)
```

Um Vektoren mit dem Sentence-Objekt zu erstellen, wird einfach die *embed()* Methode aufgerufen. Dabei werden vorab trainierte BERT-Embedding-Vektoren verwendet, die dann für die jeweiligen Worte zusammengesetzt werden. Das ist ähnlich wie der Aufruf von *fit_transform()* in der *sklearn* Bibliothek, wo ein vorab trainiertes Modell die gegebenen Daten in eine andere Darstellung transformiert.

```
transformer.embed(sentence)
```

Wie du siehst, haben wir BERT darum gebeten, den Satz einzubetten. Um nun auf die jeweiligen Embeddings und Wörter zuzugreifen, erstellen wir jeweils zwei separate Listen.

```
words = list()
embeddings = list()
```

Das Sentence-Objekt besteht aus einzelnen Token. Jeder Token repräsentiert ein Wort und jedes Wort wurde durch *embed* an ein entsprechendes Embedding gehangen. Wir können über den Satz iterieren und jedes Wort und jedes Embedding herausnehmen.

```
for token in sentence:
    word = token.text
    embedding = token.embedding.numpy()
    words.append(word)
    embeddings.append(embedding)

print("Beispiel Wort:", words[0])
print("Einbettung Wort:", embeddings[0][:4])
```

Wie du siehst, iterieren wir über jeden Token im Satz. Von jedem Token nehmen wir den Text und das entsprechende Embedding. Beide werden separat in die jeweilige Liste eingefügt. Wie du an dem Beispiel Royal nachvollziehen kannst, erhält das Wort eine numerische Repräsentation. Mit dieser Repräsentation können wir arbeiten. In unserem Beispiel wurde sich auf vier Einträge des 768 großen Vektors beschränkt.

Semantische Ähnlichkeit von Wörtern

Da du nun gesehen hast, wie Wörter in Vektoren überführt werden, welche deren Bedeutung kodieren, haben wir eine Textrepräsentation, mit der man tatsächlich rechnen kann.

Semantische Ähnlichkeit ist eine Methode zur Messung der Ähnlichkeit zwischen Bedeutungen von Wörtern. Es hilft uns, Anwendungen der Textverarbeitung besser zu

machen, indem es uns ermöglicht, die Bedeutung von Wörtern zu verstehen und darzustellen.

Daher können wir dieses Konzept verwenden, um die Gleichung "king - man = x - woman" mit BERT-Einbettungen zu lösen, indem wir die Ähnlichkeiten zwischen den Wörtern in der Gleichung vergleichen.

In dieser Gleichung geht es darum, das Wort "x" zu finden, das dem Wort "king" am ähnlichsten ist, während es dem Wort "man" am unähnlichsten ist. In gleicher Weise sollte das Wort "woman" dem Wort "man" am ähnlichsten sein.

Um das Wort "x" zu finden, können wir schlichtweg die Kosinus-Ähnlichkeit zwischen den BERT-Einbettungen der einzelnen Wortpaare in der Gleichung berechnen.
Diese ist für eine solche Aufgabe gut geeignet, da sie den Kosinus des Winkels zwischen zwei Vektoren in einem hochdimensionalen Raum misst, wobei jeder Vektor die Einbettung eines Wortes darstellt.

Die Kosinus-Ähnlichkeit reicht von 0 bis 1, wobei 1 bedeutet, dass zwei Wörter genau dieselbe Bedeutung haben, und 0 bedeutet, dass die beiden Wörter völlig unterschiedliche Bedeutungen haben.

Betrachten wir dazu ein Beispiel von drei Wortpaaren.

```
print_word_similarity("king", "queen", words, embeddings)
print_word_similarity("man", "queen", words, embeddings)
print_word_similarity("woman", "queen", words, embeddings)
```

Wie du siehst, ist die Ähnlichkeit zwischen dem Wortpaar *king* und *queen* höher als die vom Rest. Ebenso siehst du, dass das Wortpaar *woman* und *queen* eine leicht höhere Ähnlichkeit hat als *man* und *queen*. Das liegt daran, dass die Worte *woman* und *queen* kontextuell gesehen näher miteinander verwandt sind als *man* und *queen*.

Da du nun gesehen hast, dass wir mit den Vektoren ein grundlegendes Verständnis für den Zusammenhang von Wörtern über deren Ähnlichkeit erlangen können, kann auch das Rätsel gelöst werden. Dafür wird das Rätsel nach *x* umgestellt und nach der ähnlichsten Einbettung für das Wort *x* gesucht. Betrachten wir dazu die Lösung unseres Rätsels und einige weitere Beispiele.

```
get_word_suggestion("king", "man", "woman", words, embeddings)
get_word_suggestion("3", "1", "2", words, embeddings)
get_word_suggestion("three", "one", "two", words, embeddings)
get_word_suggestion("aircraft", "wings", "wheels", words, embeddings)
get_word_suggestion("Germany", "Berlin", "Bangkok", words, embeddings)
```

Wie du hier siehst, ist das gesuchte Wort *queen*. Weiterhin kannst du auch sehen, dass wir Rechnungen durchführen können, zum Beispiel 3-1 = 4-2. Das Gleiche gilt auch für die ausgeschriebenen Versionen der Zahlen. Weiter können wir aber auch andere Vergleiche

machen, z. B. ein Flugzeug ohne Flügel ist wie ein Fahrzeug ohne Reifen. Anders können wir aber auch geographische Vergleiche machen, beispielsweise Deutschland ohne Berlin ist wie Thailand ohne Bangkok.

Du siehst, es ist tatsächlich möglich, mit BERT-Einbettungen zu arbeiten und dass diese die Bedeutung von Wörtern erfassen. Werfen wir dazu einen finalen Blick in das Gedächtnis von BERT.

```
visualize_word_embeddings(words, embeddings)
```

Wie du sehen kannst, gruppieren sich die Worte entsprechend ihrer Kategorien in BERT‘s Gedächtnis. Wie du an den Zahlen erkennen kannst, gilt sowohl für die numerische als auch für die ausgeschriebene Darstellung, dass beide jeweils ein separates Cluster bilden. Ebenso liegen diese trotzdem noch getrennt von den anderen Kategorien, da beide Zahlen meinen. Weiterhin siehst du, dass sich einzelne Cluster für Fahrzeuge, Menschen und Städte bilden.

Abschluss

Ich hoffe, das heutige Tutorial hat dir gefallen und du hast einen Einblick darin bekommen, welche Möglichkeiten dir mit BERT zur Verfügung stehen. Bis dahin.

Quellen

Quelle [1] Kenton, J. D. M. W. C., & Toutanova, L. K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT* (pp. 4171-4186).

Weiterführendes Material

https://huggingface.co/course/chapter1/4?fw=pt
https://huggingface.co/blog/bert-101

Disclaimer