FOFE-Encodings als character-based Word-Embeddings für POS-Tagging im Deutschen

Inhalt

1 Vorstellung

Diese schriftliche Ausarbeitung bezieht sich auf die Implementierung eines neuronalen Part-of-Speech-Taggers mit Hilfe von character-based FOFE-Embeddings. Zunächst wird dabei auf das Konzept des FOFE-Encodings eingegangen, sowie seine Relevanz für Word-Embeddings herausgearbeitet. Anschließend werden der verwendete TIGER Datensatz und dessen Vorverarbeitungs-Schritte dargelegt. Darauf folgt eine Skizzierung der Python-Implementation, welche die eigens erstellten Klassen, die Architektur und den Ablauf des Programmes näher beschreibt. Im Zuge dessen wird auch die Auswahl der optimalen Hyperparameter erläutert, sowie die erzielten Ergebnisse vorgestellt. Die Ausarbeitung schließt mit einer Zusammenfassung der gewonnen Erkenntnisse sowie mit einem kurzen Ausblick.

2 FOFE-Methode

Die Abkürzung FOFE steht für *fixed-size ordinally-forgetting encoding* und beschreibt einen von Zhang et al. (2015) entwickelten Ansatz, Wort-Sequenzen von variabler Länge in Repräsentation mit konstanter Länge zu überführen. Besonders daran ist, dass diese Satz-Enkodierungen unter bestimmten Voraussetzungen eindeutig sind, nämlich genau dann, wenn der festzulegende FOFE-Faktor im Bereich]0; 0,5] liegt (Zhang et al., 2015). Im Zuge dieses Projektes wird die FOFE-Methode adaptiert, um einzelne Wörter zu enkodieren. Dabei wird für jeden Buchstaben ein One-hot-Vektor erstellt, dessen Länge der Größe des gewählten Alphabetes entspricht. Dieser Vektor wird mit dem Faktor α^{i-k} multipliziert, wobei α den konstanten FOFE-Faktor, i einen konstanten Wert (nach Zhang et al.: i=0) und k die aktuelle Character-Position im Wort darstellt. Diese Vektoren werden für alle Zeichen des Wortes berechnet und elementweise aufsummiert.

Eine inhärente Eigenschaft des FOFE-Encodings besteht darin, dass die Gewichtungen der Buchstaben im Wortvektor von Anfang bis Ende des Wortes abnehmen. Durch die Konkatenation mit einem weiteren FOFE-Vektor, welcher das Wort in umgekehrter Reihenfolge enkodiert, wird ein bidirektionales Encoding erreicht. Mit Blick auf das POS-Tagging birgt dies den Vorteil, dass bidirektionale FOFE-Embeddings gerade die Prä- und Suffixe von Wörtern hervorheben, während diese beiden Wortteile oftmals besonders ausschlaggebend für die zugrundeliegende Wortart sind. Diese morphologische Eigenschaft wird an den beiden Beispiel-Partizipien ***ver**schlüssel**t*** und ***ver**rechne**t*** deutlich.

3 Datensatz

Als Datensatz für die Entwicklung und Evaluierung des POS-Taggers wurde der deutsche TIGER Korpus 2.2 im CONLL09-Format herangezogen. Dieser besteht aus 50472 morphologisch annotierten Sätzen, wobei nur die Tokens sowie die zugehörigen POS-Tags entnommen wurden. Die Unterteilung des Datensatzes geschah folgendermaßen: 4000 Test-Sätze, 4000 Validierungs-Sätze und 42472 Trainings-Sätze.

4 Programmaufbau

Die Vorverarbeitung der annotierten Daten basiert auf 2 Klassen: Tiger_Data und FOFE_Encoding. Erstere dient dazu, den Datensatz einzulesen und diesen wie in Kapitel 3 beschrieben aufzuteilen. Die Klasse FOFE_Encoding liest wiederum ein Tiger_Data-Objekt ein und errechnet daraus die Embedding Matrix, welche die Größe num_words x 2*num_chars besitzt. Das Vokabular umfasst jene Wörter des Datensatzes, die mindestens zweimal vorkommen. Alle anderen Wörter werden als unbekannte Wörter mit einheitlichem Embedding repräsentiert und somit ebenfalls mittrainiert. Die Wortvektoren mit der Länge 2*num_chars stellen eine Konkatenation des Forward- und Backward-FOFE-Embeddings dar. Dabei ist in diesen Embeddings für jedes im Datensatz vorkommendes Zeichen ein Platz reserviert, inklusive einem Platz für unbekannte Zeichen (Position 0).

Das Tagger-Model wurde mithilfe der Deep-Learning-Bibliothek Keras erstellt und basiert auf einem sequentiellen, neuronalen Netzwerk. Dieses setzt sich wiederum aus einer Embedding Layer, einer bidirektionalen LSTM-Layer sowie einer Dense-Layer mit Softmax-Aktivierungsfunktion zusammen. Die Embedding-Layer übernimmt die zuvor erstellte FOFE-Embedding-Matrix als Gewichtsmatrix, jedoch ohne dass deren Gewichte im Training mittrainiert werden.

5 Hyperparameter-Tuning

Das Ermitteln der optimalen Hyperparameter geschah, indem das Netzwerk auf 20 % der Trainingsdaten mit 15 Epochen trainiert wurde und dabei die Accuracy und Loss-Werte sowohl auf den Test- als auch den Validierungs-Daten ermittelt wurde. Mit Blick auf die Hidden-Layer erwies sich ein bidirektionales LSTM mit 256 Hidden Nodes als die beste Lösung. Hierbei führte eine L2-Regularisierung von 0.001 sowohl auf dem Kernel als auch auf dem Bias zu einer Steigerung der Accuracy. Um Overfitting zu vermeiden, wurde Dropout nach der Embedding Layer sowie Recurrent-Dropout im LSTM eingesetzt, wobei eine Rate von jeweils 0.2 zu den besten Ergebnissen führte. Als Optimizer diente der RMS-Prop inklusive seiner Default-Parameter, welcher im Vergleich zum Adam-Optimizer zu geringfügig besserer Accuracy verhalf. Als weitere optimale Parameter zeigten sich eine Batchsize von 32, sowie eine maximale Epochenanzahl von 60 inklusive Early-Stopping, welches nach 2 Epochen ohne Verringerung des Validation-Losses das Training abbricht.

Der optimale FOFE-Faktor für diesen Task liegt bei einem ermittelten Wert von 0.5, allerdings wurde hierbei der initiale Exponent nicht wie bei Zhang et al. (2015) auf 0 festgelegt. Im Zuge des Hyperparameter-Tunings wurde festgestellt, dass eine Verringerung des initialen Exponenten zu einer merklich höheren Accuracy führt. Der optimale Anfangs-Exponent liegt demnach bei -5. Grund hierfür ist vermutlich die Tatsache, dass bei einem initialen Exponenten von 0 die Buchstabengewichtung bei längeren Wörtern schnell gegen 0 konvergiert. Dem wird durch einen negativen Start-Exponenten etwas entgegengewirkt.

6 Programmaufruf

Das Python-Skript ‚train.py' führt bei Aufruf das Training des POS-Taggers durch. Hierbei können die gewünschten Hyperparameter des Netzwerkes als fakultative Argumente übergeben werden. Geschieht dies nicht, werden die optimalen Parameter, wie in Kapitel 5 genannt, verwendet. Mit der Datei ‚tagger.py' kann der zuvor trainierte Tagger zur Wortart-Klassifikation eingesetzt werden. Hierzu werden die Sätze in einer Text-Datei (ein Satz pro Zeile) eingelesen und zusammen mit ihren errechneten Tags wieder über *Standard Output* ausgegeben. Der Aufruf des Skripts kann ohne Argumente erfolgen, wobei in diesem Fall Default-Namen für die Text-Datei, das Tagger-Model und die FOFE-Parameter-Datei herangezogen werden. Alternativ können die besagten Namen auch als Argumente übergeben werden.

7 Ergebnisse

Der POS-Tagger erzielte eine Accuracy von 0.95 auf den Validierungs-Daten und 0.96 auf den Test-Daten, wobei diese Werte nach 13 Epochen (Early-Stopping) erreicht wurden. Um den Einfluss der FOFE-Methode auf die Performanz des Taggers zu testen, wurde zum Vergleich zusätzlich ein Training mithilfe von ungewichteten Bag-of-Chars-Embeddings durchgeführt. Die Ergebnisse sind in Tabelle 1 dargestellt.

	Bag-of-Chars	***FOFE***
Validation-Accuracy	0.911	**0.950**
Test-Accuracy	0.913	**0.957**

Tabelle 1: Accuracies von Bag-of-Chars- und FOFE-Ansatz

8 Résumé und Ausblick

In diesem Projekt wurde die Methode des FOFE-Encodings auf buchstabenbasierte Word-Embeddings angewandt. Diese dienten wiederum als Input für einen Part-of-Speech-Tagger, welcher auf einem Long Short-Term Memory Neural Network basiert. Die Ergebnisse zeigen, insbesondere im Vergleich mit einem Bag-of-Chars-Ansatz, dass die spezielle Character-Gewichtung des FOFE-Encodings einen positiven Effekt auf die Genauigkeit des Taggers hat. Dies liegt vermutlich an der stärkeren Gewichtung von Infixen und Suffixen, welche meist viel morphologische Information beinhalten. Darüber hinaus konnte durch die Anpassung des initialen Exponenten von 0 auf -5 eine weitere Steigerung der Accuracy erzielt werden. Im Rahmen einer weiterführenden Forschung könnte das hier vorgestellte, bidirektionale FOFE-Embedding zum Training eines POS-Morph-Taggers verwendet werden.

9 Quellen

Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, Lirong Dai (2015): Fixed-Size Ordinally-Forgetting Encoding Method for Neural Network Language Models. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers)*, pages 495–500, Beijing, China, July 26-31.

Institut für Maschinelle Sprachverarbeitung, Uni Stuttgart: TIGER Corpus Release 2.2 (July 2012), URL: http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/download/start.html, Zugriff am 10.01.2019.