Goethe Universität
Institut für Informatik
Texttechnologie

SoSe 2024

NLP-gestützte Data Science

Übung 1

Manuel Stoeckel Prof. Dr. Alexander Mehler

Frist: 06. Juni 2024

Übung 1: Word2Vec **50 P**

In dieser Übung werden wir uns mit dem `word2vec`-Modell (Mikolov, Chen et al., 2013; Mikolov, Sutskever et al., 2013) beschäftigen, welches in `PyTorch` zu implementieren ist. Lesen Sie insb. das Paper *Distributed Representations of Words and Phrases and their Compositionality*, Mikolov, Sutskever et al. (2013)!

Definitionen

Das `word2vec` Skip-Gram-Modell für *SoftMax* und *Negative Sampling* (NS) sind gegeben als:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-1\leq j\leq c,\ j\neq 0} F_{\text{obj}}(w_{t+j}, w_t) \tag{1}$$

$$F_{SoftMax}(w_O, w_I) = \log \frac{\exp\left(v'^{T}_{w_O} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left(v'^{T}_{w} v_{w_I}\right)} \tag{2}$$

$$F_{NS}(w_O, w_I) = \log \sigma\left(v'^{T}_{w_O} v_{w_I}\right) + \sum_{i=1}^{k} \log \sigma\left(v'^{T}_{w_i} v_{w_I}\right) \tag{3}$$

Hier sind v und v' jeweils die Input- und Output-Embeddings, (2) bzw. (3) sind die *objective functions* der *SoftMax* bzw. *Negative Sampling* Varianten. Die *negative samples* w_i in (3) werden zufällig aus der *unigram distribution* $w_i \sim P_n(w)$ gezogen, wobei $P(w_i)$ für ein bestimmtes Wort w_i mit Frequenz im Trainingskorpus $f(w_i)$ gegeben ist als:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}} \tag{4}$$

Kontextwörter w_{t+j} werden dabei im Rahmen des *Subsampling of Frequent Words* (Mikolov, Sutskever et al., 2013, §2.3) mit Wahrscheinlichkeit $S(w_i)$ **nicht** aus einem Kontext entfernt, wobei $S(w_i)$ abweichend vom Paper wie folgt definiert ist:

$$S(w_i) = \left(\sqrt{\frac{f(w_i)}{t}}\right) \cdot \frac{t}{f(w_i)} \tag{5}$$

Dabei ist t ein Hyper-Parameter, üblicherweise 0.001, und $f(w_i)$ die Frequenz wie zuvor.

1.1 Text Processing **20 P**

- Implementieren Sie die Klassen:
  - **`TokenizedSentence`**, **`PreTokenizer`** und **`Tokenizer`**,
  - das **`Dataset`**, in welchem die Samples für das aggregiert werden, sowie
  - den **`NegativeSampler`** und die `Dataset`-Variante **`DatasetNegativeSampling`**.

Hinweise

- Im Gegensatz zu einer Mindestanzahl an Vorkommen im Trainings-Korpus für den Tokenizer, wie im originalen `word2vec`, sollen Sie einen Tokenizer mit einer maximalen Größe implementieren. Dabei werden die häufigsten Token gespeichert und weniger häufige Token verworfen.
- Der **`PreTokenizer`** soll in der Vorverarbeitung **alle Satzzeichen**[1] von anderen Wörtern **einzeln** abtrennen.
- Achten Sie darauf auch das Subsampling von häufigen Wörtern zu implementieren.

1.2 Word2Vec **25 P**

- Implementieren Sie die Klassen **`SkipGramSoftMax`** und **`SkipGramNegativeSampling`**.

1.3 Ergebnisse **5 P**

Verwenden Sie Ihren Code und dokumentieren Sie Ihre Ergebnisse (in einer begleitenden PDF).

- Trainieren Sie einen Tokenizer an dem kleinen Beispiel-Korpus (`data/train_enwiki.txt.gz`). Was sind die häufigsten Token?
- Trainieren Sie die Modelle auf dem tokenisierten Korpus. Können Sie die Ergebnisse von Mikolov et al. reproduzieren?
  - Verwenden Sie Standard-Hyperparameter zum Training. Sie können die GPUs auf den Rechnern der RBI nutzen, um Ihr Training zu beschleunigen.
  - Falls das Training zu lange dauert, können Sie es auch frühzeitig unterbrechen.
  - Dokumentieren Sie auch Ihre Zwischenergebnisse.
- Wählen Sie sinnvolle Hyperparameter für den Tokenizer und Ihre `word2vec` Modelle beim Training!

Extra-Aufgaben

CBOW **5 P**

In Aufgabe 1.2 war nur das Skip-Gram-Modell zu implementieren. Das CBOW-Modell wird im ersten `word2vec` Papier ausführlicher erläutert (Mikolov, Chen et al., 2013). Im Prinzip unterscheiden sich die Modelle aber nur dahingehend, dass beim Skip-Gram-Modell ein *input* Zielwort mit den Embeddings mehrerer *output* Kontextwörter verglichen wird, während beim CBOW-Modell der Durchschnitt der *input* Kontextwörter mit dem *output* Embedding des Zielworts verglichen wird.

- Implementieren Sie nun die CBOW-Modelle **`CbowSoftMax`** und **`CbowNegativeSampling`**.

[1] Siehe: https://docs.python.org/3/library/string.html

Literatur

Mikolov, Tomás, Kai Chen et al. (2013). „Efficient Estimation of Word Representations in Vector Space“. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Hrsg. von Yoshua Bengio und Yann LeCun. arXiv: 1301.3781.

Mikolov, Tomás, Ilya Sutskever et al. (2013). „Distributed Representations of Words and Phrases and their Compositionality“. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Hrsg. von Christopher J. C. Burges et al., S. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.