Prof. Dr. Andreas Hotho, Albin Zehe, Jonas Kaiser
Data Science Chair

02.05.2024

1. Assignment in “Machine Learning for Natural Language Processing”

Summer Term 2024

1 General Questions

1. Identify three concrete cases where Deep Learning is used for language processing in real-world applications!

    - Google Translate, DeepL
    - Predictive Keyboards (Smartphones)
    - Voice Recognition
    - Grammarly
    - GitHub Copilot
    - ChatGPT

2. Name two reasons why word embeddings are better suited for most NLP tasks than 1-hot encodings!

    - word embeddings encode word similarity, 1-hot encodings do not
    - the word itself is usually less important than its meaning and its relation to other words

3. Without covering the details of how the training is actually performed, explain the intuition behind Word2Vec!

    The Idea behind Word2Vec is to capture the meaning and semantic relationships between words based on their co-occurrence patterns in a large corpus of text. The core idea is that words that appear together frequently in similar contexts should have similar vector representations.

For example, if we take the sentence "My pet cat is sleeping", the context words "My", "pet", "is", and "sleeping" can help in predicting the target word "cat". Similarly, if we have a good representation of the word "cat", it would be easier to predict the surrounding context words.

Intuitively, to perform these predictions, words that fulfill similar roles within a language should be represented via similar vectors. In the example above most other pet animals would be reasonable alternatives, i.e. "dog" should have a representation similar to "cat". At the same time verbs other than "sleeping", that could reasonably be used in this context should have a similiar vector to it.

Across enough examples, and by training our Neural Network via gradient descent, Word2Vec uses this idea to learn a dense vector space where semantically similar words are mapped to nearby points. The goal is to create a mapping that preserves the semantic relationships between words, such as synonymy, antonymy, hyponymy, etc.

2 Bag of Words and Linear Classifiers

Suppose we have the input sentences “I really like cats” and “I mostly like dogs” and want to classify them into the classes “is about cats” and “is about dogs”. Let’s create a linear classifier for this!

1. Pretend that only the words from the sentences above are in the vocabulary. Create a bag of words representation for each sentence. For a better overview, sort the vocabulary alphabetically.
2. Create a linear classifier that correctly classifies the sentences into the classes “is about cats” and “is about dogs”. How many inputs does the classifier need? How many outputs? Assign weights by hand, since optimization was not covered in the lecture yet.
3. Given only the sentences above for selecting the weights of the linear classifier; are there multiple solutions to correctly classify both sentences? Can you construct a valid input sentence that leads to an incorrect classification for an alternative solution? What could we do to prevent such wrong solutions?

1. The vocabulary, sorted alphabetically, is: ‘cats’, ‘dogs’, ‘I’, ‘like’, ‘mostly’, ‘really’. Thus, the bag of words representation for the first sentence is [1, 0, 1, 1, 0, 1] and for the second sentence is [0, 1, 1, 1, 1, 0].

2. The linear classifier has six inputs and either one or two outputs. Since this task is a binary classification task, we can output either one number indicating the first class (< 0) or the second class (≥ 0). Alternatively, we can output two numbers indicating both classes. We go with two outputs, since this enables us to use more than two classes when necessary. We can represent the weights as a matrix with six rows and two columns, which can be multiplied by the sentence representations. The first output dimension is then indicating the class "is about cats" and the second output dimension is indicating the class "is about dogs". The output class is determined by the largest output dimension. One possible solution for the weights of the linear classifier is

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

i.e. checking if the word "cats" is in the bag of words representation of the first sentence and the word "dogs" is in the bag of words representation of the second sentence. We do not need a bias term in this case.

3. In general, we can scale the weight matrix by any non-negative number. This does not change the maximum output dimension. Another solution to this task with the given inputs is

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix},$$

i.e. paying attention to the words "really" and "mostly", since these words are different in both sentences. This solution also provides correct outputs for the example sentences, but will not generalize well to new sentences about cats and dogs. For example, the sentence "I really like dogs" will be classified as "is about cats" with this solution, since the word "really" occurs in the sentence. To prevent this, we should collect more “training sentences” to create the classifier weights. With larger datasets, words are used in different contexts, thus the best solution should pay attention to more useful words. This is what we hope when optimizing the classifier weights using a training dataset.

3 Python

3.1 Installing and Testing Python and Numpy

- Install Python on your computer. The most recent version can be found at `https://www.python.org/downloads/`.
- Install Numpy on your computer. You can install it using your command line: `pip3 install numpy`. The documentation can be found at `https://numpy.org`.
- Create a Python file, insert the matrices and vectors from the previous task in Python and compute the outputs of the linear classifier to get a feeling for the syntax of Python and Numpy. If you want, you can also implement a function that creates the bag of words representation for a sentence. Feel free to experiment with the language and tools, since you will need them for the upcoming programming tasks.

A sample solution has been uploaded as a jupyter notebook.