COMP 642 — Machine Learning Feb 15, 2022

Lecture 12: Data Types I: Sets, Text, n-grams, NLP

Lecturer: Anshumali Shrivastava *Scribe By: Maojie Tang, Kathy Wang, Soham Pajwani*

Disclaimer: *These lecture notes are intended to develop the thought process and intuition in machine learning. The materials are not thoroughly reviewed and can contain errors.*

1 Background of Word and Categorical Representation

In the discipline of Natural Language Processing (NLP), the first question we must address is word representation [1]. Different natural languages have been established, however they are all discrete sets, making it difficult to educate computers directly using these words. It is for this reason that "word representation" is essential. But how can we create a mapping or transformation mechanism that will make it easier for computers to learn languages? We may use vector to represent each element in this collection because each language is a discrete set. In addition, the input from other machine learning models can provide us with intuition. Then, for word representation, people come up with two primary techniques.

1.1 One-Hot Representation Introduction

One-hot encoding [2] provides people with intuition first. For each word or categorical value, they apply one-hot encoding. For example, if the word space contains only three words, such as cat, dog, and pig, the one-hot representation can be

$$cat = [1, 0, 0], dog = [0, 1, 0], pig = [0, 0, 1] \tag{1}$$

Obviously, representing a word or category is very efficient and simple via one-hot method, but there are three major drawbacks.

(1)High dimension

We used a three-dimensional vector to represent three words in the previous example. However, if we need to represent millions of words, the dimension of each word vector can be quite big. It is difficult to execute work in such a high dimension due to the curse of dimension.

(2) Discrete for each element

Because of one-hot encoding, a word vector's elements can only be zero or one. As a result, it is unable to convey sufficient information for the some NLP tasks.

(3) Local representation property

Because of the one-hot encoding property, only one element in each word vector can be one, while the rest are all zero, implying that one non-zero element is responsible for all of the word's information. Furthermore, when the second and third limitations are combined, a new disadvantage emerges: it is difficult to determine the degree of similarity between words. Because word vectors generated by one-hot encoding are orthonormal to each other, the degree of similarity between two different vectors is zero.

1.2 Distributed Representation Introduction

Given the shortcomings of One-hot representation, we should apply some constraints to obtain the transformation function. To begin, these transformations should be injective functions, meaning that each

element of the transformation's codomain is the image of at most one element of the domain. Second, they should be structure-preserving, which means that if two words are close in their original space, they should be close in the new space as well[3]. As a result, a distributed representation approach should have the following characteristics: (1)Relatively low dimension Distributed representation models should provide word vectors with less dimensions than one-hot representation. As a result, we don't need a lot of RAM to use these word vectors. (2)Each element is a real number In distributed representation models, each element can be any real number instead of zero or one, which means they have more combination methods to express different words. (3)Global representation property Because the word vector has more than one non-zero element, each one can contribute to the expression of word information.

2 One-Hot Representation

2.1 One-Hot encoding representation

The one-hot encoding method can be used to represent categorical values that cannot be reasonably inputted into the model as numerical or text. Consider the case of zipcodes below.

$$[77005, 77006, 44056] = [1, 0, 0] \tag{2}$$

Where for one sample from the dataset, its zip code is 77005. We would not want to represent the zip codes as a numerical value, because then 77006 would be considered similar to 77005 by numerical distance. This would make two different data points (one with 77005, the other 77006) more likely to have the same assigned label, when that would not make sense for these two entirely different geographical locations. We want to consider the zip codes as entirely separate cases, thus we classify them as categories. As noted in section 1.1, the same method can be used for text values.

2.2 Represent sentences via One-Hot representation

Sentences can also be "tokenized" and broken down by each word, or subset of phrases. We go into detail for different cases in these sections.

2.2.1 Bag-of-Words representation

We can create numerical vectors for each sentence or document and use a dictionary, or "Bag-of-Words" with multiple one-hot encodings combined. Each sentence can be represented by a vector, and each value in this vector is a one-hot encoding for the corresponding word in the Bag-of-Words dictionary. For instance, consider the example in Figure 1 [4]. The dictionary contains all of the words the documents have, with each word in a set index. Each word in each document has a one-hot encoding

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

Figure 1: Bag-of-Words Encoding, with Frequency

frequency in the same indexes as the dictionary, indicating if the document has the observed word and how many times it occurred. This Bag-of-Words can further be used in a method called Term Frequency-Inverse Document Frequency (TF-IDF), which is based on Zipf's Law or Power Law.

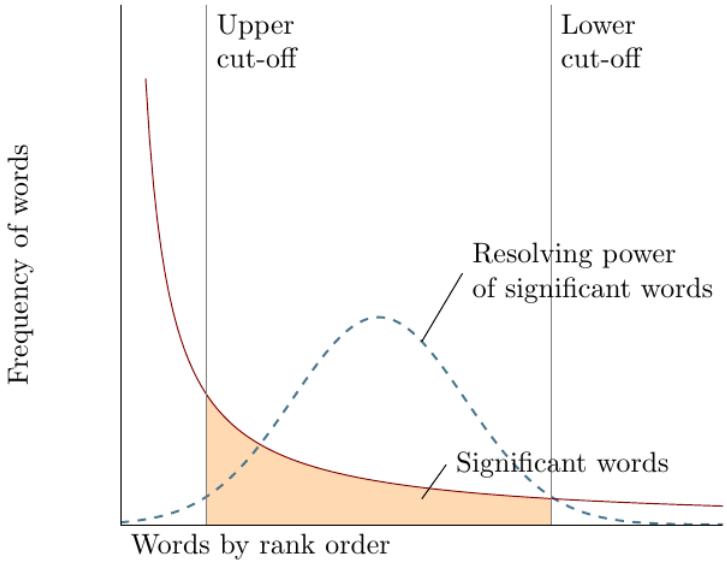

Figure 2: Power Law and Word Frequency Distribution

Zipf's law asserts that the frequencies f of certain events are inversely proportional to their rank r. The idea is that words that are very commonly used in language (ex. the, a , I) are not useful or important for use in predicting labels for documents. As seen in Figure 2, the most significant words are towards the middle of the distribution. We take this idea and then calculate the TF-IDF of each document.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Figure 3: TF-IDF Equation

The Term Frequency (TF) measures how frequently the word occurs in any one document. The Inverse-Document-Frequency (IDF) up-weights the rare words across all documents, and is calculated by taking the log of number of documents in the corpus divided by the number of documents in which the word appears [7]. For example, if a word commonly appears across all documents (such as "the") then it will have a IDF value of close to 0 because it is not going to help with separating the documents into labels. The IDF value of each word is then multiplied by the TF for each document, and then the TF-IDF value of that word for that specific document is obtained.

For the purposes of machine learning, each document in a corpus can be represented by a vector and

the values in those vectors can be the TF-IDF score for every word across all documents. These TF-IDF scores can be obtained from getting the Bag-Of-Word encoding vectors of each document and doing the calculations on them from Figure 3. If we wanted to calculate similarity between documents, we could take the cosine similarity of the TF-IDF vectors and use this to categorize the documents into similar topics.

2.2.2 Bag-of-Phrases and n-gram

When we use one hot encoding, we might miss out on important information.
For example, consider:
D1 - Rice University
D2 - Stanford University
D3 - Rice Bowl

One hot encoding will consider D1 and D2 as similar as D1 and D3, but we know they are completely different. Hence, n-gram is brought into picture.

Instead of words, we have a sequence of words represented in a set.

Assume a document: This is Rice University in Houston
The uni gram representation: {This, is, Rice, University, in, Houston}
2-gram representation: {This is, is Rice, Rice University, University in, in Houston}
So on and so forth for n-gram.

2.2.3 Character n-gram

Instead of using token for words, each character is a token by itself.

For example, when Googling "iphone", the user types "ipone" instead. If we consider each word to be a token, the typing mistake cannot be recognized.
Instead, of n-gram we can use character n-gram.

Character 3-gram representation.
iphone: {iph, pho, hon, **one**}
ipone: {ipo, pon, **one**}
Now there is a similarity of "one" between the two sets.

These sets can be treated as vectors and used in one-hot encoding like in the method seen in Section 2.2.1.

3 Distributed Representation Methods

3.1 Word2Vec

We already have a corpus for word2vec, however it just consists of a few sentences with no labels. From these sentences, we need to extract information (word vector). Assume this corpus is $V = (s_1, s_2, s_3, s_{|V|})$, and there are several words $W = (w_1, w_2, ..., w_T)$ in these sentences, where $|V|$ signifies the corpus size and T specifies the number of different words in the corpus. From a probabilistic

standpoint, we can consider each element in W to be a random variable and use maximum likelihood to obtain the best estimated parameters for each w_i. So, we should check the jointly probability:

$$P(w_{1:T}) = P(w_t)P(context(w_t)|w_t) \tag{3}$$

The equation implies that each word should be linked to its context. However, if the context is lengthy, the calculation of the jointly probability should be also lengthy. As a result, we make use of the first assumption.

Assumption I: only inside a certain range $2C$ (window size)., The context can play a role in determining the likelihood.

So, we can get

$$P(w_{1:T}) = P(w_t)P(w_{t-C:t-1}, w_{t+1:t+C}|w_t) \tag{4}$$

In addition, we can get the (conditional) likelihood $P(w_t)P(w_{t-C:t-1}, w_{t+1:t+C}|w_t)$, because $P(w_t)$ is a constant in this equation, the likelihood can be simplified as $P(w_{t-C:t-1}, w_{t+1:t+C}|w_t)$. Since the length of W is T, the jointly condition likelihood should be

$$\mathscr{L} = \prod_{t=1}^{T} P(w_{t-C:t-1}, w_{t+1:t+C}|w_t) \tag{5}$$

Now, we need the second and the third assumption.

Assumption II: These conditional jointly distribution should be independent.

Assumption III: For a given t, each $P(w_i|w_t)$ is identity identify distribution

We can get the average conditional log likelihood is

$$\begin{aligned}
\mathscr{L}' &= \frac{1}{T}\sum_{t=1}^{T} log((P(w_{t-C:t-1}, w_{t+1:t+C}|w_t)) \\
&= \frac{1}{T}\sum_{t=1}^{T} log(\prod_{i\in[-C,C],i\neq=0} P(w_i|w_t)) \\
&= \frac{1}{T}\sum_{t=1}^{T} \sum_{i\in[-C,C],i\neq=0} log(P(w_i|w_t))
\end{aligned}$$

Because in a machine learning model, we always care about how to minimize a loss function rather than maximize a likelihood function, so we can do some changes for the average conditional log likelihood function and get

$$Loss = -\frac{1}{T}\sum_{t=1}^{T} \sum_{i\in[-C,C],i\neq=0} log(P(w_i|w_t)) \tag{6}$$

We simply need to build a model for $P(w_i|w_t)$ to derive the likelihood function for the loss function. In fact, if we choose a word and a window size of $2C$, the sequence in this window becomes irrelevant because the conditional probability is all that matters. Then we may represent this conditional probability in a more generic way with $P(W_{Ouput}|W_{Input})$.

Now that we know what the loss function is, our goal is to find the discrete random variable distributions for each discrete random variable. As a result, we can use neural networks (see Figure 4) to assist us in fitting these distributions. We should add a softmax layer at the end of the network because the outputs are probabilities.

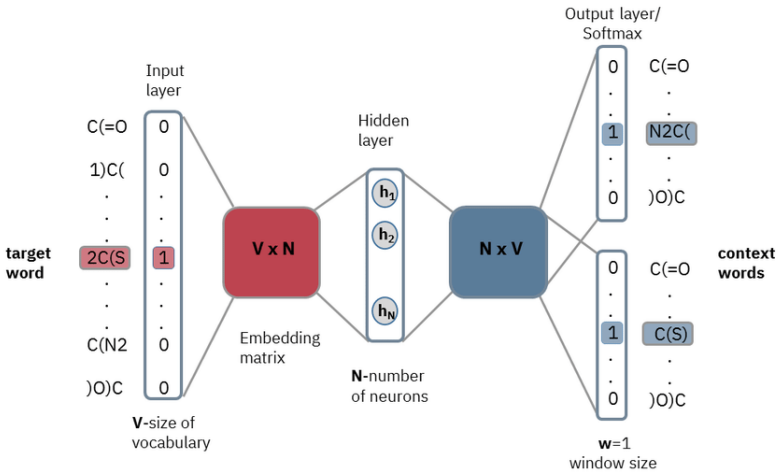

Figure 4: Skip-gram Model Structure[5]

Suppose the weights between input layer and hidden layer is U, the weights between hidden layer and output layer is V and the output of softmax layer is Δ (all vectors in this manuscript are column vectors), we can get

$$w_i^T * U * V = \Delta \tag{7}$$

, where w_i denotes the i-th word vector and Δ_i denotes the i-th output of softmax layer. If we use one-hot representation to obtain our initial word vector, we can get

$$\begin{aligned} w_i^T * U * V &= w_i^T * (u_1, u_2, ..., u_|N|) * (v_1, v_2, ..., v_{|V|}) \\ &= u_i^T * (v_1, v_2, ..., v_{|V|}) \\ &= (u_i^T v_1, u_i^T v_2, ..., u_i^T v_{|V|}) \\ &= (\Delta_1, \Delta_2, ..., \Delta_{|V|}) \end{aligned}$$

Finally, we can get

$$\Delta_j = \frac{exp(u_i^T v_j)}{\sum_{k=1}^{|V|} exp(u_i^T v_k)} \tag{8}$$

We now have the neural network for word embedding and the word vector weights U. This model is known as skip-gram and is a subset of word2vec (the other one is called Continuous Bag of Words, CBOW, see Figure 5). CBOW uses the center word to predict the contexts to predict the center word, and Skip-gram utilizes the center word to predict the contexts to predict the center word.

The distinction between CBOW and Skip-gram is that Skip-gram takes a center word as input and returns its context as output. CBOW, on the other hand, takes contexts as input and produces the center word as output.

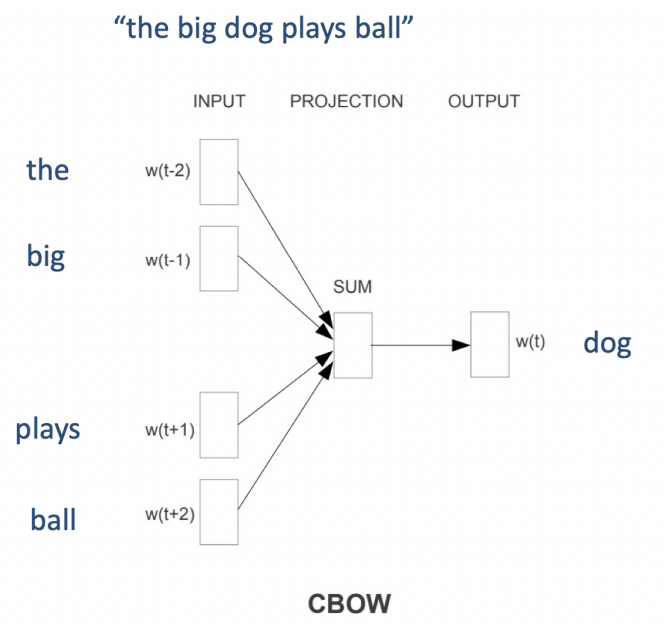

Figure 5: CBOW Structure[6]

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] S. Okada, M. Ohzeki, and S. Taguchi, "Efficient partition of integer optimization problems with one-hot encoding," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[3] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.

[4] H. M. Wallach, "Topic modeling: beyond bag-of-words," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 977–984.

[5] K. W. Church, "Word2vec," *Natural Language Engineering*, vol. 23, no. 1, pp. 155–162, 2017.

[6] T. Kenter, A. Borisov, and M. De Rijke, "Siamese cbow: Optimizing word embeddings for sentence representations," *arXiv preprint arXiv:1606.04640*, 2016.

[7] https://mungingdata.wordpress.com/2017/11/25/episode-1-using-tf-idf-to-identify-the-signal-from-the-noise