Build Skip-Gram with Negative-Sampling from Scratch

Miao WANG, Jiaqian MA, Zeliang LIU, Yue CHEN

Feb 18, 2022

Introduction

Word2vec is to convert words to vectors and make predictions. The object is to create non-sparse vectors that could represent words well so that each unique word in the corpus being assigned a corresponding vector in the space.

Among the most popular techniques used in Word2Vec, Skip-gram is gaining more traction, which is a 2-layer neural network and once optimized, the first weight matrix, the input embedding matrix can be interpreted perfectly as the vector representations of words, in the sense that similar words will have similar vectors.

Vanilla SoftMax-based skip-Gram is computationally expensive, and based on that, researchers developed negative sampling.

Although deriving from Skip-Gram, negative sampling word embeddings follow a completely different training goal, which is to find parameters to maximize the probabilities that all the observations indeed came from the data and it is calculated based on a joint probability of center word and context word instead of a conditional probability used in vanilla skip-gram.

In this report, we would talk about how we build it from scratch and our further experiments on hyper parameter tuning and initialization based on the baseline model.

Mathematical Derivation

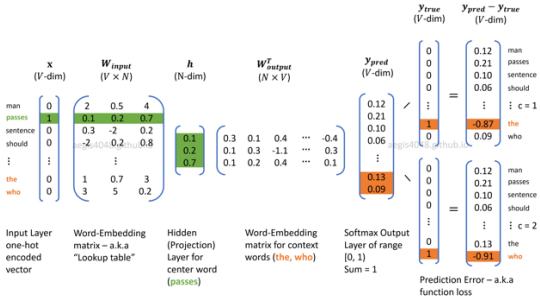

Figure 1 Neural network structure of Skip-Gram

In the vanilla Skip-Gram, the cost function using SGD is computed as such:

$$J(\theta; w^{(t)}) = -\sum_{c=1}^{C} log \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

However, inferring from this formula, we can tell that this SoftMax loss function is computationally expensive as it requires scanning through the entire output weight matrix W_{output} to compute the probability distribution of all V words.

To optimize this computation burden, negative sampling comes into play. The major difference between vanilla Skip-Gram and negative sampling is how many neurons of output weight matrix will be updated for each training sample.

In negative sampling, for each training sample (positive pair: w and $c_{positive}$), we should randomly draw K number of negative samples from a noise distribution $P_n(w)$. And instead of updating update V×N neurons as in vanilla Skip-Gram, we will only update (K+1)×N neurons in the output weight matrix.

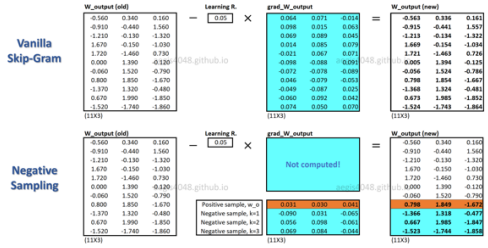

Figure 2 Model structure comparison

With negative sampling, word vectors are no longer learned by predicting context words of a center word. Instead of using SoftMax to compute the V-dimensional probability distribution of observing an output word given an input word, $p(W_o|W_I)$, the model uses sigmoid function to learn to differentiate the actual context words (positive) from randomly drawn words (negative) from the noise distribution $P_n(w)$.

Negative sampling attempts to maximize the probability of observing positive pairs p($C_{positive}$|weight)→1while minimizing the probability of observing negative pairs p($C_{negative}$|weight)→0.

The idea is that if the model can distinguish between the likely (positive) pairs vs unlikely (negative) pairs, good word vectors will be learned (KIM, 2019).

The derivations illustrated below are from the reference 1 (Goldberg and Levy, 2014).

The optimization objective can be defined as:

$$\begin{aligned}
&\arg\max_\theta \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)\\
&=\arg\max_\theta \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} (1-p(D=1|c,w;\theta))\\
&=\arg\max_\theta \sum_{(w,c)\in D} \log p(D=1|c,w;\theta) + \sum_{(w,c)\in D'} \log(1-p(D=1|w,c;\theta))\\
&=\arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c\cdot v_w}} + \sum_{(w,c)\in D'} \log(1-\frac{1}{1+e^{-v_c\cdot v_w}})\\
&=\arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c\cdot v_w}} + \sum_{(w,c)\in D'} \log(\frac{1}{1+e^{v_c\cdot v_w}})
\end{aligned}$$

If we let $\sigma(x)=\frac{1}{1+e^{-x}}$ we get:

$$\begin{aligned}
&\arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c\cdot v_w}} + \sum_{(w,c)\in D'} \log(\frac{1}{1+e^{v_c\cdot v_w}})\\
&=\arg\max_\theta \sum_{(w,c)\in D} \log \sigma(v_c\cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c\cdot v_w)
\end{aligned}$$

Where D denotes positive samples and D' denotes negative sample, v_c stands for context word vector while v_c represents center word vector .

So, we can take

$$\sum_{(w,c)\in D} \log \sigma(v_c\cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c\cdot v_w) \qquad (1)$$

as the loss function.

Gradient update on output weight matrix

With negative sampling, we do not update the entire output weight matrix W_{output}, but only a fraction of it.

Deriving from equation (1), we can update the output weight matrix as such:

$$\bar{c}_j^{(new)} = \bar{c}_j^{(old)} - \eta\cdot(\sigma(\bar{c}_j\cdot h) - t_j)\cdot h \qquad (2)$$

Where $t_j=0$ for negative words while $t_j=1$ for positive words. $\bar{c}_j$ is the j-th word vector in W_{output}.

Gradient update on input weight matrix

Just like vanilla Skip-Gram, only one word vector that corresponds to the input word in W_{input} is updated with negative sampling.

Similar to the deduction of equation (2), we can have gradient update of W_{input} as:

$$\bar{w}^{(new)} = \bar{w}^{(old)} - \eta\cdot\sum_{c_j\in\{c_{pos}\}\cup W_{neg}} (\sigma(\bar{c}_j\cdot h) - t_j)\cdot\bar{c}_j \qquad (3)$$

Model structure

Based on the skeleton codes provided, we divided the entire model into following 4 main building blocks.

Preprocessing

Inside `text2sentences()`, we applied punctuation removal, numbers removal, lemmatization to get cleaned root words.

Initialization

W_{input} and W_{output} are initialized here. Further experiments on initialization will be mainly focused on here. Also, since we have limited corpus size, when running test words, to avoid error, we created an `'unknown_vector'` by `np.random.normal` to represent word in test data that doesn't exist in our train set.

Negative sampling

Inside `negative_sampling()`, we defined $P_n(w)$ noise distribution by Raising the unigram distribution U(w) to the power of α has an effect of smoothing out the distribution.

$$P_n(\omega) = \left(\frac{U(\omega)}{2}\right)^\alpha$$

$U(\omega)$ is occurrences of a certain word divided by V. It attempts to combat the imbalance between common words and rare words by decreasing the probability of drawing common words, and increasing the probability drawing rare words. We pick α=0.75 here. And for each positive sample (WordId, ContextId) passed, we will return a list of negativeIds.

Training

Inside `trainWord()`, during backpropagation, the gradient update processes of W_{input} and W_{output} are computed as equation (2) &(3).

Impact of hyper parameters

Hyper-parameters

In this model, we have in total 7 hyper parameters to be tuned, however we didn't include α for fine tuning.
nEmbed: The length of word vector.
negativeRate: For each positive sample, how many negative sample to be chosen.
winSize: window size to be considered as context words.
minCount: The minimum requirement of word occurences. If a word appears less than it, this word is not included in the training set.
learning rate: learning rate for gradient updates.

Random Search

Random search randomly chooses hyper-parameters from the candidate space to get optimized hyper-parameters. Though Grid search and manual search are the more classical choices for hyper-parameter optimization, randomly chosen trials are more efficient. Granting random search the same computational budget, it will find better models by effectively searching a larger, more promising configuration space.
Search space settings:
nEmbed: 50,100,150;
negativeRate: 5,10,15;
winSize:3,6,9;
minCount: 0,3,6;
learning rate: 0.0001, 0.001, 0.01, 0.1.

No.	nEmbed	negative Rate	winSize	minCount	learning rate	loss	correlation
1	150	10	9	6	0.1	**-0.5432**	**0.1061**
2	100	10	9	6	0.1	-0.2500	0.1051
3	50	10	3	6	0.1	-5.2858e-06	0.1008
4	50	15	6	6	0.01	4.6688e-09	0.0946

Table1 Examples of model performance

As we can see, nEmbed has an impact on the result. Within the length of 50, 100, 150, the longer the transferred vector, more precise our model is. Also, a model with bigger winSize performs better since it is fed with more context information.

Impact of initialization

There are two weights matrices in the process, the input matrix (word embedding), and the output matrix (context embedding). They were used to capture relationships among words in a vector space.
Optimizing the embedding weight matrices contributes to representing words in a higher quality vector space so that the model will be able to find useful relationships among words. Each row in a word-embedding matrix is a word-vector for each word. The embedding matrix behaves like a look-up table. It would be useless to apply matrices with all 0s or all 1s as every word vector will be the same vector with N dimensions. The matrix should represent words exclusively and uniquely, so we define random matrices as weight matrices. As a result, the impact of initialization lies on the impact of different random weight matrices.
In order to explore the impact of initialization of input embedding and output matrix on skip gram model, we conducted several experiments with various ranges of random initialization, from (-1,1) to (-0.0001, 0.0001).

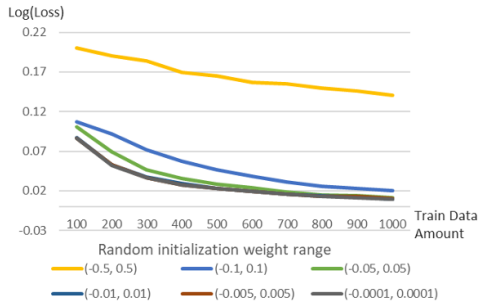

Figure 3 Impact of different random initializations on Loss

The results for random initialization at (-1,1) and (-0.8, 0.8) are not shown here. The results of these two larger variance initializations lead to higher log loss values, hence out of scope.

From the figure above, we can conclude that the loss shows a downward trend as more data were trained in all random initializations, while the loss of model initialized in a narrower range converges faster.

Initialization	(-0.5, 0.5)	(-0.1, 0.1)	(-0.05, 0.05)	(-0.01, 0.01)	(-0.005, 0.005)	(-0.0001, 0.0001)
Correlation	0.096	0.1024	0.0957	0.0931	0.0931	0.0933

Table 2 Correlation result of different initializations

There is no significant difference on correlation result between models with different random initialization, thus the variance of initialization does not affect model performance much.

Conclusion

With experiments on hyper parameter tuning and initializations, we found out that the setting for best model is nEmbed=100, negativeRate=10, winSize=9, minCount=6, learning=0.1 with (-0.5,0.5) initializations.

Through this project, we did a tear-down of Skip-Gram negative sampling technique and gained some hands-on practice on how to process word materials.

Plus, We believe with a larger train set (we only have a corpus of 1000 sentences for now), our model can achieve better performance.

Further work

Biasedness test

Recent studies demonstrate that word embeddings contain and amplify biases present in data, such as stereotypes and prejudice. Basically, the problematic bias can be generated from the corpora we used for training and the word embedding model we built (Heimerl and Gleicher, 2018).

Take gender bias for instance, the gender-neutral words like “programmer” or “nurse” are found out to have stronger connection to particular gender, which is caused by the stereotype and human bias presented in the corpus.

The WEAT(Word Embedding Association Test) is the most common tool for testing bias in word embedding model(Caliskan et al.2019).As we known, skip gram model convert input text into an output of number, and the semantically similar words are closer in the embedding space. The WEAT measures how a target words set(e.g., different occupations) associate with the attribute word set (e.g., different genders, races) by calculating their cosine similarity between the embedding vector pairs.

Reference

- Bergstra, James, et al. “Random Search for Hyper-Parameter Optimization Yoshua Bengio.” Journal of Machine Learning Research, vol. 13, 2012, pp. 281–305, www.jmlr.org/papers/volume13/bergstra12a/bergstra12a?ref=githubhelp.com. Accessed 18 Feb. 2022.
- Caliskan, Aylin, et al. “Semantics Derived Automatically from Language Corpora Contain Human-like Biases.” Science, vol. 356, no. 6334, 13 Apr. 2017, pp. 183–186, science.sciencemag.org/content/356/6334/183, 10.1126/science.aal4230. Accessed 14 Nov. 2019.
- Goldberg, Alfred, and Omer Levy. “Ord2vec Explained: Deriving Mikolov et Al.'S Negative-Sampling Word-Embedding Method.” Current Biology, vol. 24, no. 17, Sept. 2014, pp. R780–R782, 10.1016/j.cub.2014.08.014. Accessed 18 Feb. 2022.
- Heimerl, F., and M. Gleicher. “Interactive Analysis of Word Vector Embeddings.” Computer Graphics Forum, vol. 37, no. 3, June 2018, pp. 253–265, 10.1111/cgf.13417.
- KIM, ERIC. “Optimize Computational Efficiency of Skip-Gram with Negative Sampling.” Pythonic Excursions, 26 May 2019, aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling. Accessed 18 Feb. 2022.