Report of Homework 2: Sentiment analysis based on feature engineering and word2vec

Name: Jialun Shen, Student id: 16307110030

November 18, 2019

1 Word2vec Training

In *word2vec.py*, I implemented the word2vec models and trained my own words vectors with stochastic gradient descent method based on Stanford Sentiment Treebank(SST) dataset. Two types of sampling cost and gradient functions are used: softmax and negative sampling.

1.1 Softmax Cost Function

Given a predicted word vector v_c, corresponding to the center word c for skipgram, word prediction is made with the softmax function:

$$\hat{y} = p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

where w denotes the w-th word, $u_w (w = 1, ..., V)$ is the "output" word vector for the w-th word, v_c is the "input" vector for c, word o is the expected word, and V is the size of the vocabulary.

Cross entropy(CE) cost(i.e., negative log likelihood) is applied to this prediction, thus the cost function is

$$\begin{aligned} J_{softmax-CE}(o, v_c, U) = \mathrm{CE}(y, \hat{y}) &= -\log\ p(o|c) \\ &= -\log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \\ &= -u_o^T v_c + \log S \end{aligned}$$

where $S = \sum_{w=1}^{V} \exp(u_w^T v_c)$, and $U = [u_1, ..., u_w]$ is the matrix of all the output vectors.

The gradient with respect to v_c, and gradients for u_w's (including u_o) are

$$\begin{aligned} \frac{\partial J_{CE}}{\partial v_c} &= -u_o + \sum_{w=1}^{V} \frac{\exp(u_w^T v_c)}{S} u_w \\ \frac{\partial J_{CE}}{\partial u_w} &= -v_c \mathbb{I}(w = o) + \frac{\exp(u_w^T v_c)}{S} v_c \end{aligned}$$

where $\mathbb{I}(\cdot)$ is the indicator function. Calculation of softmax cost function and gradients is implemented in the function $softmaxCostAndGradient$.

1.2 Negative Sampling Cost Function

Suppose we use negative sampling loss, K negative samples(words) are drawn, denoted by u_k's(k=1,...,K,$o \notin \{1, ..., K\}$) for simplicity.

With the same notations in section 1.1, the negative sampling loss function in this case is

$$J_{neg-sample}(o, v_c, U) = -\log(\sigma\Big(u_o^T v_c\Big)) - \sum_{k=1}^{K} \log(\sigma\Big(-u_k^T v_c\Big))$$

where $\sigma(x) = 1/(1 + \mathrm{e}^{-x})$ is the sigmond function. Note that $\sigma(-x) = 1 - \sigma(x)$ and $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

The gradient with respect to v_c, and gradients for u_w's (including u_o) are

$$\begin{aligned}
\frac{\partial J_{NS}}{\partial v_c} &= -(1 - \sigma\Big(u_o^T v_c\Big))u_o + \sum_{k=1}^{K}(1 - \sigma\Big(-u_k^T v_c\Big))u_k \\
&= -\sigma(-u_o^T v_c)u_o + \sum_{k=1}^{K} \sigma(u_k^T v_c)u_k \\
\frac{\partial J_{NS}}{\partial u_w} &= -(1 - \sigma\Big(u_o^T v_c\Big))v_c \mathbb{I}(w = o) + \sum_{k=1}^{K}(1 - \sigma\Big(-u_k^T v_c\Big))v_c \mathbb{I}(w = k) \\
&= -\sigma(-u_o^T v_c)v_c \mathbb{I}(w = o) + \sum_{k=1}^{K} \sigma(u_k^T v_c)v_c \mathbb{I}(w = k)
\end{aligned}$$

Calculation of negative sampling cost function and gradients is implemented in the function *negSamplingCostAndGradient*.

1.3 Training of Word Vectors

For skip-gram, the cost for a context around c is

$$J_{skip-gram}(w_{c-m...c+m}) = \sum_{-m \le j \le m, j \ne 0} F(w_{c+j}, v_c)$$

where $[w_{c-m}, ...w_{c-1}, w_{c+1}, ..., w_{c+m}]$ is the set of context words for w_c. v_k and u_k is the "input" and "output" word vectors for w_k, F can be replaced with J_{CE} or J_{NS}.

Negative sampling cost function is used in training of word vectors because it is much more efficient than the softmax-CE loss. For each time we call J_{CE}, we have to calculate the inner product $u_w^T v_c$ for all words in vocabulary V, while J_{NS} only requires a negative sample of size $K(K = 10$ in implementation).

After 40000 iterations, the total cost is reduced to 9.438424. The training process takes approximately an hour, thanks to the relatively small vocabulary with a size of 19537. In the figure of visualization of my word vectors, positive words are not separated with negative words clearly, however.

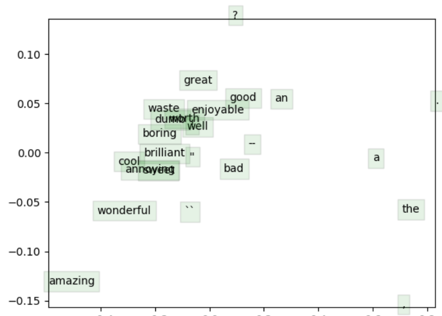

Figure 1: visualization of word vectors

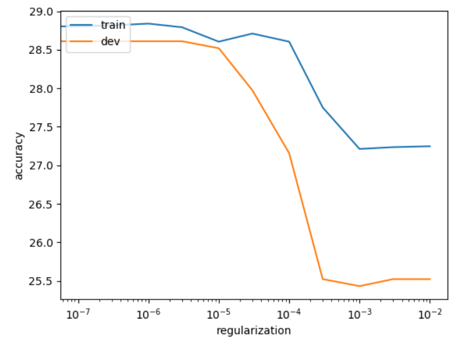

Figure 2: reg-acc of word2vec method

2 Sentiment Analysis Based on Word2vec

For word2vec-based sentiment analysis, the average of all the word vectors is used as the feature for each sentence. This is implemented by the function *getSentenceFeature* in *softmaxreg.py*. In *sentiment_word2vec.py*, diffenent regularization parameters are tested. The "optimal" parameter is 1.0e-6, which has an accuracy of 26.88% on test set.

Table 1: Accuracy of diffenent regularization parameters on training and develepment set

Reg	Train acc(%)	Dev acc(%)	Reg	Train acc(%)	Dev acc(%)
0.0	28.80	28.61	3.0e-5	28.71	27.97
1.0e-7	28.80	28.61	1.0e-4	28.60	27.16
3.0e-7	28.82	28.61	3.0e-4	27.75	25.52
1.0e-6	28.84	28.61	1.0e-3	27.21	25.43
3.0e-6	28.79	28.61	3.0e-3	27.24	25.52
1.0e-5	28.60	28.52	1.0e-2	27.25	25.52

3 Sentiment Analysis Based on Feature Engineering

For feature engineering-based sentiment analysis, a naïve bayes classifier with bag of words features is trained in *sentiment_bagofwords.py*. It has an accuracy of 86.75%, 37.24% and 38.51% on train, dev and test set respectively. Naïve bayes performs better than word2vec method with simplier model and much less calculation in this task. Five most informative features are:

Table 2: Five most informative features

Feature	class1:class2	#d class1:class2
worst=1	0:3	21.7:1.0
?=1	2:4	18.4:1.0
bad=1	0:4	18.2:1.0
wonderful=1	4:1	17.9:1.0
moving=1	4:0	16.9:1.0