Assignment 1: Word Vectors

Niranjan Balsubramanian and Jun Kang

CSE 538 Fall 2018

1 Due Date and Collaboration

- The assignment is due on Oct 1st, 2018 at 11:59 pm. We will accept late submissions until Oct 3 11:59 pm with a penalty of 10 points per day. (Out of 100 points)
- You can collaborate to discuss ideas and to help each other for better understanding of concepts and math.
- You should NOT collaborate on the code level. This includes all implementation activities: design, coding, and debugging.
- You should NOT not use any code that you did not write to complete the assignment.
- The homework will be cross-checked. **Do not cheat at all! It's worth doing the homework partially instead of cheating and copying your code and get 0 for the whole homework.**
- We collected a list of FAQ based on previous semester (many thanks to Jun for his effort on answering questions) it is kept here and will be updated periodically. Before asking question in piazza check it first.

2 Word Vectors

In this assignment you will implement methods to learn word representations building on Skip-grams and apply the learned word representations to a semantic task.

A skeletal TensorFlow implementation for learning word representations will be provided along with a README file. Also, to reduce the training time, pretrained models will be provided.

In this document, we describe the overview of tasks and related concepts for the assignment. All instructions related to the implementation can be found in the README file.

2.1 Skip-grams

To train word vectors, you need to generate training instances from the given data. You will implement a method that will generate such training instances in batches. (Refer to README file.)

2.2 Loss Functions

You will implement two loss functions to train word vectors: (1) Cross entropy loss – the negative of the likelihood objective function we saw in class, and (2) Noise Contrastive Estimation – an alternate function that explicitly uses a set of k negative words.

2.2.1 Cross entropy loss function

This is the negative of the log-likelihood objective function we saw in class. These are called "loss" functions since they measure how bad the current model is from the expected behavior.

Refer to the class notes on this topic and refer to the README file and code for instructions.

2.2.2 Noise contrastive estimation loss function

The NCE method formulates a slightly different classification task and a corresponding loss. (This paper describes the method in detail:
`https://www.cs.toronto.edu/~amnih/papers/wordreps.pdf`)

Here we will give you a gist of the paper and the main equations you need for implementation.

The idea here is to build a classifier that can give high probabilities to words that are the correct target words (i.e., w_o) and low probabilities to words that are incorrect target words. As with cross entropy loss, here we define the classifier using a function that uses the word vectors of the context and target as free parameters.

The key difference however is that instead of using the entire vocabulary, here we sample a set of k negative words (we denote by V^k) for each instance, and create an augmented instance which is a collection of the true target word and k negative words. Now the vectors are trained to maximize the probability of this augmented instance. The paper details how to compute the probability of this augmented instance.

The key for implementation purposes is to know that the NCE loss function is defined in terms of (i) a function defined over the target and context vectors $s(w_o, w_c)$, and (ii) the unigram probability of the target word ($Pr(w_o)$). The NCE loss function for the current set of parameters θ (i.e., the current vectors) over a given batch of instances is given by:

$$J(\theta, Batch) = \sum_{(w_o, w_c) \in Batch} - \left[\log Pr(D = 1, w_o | w_c) + \sum_{x \in V^k} \log(1 - Pr(D = 1, w_x | w_c)) \right]$$

where,

$$Pr(D = 1, w_o | w_c) = \sigma\left(s(w_o, w_c) - \log\left[kPr(w_o)\right]\right)$$
$$Pr(D = 1, w_x | w_c) = \sigma\left(s(w_x, w_c) - \log\left[kPr(w_x)\right]\right)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and

$$s(w_o, w_c) = (u_c^T u_o) + b_o$$

where u_c, and u_o are the context and the target word vectors, and b_o is a bias vector specific to w_o.

The code includes a function for computing unigram probabilities.

2.3 Analogies using word vectors

You will evaluate the word vectors you learned from both approaches (loss functions) using word analogy tests.

Consider the following word pairs. The words in each pair are related in the same way:

pilgrim:shrine, hunter:quarry, assassin:victim, climber:peak

We can say these words share the same relation R. A pilgrim's target is the shrine, the hunter's target is the quarry and so on.
Now you are given a list of candidate word pairs, which may or may not share the same relation as in the list above.

(1) pig:mud
(2) politician:votes
(3) dog:bone
(4) bird:worm

The analogy task is to answer two questions.

Q1. Which word pair has the MOST illustrative (similar) example of the relation R?
Q2. Which word pair has the LEAST illustrative (similar) example of the relation R?

For each question, there are examples pairs of a certain relation. Your task is to find the most/least illustrative word pair of the relation.

One simple way you can do this is using simple vector arithmetic.

Recall that vectors are representing some direction in space. If (a, b) and (c,d) pairs are analogous pairs, then the transformation from a to b (i.e., some x vector when added to a gives b: a + x = b) should be highly similar to the transformation from c to d (i.e., some y vector when added to c gives d: b + y = d). In other words, the difference vector (b-a) should be similar to difference vector (d-c).

For this task, compute the difference vectors of the example word pairs to learn the common relation of them. Then, examine each choice to find the most/least illustrative pair.

See README and code for implementation details and instructions.

3 Experiments

You will conduct some hyper-parameter tuning experiments to figure out the best way to learn the word vectors. You will learn different word vectors and try them on the analogy task for the training data set (word_pairs_dev.txt). You will pick the configuration that works best for each loss function as your best word vector under that method.

Your goal is to understand how the different hyper-parameters affect the embeddings. You will try the following hyper-parameters. You can do just a few or a gazillion experiments here. I'd recommend trying at least five configurations for each method. There are really very few rule of thumbs at this point on how to do these experiments. You can consult any resources to figure out a good way to go through the hyper-parameter settings.

- Experiment with different sized context windows. This would require modifying the code that sets up the context windows.
- Instead of evaluating over the entire set of negative contexts for the cross entropy loss, you can simply sample some negative words to compute the loss. This sampling can be done based on the unigram probability of the negative words. You can try different number of negative words to see its impact on performance. You can also vary the k in NCE loss as well.
- You can vary the vector sizes themselves, the learning rate for gradient descent, the number of epochs to train for etc.

Your goal should be achieve the best performance on the hidden test set. Find settings where the model can generalize well for the external task.

4 What should you turn in?

- README – Implementation and experiment details such as exact configurations and training steps

- Code – The implementations for the word2vec model, the two loss functions, and the code for computing analogies.
- Model – The best word2vec model files for NCE and Cross-entropy methods: nce.model, and cross-entropy.model. Please indicate in your README, the exact configuration that produced these models.
- Prediction file – For the analogy test, generate predictions for the test file using your best NCE and cross-entropy model. (two predictions)
- Report (PDF) – This should a 3-4 page report, which includes four sections:
  1. Hyper-parameters explored w/ a description of what each parameter controls and how varying each is likely to affect the training. These are the parameters I suggest to tune: max_num_steps, batch_size, skip_window, num_skips
  2. Results on the analogy task for five different configurations of hyper-parameters, along with the best configuration (on the word_pairs_dev.txt). Explain these results. Are there any trends you observe? For example, what does increasing dimensions of vectors do? You should say what you learned from exploring these configurations. You are free to consult any resources to support your arguments.
  3. Top 20 similar words according to your NCE and cross entropy model for the words:{first, american, would}. Please report these in a table. And discuss what kinds of similarities do you notice?
  4. A summary of the justification behind the NCE method loss. No more than a page. This should be a summary of section 3 (3.1 specifically) in the NCE paper. Explain how the proposed loss function models the probability distribution of a word in its context.

5 Grading

Here is a coarse grained grading rubric.

- 30 points – Cross entropy loss implementation.
- 30 points – NCE loss implementation.
- 20 points – Experimental evaluation.
- 10 points – 5 points for getting within a reasonable delta of the TA's implementation on the analogy test set. The remaining 5 points will be distributed based on a relative grading on the best performing system.
- 10 points – A summary of the justification for the NCE method loss.