GSoC '17 Project Proposal

Performance Improvement in gensim and fastText

Name and Contact Information

- **Name :** Prakhar Pratyush
- **University :** Indian Institute of Technology Roorkee, India
- **Email :** er.prakhar2b@gmail.com
- **Github/ Gitter :** prakhar2b
- **Video Chat :** hangouts (er.prakhar2b@gmail.com)

Pre- GSoC involvements

Here is the list of PRs I have contributed to -

- **#1186 (merged):** Fix #824 : no corpus in init, but trim_rule in init in word2vec
- **#1190 (merged):** Opened #1187 and modified error message in word2vec
- **#1200 :** Fix #1198 changed params to make API consistent with the code
- **#1208 (merged):** grepped through all the wrapper codes to fix #671
- **#1226 :** Fix #691 : index object pickle to be always under `index.output_prefix`

Other contributions in **giacbrd/shallowLearn** (relevant to this project)

- #16 **-** opened the issue to clarify the ambiguity in the performance graph
- **#13 -** Cythonize the hash function (currently working on this)
- **#17** - predicting multiple documents at once (wishlist) ; @giacbrd opened this issue after looking into this (Ipython) benchmark.

Abstract

Gensim is an NLP library which trains a huge amount of data for semantic/syntactic analysis and topic modelling. For such a huge amount of data (~ billions of words), implementation needs to be reasonably fast, otherwise difference maybe in the order of days or even weeks of training time.

Facebook Research recently released a new word embedding and classification library fastText, which performs better than word2vec on syntactic task, and trains much faster for supervised text classification. The architecture of supervised text classification is a modified version of word2vec cbow model, and takes into account the n-gram features implemented using the hashing trick. The initial implementation by facebook was in C++, later @giacbrd produced a Python implementation **(labeled word2vec)** building on top of gensim code.

The goal of this project is :-

- Evaluate and benchmark the performance of fastText in python (labeled word2vec).
- Taking inspiration from gensim word2vec code optimization, this project aims at refactoring/ reorganizing, and re-implementing the frequently used and time consuming part in Cython **to match the performance with fastText (Facebook) and vowpal wabbit supervised classification.**
- After finalizing the parallel version, GPU version of the code will be implemented which might require re-writing some part in TensorFlow.
- For collocations (n-gram), gensim has phrases module. This project also aims at multi-core implementation of **phrases module in cython.**
- To improve gensim overall performance, other bottlenecks will be found and refactored throughout the project.
- Implement hashing trick in labeledword2vec (wishlist)

Technical Details

In unsupervised mode, the cbow model of word2vec works by sliding a window over the words (context), and building a weight matrix by trying to predict the middle word (target). This weight matrix turns out to be our word vector, which we later use for other NLP tasks.

fastText supervised classification is very similar to the word2vec cbow model. It continues the philosophy of word2vec that **"the closer the word is to the target the more predictive it should be"**. In word2vec, the goal was to obtain a word embedding, so the target was middle word (in context) for semantic similarity; but in fastText, the goal is classification, so the target (predicting task) is the label.

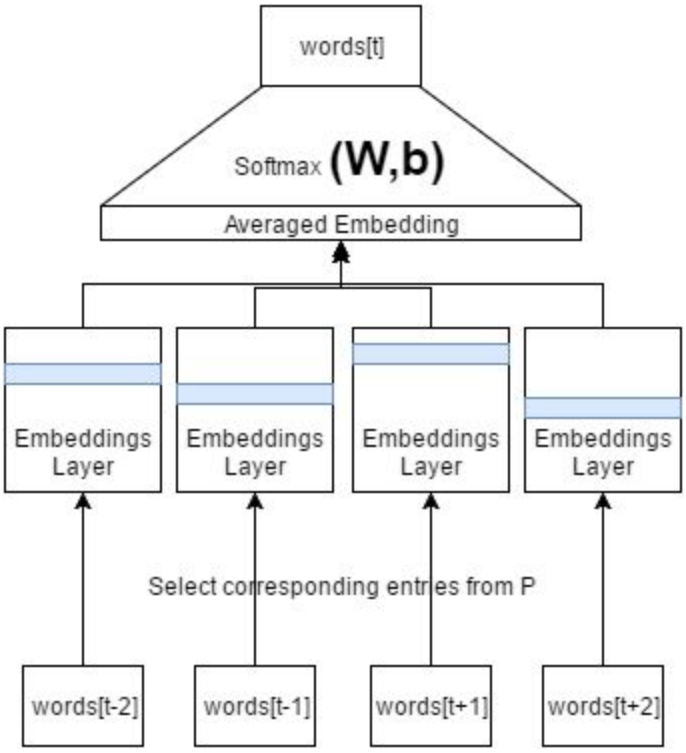

fig 1. Word2vec cbow model

Instead of sliding a window over the words, in fastText we slide over each entity related to the label (sentences, for text classification), and take N n-gram

features (N is the number of labels) from each sentence which is later averaged to form the text representation. We train the model in the same way as word2vec.

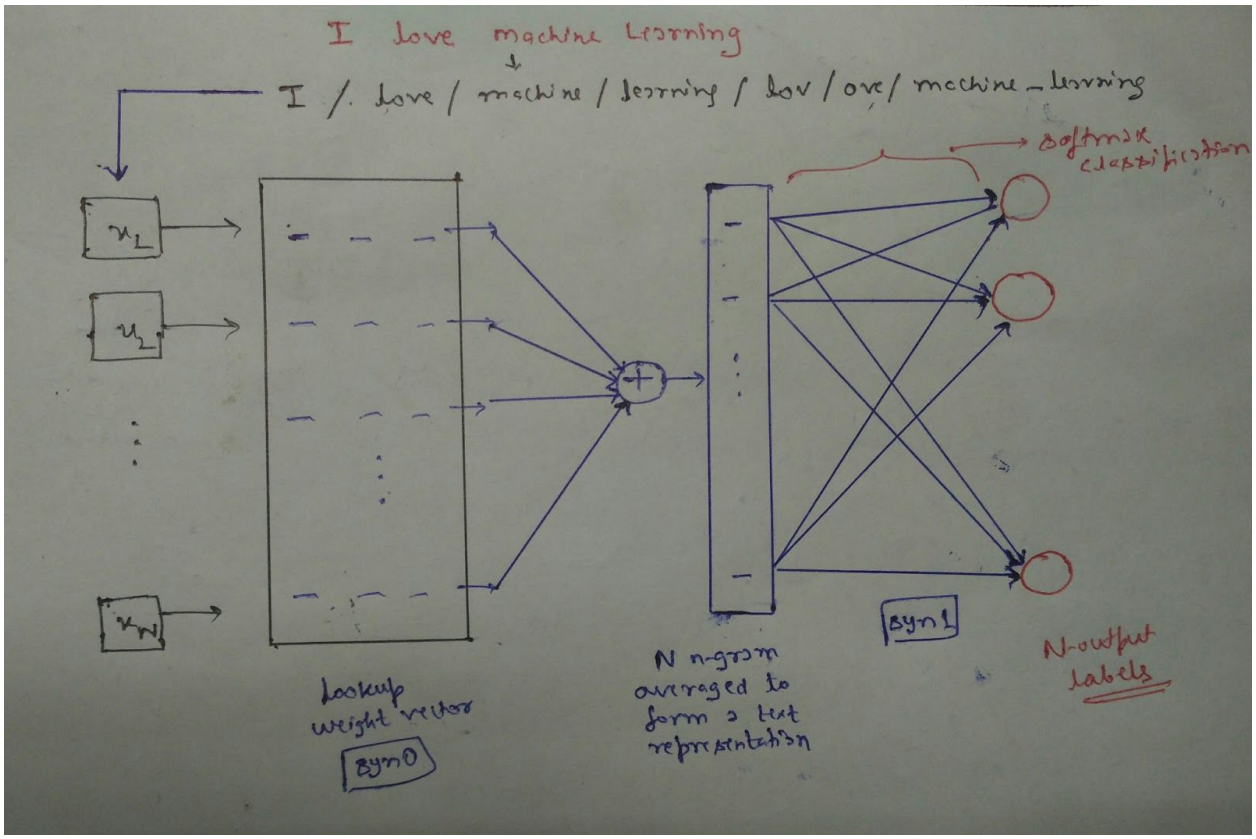

fig 2. fastText supervised classification

Gensim has a phrases module which provides collocations (bigram) based on how frequently adjacent words appear in the corpus. In general, n-gram features can be obtained by calling it multiple times.

phrases module is organized into two classes - **Phrases** and **Phraser**. Phrases to detect phrases and Phraser (faster and memory efficient) as an alternative that is built from initial Phrases instance.

Note:- prune_vocab only when **len(vocab) > max_vocab**, why not prune_vocab every time as we do in word2vec *scale_vocab*. Discuss this (maybe in community bonding period)

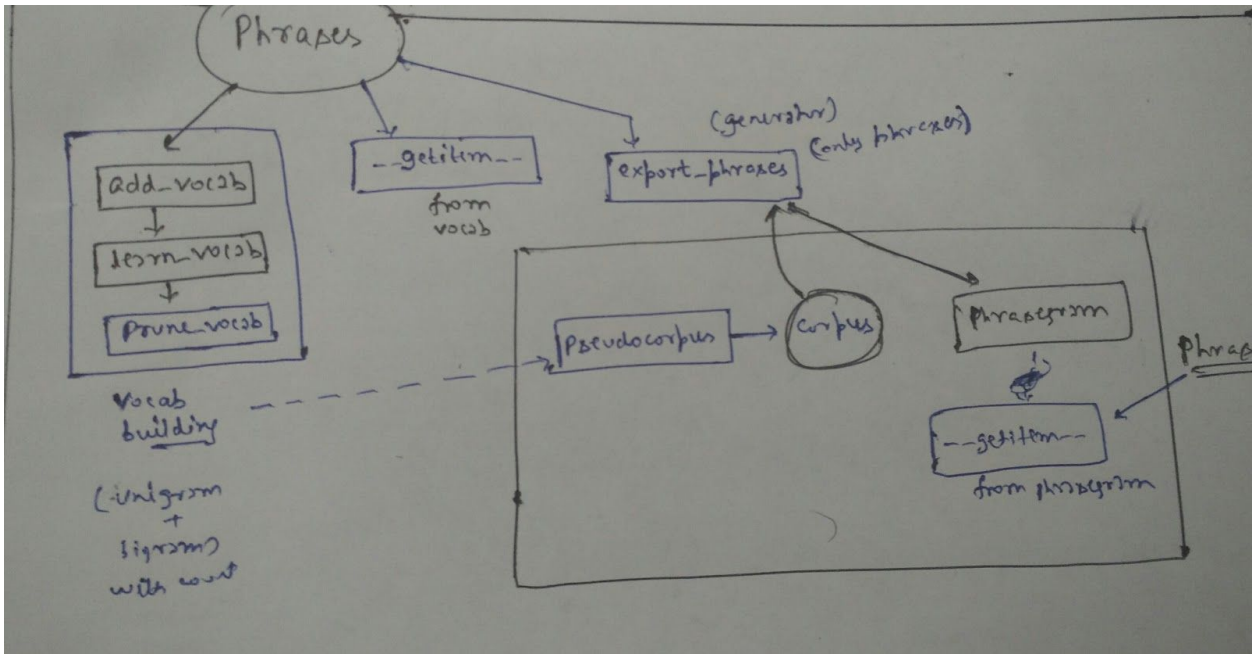

fig 3. Architecture of phrases.py in gensim

phrases.py

The phrases module in gensim is implemented purely in Python, therefore speed is quite slow as of now. If we provide input using phrases stream, for example, to word2vec, the training consumes input faster than phrases can supply it.

Therefore, one metric to evaluate speed would be to **reduce the time word2vec has to wait** while we provide input to it from phrases.

```
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1702   44.339    0.026  120.807    0.071 phrases.py:255(__getitem__)
12903158   23.411    0.000   45.446    0.000 utils.py:227(any2unicode)
12903178   12.167    0.000   12.167    0.000 {_codecs.utf_8_decode}
17005207   10.659    0.000   10.659    0.000 {method 'encode' of 'unicode' objects}
  677776    9.674    0.000   14.090    0.000 word2vec.py:1114(seeded_vector)
  677776    8.839    0.000    9.053    0.000 keyedvectors.py:92(__init__)
17005207    7.786    0.000   20.273    0.000 utils.py:218(any2utf8)
29910137    6.861    0.000    6.861    0.000 {isinstance}
    1701    6.380    0.004    6.380    0.004 {zip}
       1    5.493    5.493  130.635  130.635 word2vec.py:548(scan_vocab)
       1    5.475    5.475   15.021   15.021 word2vec.py:578(scale_vocab)
       1    4.935    4.935   19.871   19.871 word2vec.py:1098(reset_weights)
```

An initial profiling of phrases module show bottlenecks in the code that needs to be improved. For example, the learn_vocab function in Phrase class has a for loop that consumes most of the time.

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1   27.627   27.627   57.971   57.971 phrases.py:160(learn_vocab)
 17005207    9.813    0.000    9.813    0.000 {method 'encode' of 'unicode' objects}
 17005207    7.517    0.000   19.002    0.000 utils.py:218(any2utf8)
    12209    3.622    0.000    3.622    0.000 {method 'read' of 'file' objects}
     1701    2.899    0.002    2.899    0.002 {zip}
```

```
Timer unit: 1e-06 s

Total time: 125.224 s
File: /home/prakhar/anaconda2/lib/python2.7/site-packages/gensim/models/phrases.py
Function: learn_vocab at line 160

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   160                                           @staticmethod
   161                                           def learn_vocab(sentences, max_vocab_size, delimiter=
   162                                               """Collect unigram/bigram counts from the `sente
   163         1            3      3.0      0.0          sentence_no = -1
   164         1            2      2.0      0.0          total_words = 0
   165         1         2156   2156.0      0.0          logger.info("collecting all words and their coun
   166         1            5      5.0      0.0          vocab = defaultdict(int)
   167         1            2      2.0      0.0          min_reduce = 1
   168      1702      3262722   1917.0      2.6          for sentence_no, sentence in enumerate(sentences
   169      1701         6713      3.9      0.0              if sentence_no % progress_per == 0:
   170         1            3      3.0      0.0                  logger.info("PROGRESS: at sentence #%i,
   171         1         1945   1945.0      0.0                              (sentence_no, total_words, l
   172  17006908     46051379      2.7     36.8              sentence = [utils.any2utf8(w) for w in sente
   173  17005207     18789297      1.1     15.0              for bigram in zip(sentence, sentence[1:]):
   174  17003506     19528261      1.1     15.6                  vocab[bigram[0]] += 1
   175  17003506     24720384      1.5     19.7                  vocab[delimiter.join(bigram)] += 1
   176  17003506     12843629      0.8     10.3                  total_words += 1
```

peak memory: 541.90 MiB, increment: 436.45 MiB

export_phrases() also needs to be improved/ parallelized

```
 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 497069   22.709    0.000   80.378    0.000 phrases.py:216(export_phrases)
8293112   19.150    0.000   40.333    0.000 utils.py:218(any2utf8)
4146557    8.885    0.000   13.593    0.000 phrases.py:315(pseudocorpus)
```

```
229   4146557     20404254      4.9     14.3          for sentence in sentences:
230  12439668     58552322      4.7     41.0              s = [utils.any2utf8(w) for w in sentence]
231   4146556      2912165      0.7      2.0              last_bigram = False
232   4146556      3048995      0.7      2.1              vocab = self.vocab
233   4146556      2767347      0.7      1.9              threshold = self.threshold
234   4146556      2757095      0.7      1.9              delimiter = self.delimiter  # delimiter used for lookup
235   4146556      2758546      0.7      1.9              min_count = self.min_count
236   8293112     10877291      1.3      7.6              for word_a, word_b in zip(s, s[1:]):
237   4146556      5479885      1.3      3.8                  if word_a in vocab and word_b in vocab:
238   4146556      4636831      1.1      3.2                      bigram_word = delimiter.join((word_a, word_b))
239   4146556      3757021      0.9      2.6                      if bigram_word in vocab and not last_bigram:
240   4146556      4926712      1.2      3.4                          pa = float(vocab[word_a])
241   4146556      3720664      0.9      2.6                          pb = float(vocab[word_b])
242   4146556      3567098      0.9      2.5                          pab = float(vocab[bigram_word])
243   4146556      5201566      1.3      3.6                          score = (pab - min_count) / pa / pb * len(vocab
244                                                                  # logger.debug("score for %s: (pab=%s - min_cou
245                                                                  #     bigram_word, pab, self.min_count, pa, pb,
246   4146556      3395652      0.8      2.4                          if score > threshold:
247    497068       321344      0.6      0.2                              if as_tuples:
248    497068       370364      0.7      0.3                                  yield ((word_a, word_b), score)
249                                                                      else:
250                                                                          yield (out_delimiter.join((word_a, word
251    497068       381112      0.8      0.3                              last_bigram = True
252    497068       316155      0.6      0.2                              continue
253   3649488      2762405      0.8      1.9                          last_bigram = False
```

Possible Improvements -

- Efficient memory access
- Numpy, Numba (optional) etc
- Nested loop as matrix operation
- Cythonize
- BLAS
- Other tricks
- Parallelize

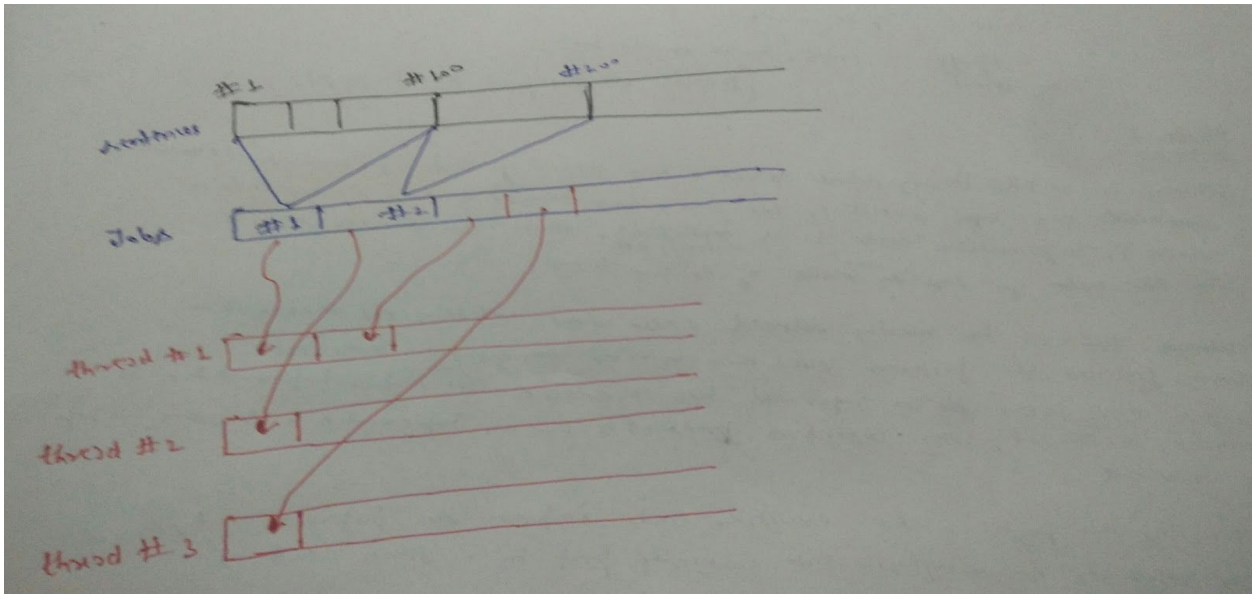

fig 4. Multiprocessing with worker threads in gensim [4]

LabeledWord2Vec

LabeledWord2Vec is currently designed as a subclass of Word2Vec. An initial profiling over pure python version shows it needs a lot of refactoring and optimization. Here is the IPython benchmark trained on dbpedia dataset.

Proposed change in architecture:-

BaseNN (Base Shallow Neural Network**)**
- train()

Word2Vec (child of BaseNN**)**
- Vocabulary building, and other codes

LabeledWord2Vec (child of BaseNN**)**
- Vocabulary building
- Label vocab building
    - Incorporate label vocab into KeyedVector. Default *None* for word2vec.
- Scan_vocab : currently it is count_raw_words()
- Scale_vocab, and other codes
- labeledw2v_inner.pyx - Cython code for score/loss etc.

Wishlist :- (Hashing trick) Gensim has corpora.hashdictionary. Try to introduce that into keyedvector, as it is very crucial in the performance of facebook fastText.

There are other chunks of codes which are very similar in both word2vec and labeledw2v, which can be organized in a better way. For example, update_weight, reset_weight etc.

Proposed Timeline

I have been involved with gensim for around two months now. I am familiar with most of the code base, so I'll be looking to start early so that things go smoothly during coding period. I often set pre-deadlines which helps in improvising in the last minutes too. I also strongly believe in Pareto principle that 80% of the effects come from 20% of the causes, which is even more relevant for this project as the primary goal is to streamline the bottlenecks and improve the code accordingly.

The whole project can be broadly categorized into milestones and labels :-

- **Milestone 1** : Phrases
    - **Label 1.1** : learn_vocab
    - **Label 1.2** : export_phrases
    - **Label 1.3** : pseudocorpus, phrasegrams, getitems
    - **Label 1.4** : multi-core
    - **Label 1.5** : Test **(Mid Term I)**

- **Milestone 2**: fastText (labeledw2v)
    - **Label 2.1** : BaseNN as parent class of word2vec
    - **Label 2.2** : Cythonize labeledw2v training
    - **Label 2.3** : Cythonize/ improve remaining parts
    - **Label 2.4** : Refactor/ improve, and compare with fastText/ vowpal wabbit
    - **Label 2.5** : Write tests and create benchmark **(Mid Term II)**
    - **Label 2.6** : Precision and Recall
    - **Label 2.7** : Predict multiple documents at one
    - **Label 2.8** : Save prediction result

- **Milestone 3**: GPU
    - **Label 3.1** : Training
    - **Label 3.2** : Write tests and create benchmark

- **Wishlist (W) :** work in free time / after Milestones
    - **Label W.1** : Hashing in labeledword2vec (hashing trick with bucket)
    - **Label W.2** : Cythonize / parallelize scan_vocab in word2vec

Community Bonding Period

- **May 4 - May 15 :**
    - Get more familiar with the code structure, and community by actively Participating in discussions (mailing list /gitter) and solving bugs/ issues.
    - Discuss and think more about possible change in architecture of phrases module.
    - Profile and benchmark phrases and other models in a more comprehensive manner.

- **May 16 - May 29 :**
    - Continue participation in mailing list /gitter / issue triage
    - Discuss and think more about possible change in architecture of Word2vec and labeledword2vec module.
    - Research more on TensorFlow (GPU)
    - debug, profile and benchmark labeledword2vec in a more comprehensive manner
    - set up blogs for weekly/ fortnight update. Keep your work documented

Week I (May 30 - June 5)

- Start with learn_vocab() function in phrases module, and refactor by using different optimization methods and benchmark the improvements.
- Create phrases_inner.pyx and write cython code for learn_vocab
- Work on add_vocab, and prune_vocab too.
- 30 minutes per day - write codes for fun - tensorflow (gpu)
- Accomplishment :- Label 1.1

Week II (June 6 - June 12)

- Refactor export_phrases and cythonize it

- Write multi-core cython code for learn_vocab. There might be issues relating to GIL, debug and run tests.
- Write blogs about work done so far
- 30 minutes for 3 day : Tensorflow (gpu)
- Accomplishment : Label 1.2

Week III (June 13 - June 19)

- Refactor phrasegram, getitem and other remaining parts and cythonize getitem
- Implement multi-core version
- Accomplishment : Label 1.3, 1.4

Week IV (June 20 - June 23)

- Refine everything done so far. Finish multi-core version
- Debug and test. Finish.
- Accomplishment : Label 1.5

Mid Term Evaluation I (June 26 - June 30)

- Work on feedback/review (if any)
- Refine/debug (initialize Milestone 3 if completed early)
- Accomplishment : Milestone 1

Week V (June 30 - July 6)

- Create a BaseNN class in basemodel.py
- Refactor word2vec/labeledw2v, and make it run with BaseNN parent class
- Work on improving vocab building in labeledw2v
- Work on optimizing pure python part
- Accomplishment : Label 2.1

Week VI (July 7 - July 13) - Week VII (July 14 - July 20)

- Work on parallel training code in cython
- Refactor, optimize, benchmark with fastText/ vowpal wabbit
- Accomplishment : Label 2.2, Label 2.3

Week VIII (July 21 - July 25)

- Cleanup and refine the code written so far
- Work on remaining optimization tasks and create benchmark
- Accomplishment : Label 2.4

Mid Term Evaluation II (July 24 - July 28)

- Work on feedback/ review, if any.
- Refine/ debug
- Accomplishment : Label 2.5

Week IX (July 29 - Aug 6)

- Work on remaining optimization tasks
- Implement evaluation measures- precision /recall. There is some ambiguity in facebook fastText's implementation of precision/recall. Write from scratch or maybe create a wrapper function for scikit-learn evaluation measures.
- Implement prediction for multiple documents at once, and save the result in a file.
- Work on hashing trick (It's a key factor in the speed of fastText)
- Write tests
- Accomplishment : Label 2.6, Label 2.7, Label 2.8, Milestone 2

Week X (Aug 7 - Aug 13)

- Start working on GPU part with tensorFlow

Week XI (Aug 14 - Aug 20)

- Continue working on GPU
- Testing. Benchmarking
- Accomplishment : Label 3.1

Week XII (Aug 21 - Aug 29)

- Complete GPU version of labeledw2v
- Finalize codes and documents for final submission
- Work on feedback/review, if any.
- Wrap up everything

- Accomplishment : Label 3.2, Milestone 3

More About Me

I am an aspiring researcher & entrepreneur, currently pursuing Electronics & Communication Engineering (UG) at IIT Roorkee. I am passionate about Embedded Systems & IoT, and Machine Learning. I am always more interested in practical implementations of new technologies, and often look up and contribute to open source.

As an aspiring entrepreneur, I like to call myself - *Jack of all codes*. My journey from writing first arduino C code for a robotic arm, to making a python contribution in a machine learning library (gensim), is a result of curiosity and passion. I have worked on web development (backend) , Robotics, IoT, and Android development, to finally realize my passion in machine learning, and I aspire to make significant contributions in this field.

Why ME ?

I have been contributing to gensim for around two months now. I have all the prerequisites, and I've spent significant amount of time in understanding the bits and pieces of this project. Once I gain momentum, I can work for longer hours at stretch. My favourite leisure activity is also *training neurons with caffeine,* these days.

Proficiency :-

- **Programming Languages** : C/C++, Python, Cython (in order)
- **Libraries** : sklearn, tensorflow, gensim
- **IDE** : Anaconda (IPython Notebook)/ Sublime Text on Ubuntu 14.04
- **Relevant Courses** : Machine Learning (Andrew Ng/ Coursera), Deep Learning (Google/ Udacity)

I'm a voracious reader as well. Apart from reading novels, I also continuously read a lot of tech blogs, and frankly speaking, I am a self taught programmer - all

credit to the internet. I got introduced to gensim first through a tech blog at kaggle, and later while surfing through previous year gsoc organization lists. During the timeline of this project, I have no other commitments.

Acknowledgement

1. Giacomo Berardi (@giacbrd) - Thanks for replying to my emails.
2. Gensim community

References

1. GSoC 2017 project ideas
2. Bag of Tricks for Efficient Text Classification
3. word2vec Parameter Learning Explained
4. Radim Řehůřek - Faster than Google ?
5. Labeled w2v by @giacbrd PR#1153
6. fastText (Facebook Research)
7. Using GPUs | TensorFlow
8. Mathworks : Optimizing Memory Access