Notebook

March 14, 2019

1 Import library

```
In [17]: from __future__ import division, absolute_import, print_function
         # import modules & set up logging
         import gensim, logging
         import smart_open, os
         logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging
         import datetime
         import pandas as pd

         # fichier incltu dans le projet
         import save_notebook
         import utils

         import time
         from six.moves import urllib, xrange
         import tensorflow as tf
         from tensorflow.contrib.tensorboard.plugins import projector
         import numpy as np
         import collections, math, os, random, zipfile
         import glob
         import pickle
         import shutil
         import sys
```

1.0.1 Check if tensorboard use GPU

```
In [2]: from tensorflow.python.client import device_lib
        print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 2343263800986691833
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 1461770649
locality {
  bus_id: 1
  links {
  }
}
incarnation: 6883058730262070735
physical_device_desc: "device: 0, name: GeForce GTX 960M, pci bus id: 0000:01:00.0, compute cap
]
```

2 Prepare input data

```
In [3]: word_embedding_model_name = "word2vec - SKIP GRAM"
        files =  os.listdir("../wikipedia/data")
        now = str(datetime.datetime.now()).replace(" ","")
```

```
In [4]: #Create a unique file big with one wikipedia page per line
        path="../wikipedia/data/"
        if not os.path.exists('./data/wikipedia_informatic.txt'):
            with open('./data/wikipedia_informatic.txt', 'w+',encoding="utf8" ) as out_file:
                for file in files:
                    if "ipynb_checkpoints" in file:
                        continue
                    try:
                        with open(path + file, encoding="utf8") as in_file:
                            out_file.write(in_file.read().replace("\n",""))
                    except:
                        continue
        else:
            # We tokenize the file
            with open('./data/wikipedia_informatic.txt', 'r', encoding="utf8") as f:
                wiki_vocab = f.readlines()
            wiki_vocab = [x.strip() for x in wiki_vocab]
            wiki_vocab_tokenized = []
            wiki_vocab_tokenized = gensim.utils.simple_preprocess(str(wiki_vocab))
```

3 Clean data

source https://github.com/udacity/deep-learning/blob/master/embeddings/Skip-Grams-Solution.ipynb

```
In [5]: vocabulary_size = 200000
        #https://medium.com/deep-math-machine-learning-ai/chapter-9-2-nlp-code-for-word2vec-ne

        def build_dataset(vocab_tokenized):
            dictionnary_count = {}
```

```
    dictionnary_count['UNK'] = -1
    dictionnary_count.update(collections.Counter(vocab_tokenized).most_common(vocabula

    dictionary = {}
    for word in dictionnary_count:
        dictionary[word] = len(dictionary)

#     data_index = []

    unk_count = 0
    for word in vocab_tokenized:
        if word in dictionary:
            index = dictionary[word]
        else:
            index = 0  # dictionary['UNK']
            unk_count += 1

#         data_index.append(index)

    dictionnary_count["UNK"] = unk_count
    reverse_dictionary = dict(zip(dictionary.values(), dictionary.keys()))

    return dictionnary_count, dictionary, reverse_dictionary


dictionnary_count, dictionary, reverse_dictionary = build_dataset(wiki_vocab_tokenized

print('most common words (+UNK):', utils.take(10, dictionnary_count.items()))
# print('sample data:', data_index[:10], [reverse_dictionary[i] for i in data_index[:1
```

```
most common words (+UNK): [('UNK', 0), ('de', 531786), ('la', 237117), ('le', 206228), ('et',
```

3.1 Subsampling

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where t is a threshold parameter and $f(w_i)$ is the frequency of word w_i in the total dataset.

In [6]:
```
#subsampling data
def subsampling_data(vocab_tokenized, reverse_dictionary, dictionary_count, threshold)
    np.random.seed(123)

#     training_dictionary_subsample = dictionary_subsample
    training_dictionary_subsample = []
    size_of_corpus = len(vocab_tokenized)
```

```
            for index in reverse_dictionary:
                word = reverse_dictionary[index]
                if word == "UNK":
        #             dictionary_subsample[word] = -1
                    continue

                count_of_word = dictionnary_count[word]
                frequence_of_word = count_of_word / size_of_corpus
                probability = 1 - ( np.sqrt(threshold / frequence_of_word))

                if probability < np.random.random():
                    training_dictionary_subsample.append(index)

            return training_dictionary_subsample
```

```
In [7]: trainded_data = subsampling_data(wiki_vocab_tokenized, reverse_dictionary ,dictionnary_
```

4 Definition of Model

```
In [8]: def get_target(words, index, window_size=5):
            R = np.random.randint(1, window_size+1)
            start = index - R if (index - R) > 0 else 0
            stop = index + R
            target_words = set(words[start:index] + words[index+1:stop+1])
            return list(target_words)
```

```
In [9]: def create_placeholders():
            x = tf.placeholder(tf.int64, [None], name="x")
            y = tf.placeholder(tf.int64, [None, None], name="y")
            return x, y
```

```
In [10]: def get_batches(words, batch_size, window_size=5):
             n_batches = len(words)//batch_size

             # only full batches
             words = words[:n_batches*batch_size]

             for index in range(0, len(words), batch_size):
                 x, y = [], []
                 step_batch = index+batch_size
                 batch = words[index:step_batch]
                 for i in range(len(batch)):
                     batch_x = batch[i]
                     batch_y = get_target(batch, i, window_size)
                     y.extend(batch_y)
                     x.extend([batch_x]*len(batch_y))
                 yield x, y
```

```
In [11]: def get_embed(x, embedding_size, vocabulary_size):
             embedding_matrice = tf.Variable(tf.random_uniform([vocabulary_size,embedding_size
             embed = tf.nn.embedding_lookup(embedding_matrice, x)

             return embedding_matrice, embed
```

```
In [12]: def negative_sampling(vocab_size, embed, y, embedding_size, n_sampled = 100):
         # Number of negative y to sample

             softmax_w = tf.Variable(tf.truncated_normal((vocab_size, embedding_size), stddev=
             softmax_b = tf.Variable(tf.zeros(vocab_size))

             # Calculate the loss using negative sampling

             loss = tf.nn.sampled_softmax_loss(softmax_w, softmax_b,
                                               y, embed,
                                               n_sampled, vocab_size)


             cost = tf.reduce_mean(loss)
             optimizer = tf.train.AdamOptimizer().minimize(cost)
             return loss, cost, optimizer
```

```
In [13]: def get_result(embedding):
             norm = tf.sqrt(tf.reduce_sum(tf.square(embedding), 1, keepdims=True))
             normalized_embedding = embedding / norm
             return normalized_embedding
```

```
In [14]: def print_result(normalized_embedding, id_word):

                     ## From Thushan Ganegedara's implementation
             valid_size = 10 # Random set of words to evaluate similarity on.
             valid_window = 100
             # pick 8 samples from (0,100) and (1000,1100) each ranges. lower id implies more 
             id_word = tf.constant(id_word)

         #     np_valid_examples = np.array(id_word , valid_size//2)

         #     np_valid_examples = np.append(np_valid_examples, random.sample(range(1000,1000+

             #valid_examples = tf.convert_to_tensor(np_valid_examples)
             #We use the cosine distance:
             valid_dataset = tf.Variable([id_word , valid_size//2])
         #     valid_dataset = tf.constant(np_valid_examples)

             valid_embedding = tf.nn.embedding_lookup(normalized_embedding, valid_dataset)
             similarity = tf.matmul(valid_embedding, tf.transpose(normalized_embedding))
             return similarity, valid_dataset
```

```
In [15]: def is_early_stopping(array_average_loss, max_steps_without_decrease):
             array_last_average_loss = array_average_loss[-max_steps_without_decrease:len(arra
             for element in range(0, max_steps_without_decrease-1):
                 # if one loss at step t is bigger than loss as t+1 so it is converging
                 if array_last_average_loss[element] > array_last_average_loss[element +1]:
                     return False
             return True

In [16]: def model_initializer(x, embedding_size, vocabulary_size, id_word):

             train_graph = tf.get_default_graph()

             #     on declare
             with train_graph.as_default():
                 x, y = create_placeholders()
                 embedding, x_embed = get_embed(x, embedding_size, vocabulary_size)
                 loss, cost, optimizer = negative_sampling(vocabulary_size, x_embed, y, embeddi
                 normalized_embedding = get_result(embedding)
                 similarity, valid_examples = print_result(normalized_embedding, id_word)
                 tf.summary.scalar("minibatch_cost", cost)
                 merge = tf.summary.merge_all()

                 saver = tf.train.Saver()
             return train_graph, x, y, x_embed, cost, optimizer, normalized_embedding, similari

In [29]: def train(x_train, reverse_dictionary, epochs, batch_size, window_size, embedding_siz
             # initialization of graph
             train_graph, x, y, x_embed, cost, optimizer, normalized_embedding, similarity, val

             #init var
             date_started = time.time()
             epoch_saver = tf.train.Saver(max_to_keep=3)  # keep 3 last iterations
             epoch_initializer = 0
             minibatch_cost_sum = 0

             #we store all result for early_stopping
             array_average_loss = []

             with tf.Session(graph=train_graph) as sess:

                 iteration = 1
                 sess.run(tf.global_variables_initializer())
                 date_started = time.time()

                  #If true we load the last checkpoint if exist
                 if restore_last_session:
                     try:
                             FIND_CHKP = False
                             LOG_DIR_SUBFOLDER = LOG_DIR.rsplit('/', 1)[0]+"/"
                             LOG_DIR_LIST = sorted(glob.glob(os.path.join(LOG_DIR_SUBFOLDER, '
                             for folder in LOG_DIR_LIST:
                                 if os.path.exists(folder + "save_model"):
                                     ckpt = tf.train.get_checkpoint_state(folder +"/save_model/

                                     #check if the checkpoint is the same architecture
                                     if str(embedding_size) + "-epoch_saver" in str(ckpt):
                                         LOG_DIR = folder
                                         FIND_CHKP = True
                                         break
                             if FIND_CHKP is False:
                                 sys.exit("no checkpoint to load")
                             iteration = int(str(ckpt).rsplit("epoch_saver-i")[1].rsplit("i-")
                             print("checkpoint load from directory " + LOG_DIR + " with archite

                             epoch_saver.restore(sess, ckpt.model_checkpoint_path)
                             epoch_initializer = int(ckpt.model_checkpoint_path.rsplit('i--',1
                             epochs = epochs + epoch_initializer

                             with open (LOG_DIR +"/save_model/array_average_loss.save", 'rb') a
                                 array_average_loss = pickle.load(file)

                     except Exception as e: print(str(" can't load checkpoint => " + str(e)))


                 #for tensorboard graph
                 writer = tf.summary.FileWriter(LOG_DIR, sess.graph, filename_suffix=str(date_s

                 for e in range(epoch_initializer, epochs+1):

                     print("epoque ===========================================================

                     n_batches = len(x_train)//batch_size
                     minibatch_data = get_batches(x_train, batch_size, window_size)

                     minibatch_iteration = 0
                     for minibatch_X, minibatch_Y in minibatch_data:

                         minibatch_cost, _, merge_minibatch= sess.run([cost, optimizer, merge]

                         minibatch_cost_sum = minibatch_cost_sum + minibatch_cost
                         average_loss = minibatch_cost_sum / iteration
                         array_average_loss.append(average_loss)

                         average_loss = tf.Summary(value=[tf.Summary.Value(tag="average_loss",
                         writer.add_summary(average_loss, iteration)


                         #we stop learning it the loss does not decrease
```

```
                if iteration > max_steps_without_decrease:
                    stop = is_early_stopping(array_average_loss, max_steps_without_de
                    if stop:
                        print("stop because early stopping")
                        break

                percent_in_epoch =  minibatch_iteration * 100 / utils.round_down(n_ba

                if percent_in_epoch  % 25 == 0:
                    print("average loss  : "  + str(average_loss))
                    print("iteration => " + str(iteration) + " and " + str(percent_in
                    sim, v_examples = sess.run([similarity,valid_examples], feed_dict


                    valid_word = reverse_dictionary[v_examples[0]]
                    top_k = 5 # number of nearest neighbors
                    nearest = (-sim[0, :]).argsort()[1:top_k+1]
                    log = 'Nearest to %s:' % valid_word
                    for k in range(top_k):
                        close_word = reverse_dictionary[nearest[k]]
                        log = '%s %s,' % (log, close_word)
                    print(log)

                    print("------------------------------------------------------------
                iteration += 1
                minibatch_iteration += + 1


            #we save model at each epoch
            if not os.path.exists(LOG_DIR +"/save_model/"):
                os.makedirs(LOG_DIR +"/save_model")
            epoch_saver.save(sess,  LOG_DIR +"/save_model/" + str(embedding_size) + "

            with open(LOG_DIR +"/save_model/array_average_loss.save", 'wb+') as file:
                pickle.dump(array_average_loss, file)


        save_path = saver.save(sess,os.path.join(LOG_DIR ,'model'+ str(embedding_size
        embed_mat = sess.run(normalized_embedding, feed_dict={x: np.array(trainded_da

        return embed_mat, epochs, LOG_DIR
```

In [31]:

```
model_name = utils.get_model_name(word_embedding_model_name)
# LOG_DIR  = './tensorboard-graph/'+ model_name
tf.reset_default_graph()
date_before_learning = datetime.datetime.now()
```

In [32]:

```
id_word = dictionary["redhat"]
word_embedding, epochs, LOG_DIR = train(trainded_data, reverse_dictionary, 15, 16, 5,
         time_training = datetime.datetime.now() - date_before_learning

2019-03-14 18:51:10,698 : INFO : Restoring parameters from ./tensorboard-graph/word2vec - SKIP


checkpoint load from directory ./tensorboard-graph/word2vec - SKIP GRAM\03-14-18-36\ with archi
epoque =======================================================================================
average loss  : value {
  tag: "average_loss"
  simple_value: 0.0003204015374649316
}

iteration => 33928 and 0.0 % for this epoch at time 2019-03-1418:51:11.149501
Nearest to redhat: simenon, afe, librestheodor, culturellela, concession,
-------------------------------------------------------------------------------
average loss  : value {
  tag: "average_loss"
  simple_value: 0.35378092527389526
}

iteration => 36753 and 25.0 % for this epoch at time 2019-03-1418:51:31.142871
Nearest to redhat: afe, simenon, rhl, peint, culturellela,
-------------------------------------------------------------------------------
average loss  : value {
  tag: "average_loss"
  simple_value: 0.5552752017974854
}

iteration => 39578 and 50.0 % for this epoch at time 2019-03-1418:51:52.430693
Nearest to redhat: afe, simenon, rhl, peint, culturellela,
-------------------------------------------------------------------------------
average loss  : value {
  tag: "average_loss"
  simple_value: 0.6949954032897949
}

iteration => 42403 and 75.0 % for this epoch at time 2019-03-1418:52:11.848346
Nearest to redhat: culturellela, afe, rhl, peint, professoral,
-------------------------------------------------------------------------------
average loss  : value {
  tag: "average_loss"
  simple_value: 0.797066867351532
}

iteration => 45228 and 100.0 % for this epoch at time 2019-03-1418:52:33.616246
Nearest to redhat: afe, culturellela, librestheodor, rhl, peint,
-------------------------------------------------------------------------------
```

```
epoque ========================================================================================
average loss  : value {
  tag: "average_loss"
  simple_value: 0.7975736260414124
}

iteration => 45237 and 0.0 % for this epoch at time 2019-03-1418:52:34.407815
Nearest to redhat: afe, culturellela, librestheodor, rhl, peint,
--------------------------------------------------------------------------------
average loss  : value {
  tag: "average_loss"
  simple_value: 1.0240849256515503
}

iteration => 48062 and 25.0 % for this epoch at time 2019-03-1418:53:12.365607
Nearest to redhat: afe, culturellela, peint, rhl, professoral,
--------------------------------------------------------------------------------
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
epoque ========================================================================================
stop because early stopping
```

4.0.1 Create index label for tensorboard

```
In [ ]: f = open(os.path.join(LOG_DIR, "metadata.tsv"), 'w+', encoding="utf8")
        i = 0
        for idx in trainded_data:
            f.write(reverse_dictionary[idx] + '\n')
            if i ==  99999:
                break
            i = i + 1

        f.close()
```

4.1 Visu

```
In [ ]: train_graph = tf.get_default_graph()

        with tf.Session(graph=train_graph) as sess:

            #prepare for embedding on tensorboard
            embedding_writer = tf.summary.FileWriter(LOG_DIR +'/projector', sess.graph)
            config = projector.ProjectorConfig()
            embedding_conf = config.embeddings.add()
            embedding_conf.tensor_name = 'embedding'
            embedding_conf.metadata_path = os.path.join(LOG_DIR+'/projector' ,  'metadata.tsv'
            projector.visualize_embeddings(embedding_writer, config)
```

5 Save

```
In [ ]: name_notebook_exported = save_notebook.save_notebook("word2vec_with_tf.ipynb")
        utils.write_result(word_embedding_model_name, time_training, name_notebook_exported, e
```

```
In [ ]:
```