DNNs

August 18, 2019

[1]:
```
from models import get_model
import argparse
import pickle
import string
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import roc_auc_score
import preprocessor as p
from collections import Counter
import os
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.contrib import learn
from tflearn.data_utils import to_categorical, pad_sequences
from scipy import stats
import tflearn
import json
```

```
WARNING: Logging before flag parsing goes to stderr.
W0818 18:03:32.183784 139815936833344 deprecation_wrapper.py:119] From
/home/sujeendra/miniconda3/envs/tf/lib/python3.7/site-
packages/tflearn/helpers/summarizer.py:9: The name tf.summary.merge is
deprecated. Please use tf.compat.v1.summary.merge instead.

W0818 18:03:32.185410 139815936833344 deprecation_wrapper.py:119] From
/home/sujeendra/miniconda3/envs/tf/lib/python3.7/site-
packages/tflearn/helpers/trainer.py:25: The name tf.summary.FileWriter is
deprecated. Please use tf.compat.v1.summary.FileWriter instead.

W0818 18:03:32.194820 139815936833344 deprecation_wrapper.py:119] From
/home/sujeendra/miniconda3/envs/tf/lib/python3.7/site-
packages/tflearn/collections.py:13: The name tf.GraphKeys is deprecated. Please
use tf.compat.v1.GraphKeys instead.

W0818 18:03:32.200190 139815936833344 deprecation_wrapper.py:119] From
/home/sujeendra/miniconda3/envs/tf/lib/python3.7/site-
packages/tflearn/config.py:123: The name tf.get_collection is deprecated. Please
```

```
use tf.compat.v1.get_collection instead.

W0818 18:03:32.209696 139815936833344 deprecation_wrapper.py:119] From
/home/sujeendra/miniconda3/envs/tf/lib/python3.7/site-
packages/tflearn/config.py:129: The name tf.add_to_collection is deprecated.
Please use tf.compat.v1.add_to_collection instead.

W0818 18:03:32.210695 139815936833344 deprecation_wrapper.py:119] From
/home/sujeendra/miniconda3/envs/tf/lib/python3.7/site-
packages/tflearn/config.py:131: The name tf.assign is deprecated. Please use
tf.compat.v1.assign instead.

Using Theano backend.
```

[2]:
```
def load_data(filename):
    print("Loading data from file: " + filename)
    data = pickle.load(open(filename, 'rb'))
    x_text = []
    labels = []
    for i in range(len(data)):
        if(HASH_REMOVE):
            x_text.append(p.tokenize((data[i]['text']).encode('utf-8')))
        else:
            x_text.append(data[i]['text'])
        labels.append(data[i]['label'])
    return x_text,labels

def get_filename(dataset):
    global NUM_CLASSES, HASH_REMOVE
    if(dataset=="twitter"):
        NUM_CLASSES = 3
        HASH_REMOVE = True
        filename = "/home/sujeendra/Desktop/data/data/twitter_data.pkl"
    elif(dataset=="formspring"):
        NUM_CLASSES = 2
        filename = "/home/sujeendra/Desktop/data/data/formspring_data.pkl"
    elif(dataset=="wiki"):
        NUM_CLASSES = 2
        filename = "/home/sujeendra/Desktop/data/data/wiki_data.pkl"
    return filename
```

[3]:
```
def get_embedding_weights(filename, sep):
    embed_dict = {}
    file = open(filename,'r')
    for line in file.readlines():
        row = line.strip().split(sep)
        embed_dict[row[0]] = row[1:]
    print('Loaded from file: ' + str(filename))
```

```
    file.close()
    return embed_dict

def map_embedding_weights(embed, vocab, embed_size):
    vocab_size = len(vocab)
    embeddingWeights = np.zeros((vocab_size , embed_size))
    n = 0
    words_missed = []
    for k, v in vocab.iteritems():
        try:
            embeddingWeights[v] = embed[k]
        except:
            n += 1
            words_missed.append(k)
            pass
    print("%d embedding missed"%n, " of " , vocab_size)
    return embeddingWeights

def get_embeddings_dict(vector_type, emb_dim):
    if vector_type == 'sswe':
        emb_dim==50
        sep = '\t'
        vector_file = 'word_vectors/sswe-u.txt'
    elif vector_type =="glove":
        sep = ' '
        if data == "wiki":
            vector_file = 'word_vectors/glove.6B.' + str(emb_dim) + 'd.txt'
        else:
            vector_file = 'word_vectors/glove.twitter.27B.' + str(emb_dim) + 'd.
 ↪txt'
    else:
        print ("ERROR: Please specify a correst model or SSWE cannot be loaded␣
 ↪with embed size of: " + str(emb_dim))
        return None

    embed = get_embedding_weights(vector_file, sep)
    return embed
```

[4]:
```
def evaluate_model(model, testX, testY):
    temp = model.predict(testX)
    y_pred  = np.argmax(temp, 1)
    y_true = np.argmax(testY, 1)
    precision = metrics.precision_score(y_true, y_pred, average=None)
    recall = metrics.recall_score(y_true, y_pred, average=None)
    f1_score = metrics.f1_score(y_true, y_pred, average=None)
    print("Precision: " + str(precision) + "\n")
    print("Recall: " + str(recall) + "\n")
```

```
    print("f1_score: " + str(f1_score) + "\n")
    print(confusion_matrix(y_true, y_pred))
    print(":: Classification Report")
    print(classification_report(y_true, y_pred))
    return precision, recall, f1_score
```

[5]:
```
def dump_learned_embedding(data, model_type, vector_type, embed_size, embed,␣
↪vocab_processor):
    vocab = vocab_processor.vocabulary_._mapping
    vocab_size = len(vocab)
    embedDict = {}
    n = 0
    words_missed = []
    for k, v in vocab.iteritems():
        try:
            embeddingDict[v] = embed[k]
        except:
            n += 1
            words_missed.append(k)
            pass
    print("%d embedding missed"%n, " of " , vocab_size)

    filename = output_folder_name + data + "_" + model_type + "_" + vector_type␣
↪+ "_" + embed_size + ".pkl"
    with open(filename, 'wb') as handle:
        pickle.dump(embedDict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[6]:
```
def get_train_test(data, x_text, labels):

    X_train, X_test, Y_train, Y_test = train_test_split( x_text, labels,␣
↪random_state=42, test_size=0.10)

    post_length = np.array([len(x.split(" ")) for x in x_text])
    if(data != "twitter"):
        max_document_length = int(np.percentile(post_length, 95))
    else:
        max_document_length = max(post_length)
    print("Document length : " + str(max_document_length))

    vocab_processor = learn.preprocessing.
↪VocabularyProcessor(max_document_length, MAX_FEATURES)
    vocab_processor = vocab_processor.fit(x_text)

    trainX = np.array(list(vocab_processor.transform(X_train)))
    testX = np.array(list(vocab_processor.transform(X_test)))

    trainY = np.asarray(Y_train)
    testY = np.asarray(Y_test)
```

```
    trainX = pad_sequences(trainX, maxlen=max_document_length, value=0.)
    testX = pad_sequences(testX, maxlen=max_document_length, value=0.)

    trainY = to_categorical(trainY, nb_classes=NUM_CLASSES)
    testY = to_categorical(testY, nb_classes=NUM_CLASSES)

    data_dict = {
        "data": data,
        "trainX" : trainX,
        "trainY" : trainY,
        "testX" : testX,
        "testY" : testY,
        "vocab_processor" : vocab_processor
    }

    return data_dict
```

[7]:
```
def return_data(data_dict):
    return data_dict["data"], data_dict["trainX"], data_dict["trainY"],␣
↪data_dict["testX"], data_dict["testY"], data_dict["vocab_processor"]
```

[8]:
```
def shuffle_weights(model, weights=None):
    """Randomly permute the weights in `model`, or the given `weights`.
    This is a fast approximation of re-initializing the weights of a model.
    Assumes weights are distributed independently of the dimensions of the␣
↪weight tensors
      (i.e., the weights have the same distribution along each dimension).
    :param Model model: Modify the weights of the given model.
    :param list(ndarray) weights: The model's weights will be replaced by a␣
↪random permutation of these weights.
      If `None`, permute the model's current weights.
    """
    if weights is None:
        weights = model.get_weights()
    weights = [np.random.permutation(w.flat).reshape(w.shape) for w in weights]
    # Faster, but less random: only permutes along the first dimension
    # weights = [np.random.permutation(w) for w in weights]
    model.set_weights(weights)
```

[9]:
```
def train(data_dict, model_type, vector_type, embed_size, dump_embeddings=True):

    data, trainX, trainY, testX, testY, vocab_processor = return_data(data_dict)

    vocab_size = len(vocab_processor.vocabulary_)
    print("Vocabulary Size: {:d}".format(vocab_size))
    vocab = vocab_processor.vocabulary_._mapping
```

```
   print("Running Model: " + model_type + " with word vector initiliazed with "␣
↪+ vector_type + " word vectors.")
   model = get_model(model_type, trainX.shape[1], vocab_size, embed_size,␣
↪NUM_CLASSES, LEARN_RATE)

   initial_weights = model.get_weights()
   shuffle_weights(model, initial_weights)

   if(model_type == 'cnn'):
       if(vector_type!="random"):
           print("Word vectors used: " + vector_type)
           embeddingWeights = tflearn.
↪get_layer_variables_by_name('EmbeddingLayer')[0]
           model.set_weights(embeddingWeights,␣
↪map_embedding_weights(get_embeddings_dict(vector_type, embed_size), vocab,␣
↪embed_size))
           model.fit(trainX, trainY, n_epoch = EPOCHS, shuffle=True,␣
↪show_metric=True, batch_size=BATCH_SIZE)
       else:
           model.fit(trainX, trainY, n_epoch = EPOCHS, shuffle=True,␣
↪show_metric=True, batch_size=BATCH_SIZE)
   else:
       if(vector_type!="random"):
           print("Word vectors used: " + vector_type)
           model.layers[0].
↪set_weights([map_embedding_weights(get_embeddings_dict(vector_type,␣
↪embed_size), vocab, embed_size)])
           model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
↪batch_size=BATCH_SIZE,
                 verbose=1)
       else:
           model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
↪batch_size=BATCH_SIZE,
                 verbose=1)

   if (dump_embeddings==True):
       if(model_type == 'cnn'):
           embeddingWeights = tflearn.
↪get_layer_variables_by_name('EmbeddingLayer')[0]
       else:
           embed = model.layers[0].get_weights()[0]

       embed_filename = output_folder_name + data + "_" + model_type + "_" +␣
↪vector_type + "_" + str(embed_size) + ".pkl"
       embed.dump(embed_filename)
```

```
        vocab_filename = output_folder_name + data + "_" + model_type + "_" +␣
↪vector_type + "_" + str(embed_size) + "_dict.json"
        reverse_vocab_filename = output_folder_name + data + "_" + model_type +␣
↪"_" + vector_type + "_" + str(embed_size) + "_reversedict.json"

        with open(vocab_filename, 'w') as fp:
            json.dump(vocab_processor.vocabulary_._mapping, fp)
        with open(reverse_vocab_filename, 'w') as fp:
            json.dump(vocab_processor.vocabulary_._reverse_mapping, fp)

    return  evaluate_model(model, testX, testY)
```

[10]:
```
def print_scores(precision_scores, recall_scores, f1_scores):
    for i in range(NUM_CLASSES):
        print("\nPrecision Class %d (avg): %0.3f (+/- %0.3f)" % (i,␣
↪precision_scores[:, i].mean(), precision_scores[:, i].std() * 2))
        print( "\nRecall Class %d (avg): %0.3f (+/- %0.3f)" % (i, recall_scores[:
↪, i].mean(), recall_scores[:, i].std() * 2))
        print( "\nF1 score Class %d (avg): %0.3f (+/- %0.3f)" % (i, f1_scores[:,␣
↪i].mean(), f1_scores[:, i].std() * 2))
```

[11]:
```
def get_data(data, oversampling_rate):

    x_text, labels = load_data(get_filename(data))

    if(data=="twitter"):
        NUM_CLASSES = 3
        dict1 = {'racism':2,'sexism':1,'none':0}
        labels = [dict1[b] for b in labels]

        racism = [i for i in range(len(labels)) if labels[i]==2]
        sexism = [i for i in range(len(labels)) if labels[i]==1]
        x_text = x_text + [x_text[x] for x in racism]*(oversampling_rate-1)+␣
↪[x_text[x] for x in sexism]*(oversampling_rate-1)
        labels = labels + [2 for i in range(len(racism))]*(oversampling_rate-1)␣
↪+ [1 for i in range(len(sexism))]*(oversampling_rate-1)

    else:
        NUM_CLASSES = 2
        bully = [i for i in range(len(labels)) if labels[i]==1]
        x_text = x_text + [x_text[x] for x in bully]*(oversampling_rate-1)
        labels = list(labels) + [1 for i in␣
↪range(len(bully))]*(oversampling_rate-1)

    print("Counter after oversampling")
    from collections import Counter
    print(Counter(labels))
```

```
    filter_data = []
    for text in x_text:
        filter_data.append("".join(l for l in text if l not in string.
 ↪punctuation))

    return x_text, labels
```

[15]:
```
models = [ 'cnn', 'lstm', 'blstm', 'blstm_attention']
word_vectors = ["random", "glove" ,"sswe"]
EPOCHS = 10
BATCH_SIZE = 128
MAX_FEATURES = 2
NUM_CLASSES = None
DROPOUT = 0.25
LEARN_RATE = 0.01
HASH_REMOVE = None
output_folder_name = "/home/sujeendra/Desktop/results/"
```

[13]:
```
def run_model(data, oversampling_rate, model_type, vector_type, embed_size):
    x_text, labels = get_data(data, oversampling_rate)
    data_dict = get_train_test(data,  x_text, labels)
    precision, recall, f1_score = train(data_dict, model_type, vector_type,␣
 ↪embed_size)
```

[16]:
```
data = "formspring"
model_type = "blstm_attention"
vector_type = "random"

'''for embed_size in [25, 50, 100, 200]:
    run_model(data, 3, model_type, vector_type, embed_size)'''
run_model(data, 3, model_type, vector_type, 50)
```

```
Loading data from file: /home/sujeendra/Desktop/data/data/formspring_data.pkl
Counter after oversampling
Counter({0: 11997, 1: 2328})
Document length : 62
Vocabulary Size: 7190
Running Model: blstm_attention with word vector initiliazed with random word
vectors.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 62, 50)            359500
_________________________________________________________________
dropout_3 (Dropout)          (None, 62, 50)            0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 62, 100)           40400
_________________________________________________________________
```

```
att_layer_2 (AttLayer)       (None, 100)               100
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 202
=================================================================
Total params: 400,202
Trainable params: 400,202
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
12892/12892 [==============================] - 42s 3ms/step - loss: 0.4563 -
acc: 0.8333
Epoch 2/10
12892/12892 [==============================] - 43s 3ms/step - loss: 0.2579 -
acc: 0.8999
Epoch 3/10
12892/12892 [==============================] - 43s 3ms/step - loss: 0.1372 -
acc: 0.9535
Epoch 4/10
12892/12892 [==============================] - 43s 3ms/step - loss: 0.0827 -
acc: 0.9753
Epoch 5/10
12892/12892 [==============================] - 44s 3ms/step - loss: 0.0593 -
acc: 0.9829
Epoch 6/10
12892/12892 [==============================] - 43s 3ms/step - loss: 0.0363 -
acc: 0.9894
Epoch 7/10
12892/12892 [==============================] - 44s 3ms/step - loss: 0.0441 -
acc: 0.9872
Epoch 8/10
12892/12892 [==============================] - 44s 3ms/step - loss: 0.0274 -
acc: 0.9924
Epoch 9/10
12892/12892 [==============================] - 44s 3ms/step - loss: 0.0258 -
acc: 0.9932
Epoch 10/10
12892/12892 [==============================] - 44s 3ms/step - loss: 0.0209 -
acc: 0.9939
Precision: [0.99663866 0.90946502]

Recall: [0.98178808 0.98222222]

f1_score: [0.98915763 0.94444444]

[[1186   22]
 [   4  221]]
```

```
:: Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1208
           1       0.91      0.98      0.94       225

    accuracy                           0.98      1433
   macro avg       0.95      0.98      0.97      1433
weighted avg       0.98      0.98      0.98      1433
```

[ ]: