Assignment-3.1

June 13, 2020

1 Assignment 3.1. Sequence Classification

2 Task: Aspect-level Sentiment Classification(10pt)

Reading material: - [1] R. He, WS. Lee & D. Dahlmeier. Exploiting document knowledge for aspect-level sentiment classification. 2018. https://arxiv.org/abs/1806.04346.

Build an attention-based aspect-level sentiment classification model with biLSTM. Your model shall include:

- BiLSTM network that learns sentence representation from input sequences.
- Attention network that assigns attention score over a sequence of biLSTM hidden states based on aspect terms representation.
- Fully connected network that predicts sentiment label, given the representation weighted by the attention score.

Requirements:

- You shall train your model bsaed on transferring learning. That is, you need first train your model on documnet-level examples. Then the learned weights will be used to initialize aspect-level model and fine tune it on aspect-level examples.
- You shall use the alignment score function in attention network as following expression:

$$f_{score}(h,t) = tanh(h^T W_a t)$$

- You shall evaluate the trained model on the provided test set and show the accuracy on test set.

[1]:
```
import os
import sys
import codecs
import operator
import numpy as np
import re
from time import time
```

[2]:
```
import _pickle as cPickle
```

3 Load Data

[3]:
```
def read_pickle(data_path, file_name):

    f = open(os.path.join(data_path, file_name), 'rb')
    read_file = cPickle.load(f)
    f.close()

    return read_file

def save_pickle(data_path, file_name, data):

    f = open(os.path.join(data_path, file_name), 'wb')
    cPickle.dump(data, f)
    print(" file saved to: %s"%(os.path.join(data_path, file_name)))
    f.close()
```

[4]:
```
aspect_path = 'data/aspect-level'


vocab = read_pickle(aspect_path, 'all_vocab.pkl')

train_x = read_pickle(aspect_path, 'train_x.pkl')
train_y = read_pickle(aspect_path, 'train_y.pkl')
dev_x = read_pickle(aspect_path, 'dev_x.pkl')
dev_y = read_pickle(aspect_path, 'dev_y.pkl')
test_x = read_pickle(aspect_path, 'test_x.pkl')
test_y = read_pickle(aspect_path, 'test_y.pkl')

train_aspect = read_pickle(aspect_path, 'train_aspect.pkl')
dev_aspect = read_pickle(aspect_path, 'dev_aspect.pkl')
test_aspect = read_pickle(aspect_path, 'test_aspect.pkl')


pretrain_data = read_pickle(aspect_path, 'pretrain_data.pkl')
pretrain_label = read_pickle(aspect_path, 'pretrain_label.pkl')
```

[5]:
```
class Dataiterator_doc():
    '''
      1) Iteration over minibatches using next(); call reset() between epochs␣
 ↪to randomly shuffle the data
      2) Access to the entire dataset using all()
    '''

    def __init__(self, X, y, seq_length=32, decoder_dim=300, batch_size=32):
        self.X = X
        self.y = y
```

```
        self.num_data = len(X) # total number of examples
        self.batch_size = batch_size # batch size
        self.reset() # initial: shuffling examples and set index to 0

    def __iter__(self): # iterates data
        return self


    def reset(self): # initials
        self.idx = 0
        self.order = np.random.permutation(self.num_data) # shuffling examples␣
↪by providing randomized ids

    def __next__(self): # return model inputs - outputs per batch
        X_ids = [] # hold ids per batch
        while len(X_ids) < self.batch_size:
            X_id = self.order[self.idx] # copy random id from initial shuffling
            X_ids.append(X_id)
            self.idx += 1 #
            if self.idx >= self.num_data: # exception if all examples of data␣
↪have been seen (iterated)
                self.reset()
                raise StopIteration()

        batch_X = self.X[np.array(X_ids)] # X values (encoder input) per batch
        batch_y = self.y[np.array(X_ids)] # y_in values (decoder input) per␣
↪batch
        return batch_X, batch_y


    def all(self): # return all data examples
        return self.X, self.y
class Dataiterator_aspect():
    '''
      1) Iteration over minibatches using next(); call reset() between epochs␣
↪to randomly shuffle the data
      2) Access to the entire dataset using all()
    '''

    def __init__(self, aspect_data, seq_length=32, decoder_dim=300,␣
↪batch_size=32):

        len_aspect_data = len(aspect_data[0])
        #self.len_doc_data = len(doc_data[0])

        self.X_aspect = aspect_data[0]
        self.y_aspect = aspect_data[1]
```

```
        self.aspect_terms = aspect_data[2]
        self.num_data = len_aspect_data
        self.batch_size = batch_size # batch size
        self.reset() # initial: shuffling examples and set index to 0

    def __iter__(self): # iterates data
        return self


    def reset(self): # initials
        self.idx = 0
        self.order = np.random.permutation(self.num_data) # shuffling examples␣
↪by providing randomized ids

    def __next__(self): # return model inputs - outputs per batch

        X_ids = [] # hold ids per batch
        while len(X_ids) < self.batch_size:
            X_id = self.order[self.idx] # copy random id from initial shuffling
            X_ids.append(X_id)
            self.idx += 1 #
            if self.idx >= self.num_data: # exception if all examples of data␣
↪have been seen (iterated)
                self.reset()
                raise StopIteration()

        batch_X_aspect = self.X_aspect[np.array(X_ids)] # X values (encoder␣
↪input) per batch
        batch_y_aspect = self.y_aspect[np.array(X_ids)] # y_in values (decoder␣
↪input) per batch
        batch_aspect_terms = self.aspect_terms[np.array(X_ids)]

        return batch_X_aspect, batch_y_aspect, batch_aspect_terms


    def all(self): # return all data examples
        return self.X_aspect, self.y_aspect, self.aspect_terms
```

[6]:
```
from tensorflow import keras
from keras.models import Model
from keras.layers import Input, Embedding, Dense, Lambda, Dropout,␣
↪LSTM,Bidirectional, Flatten
from keras.layers import Reshape, Activation, RepeatVector, concatenate,␣
↪Concatenate, Dot, Multiply
import keras.backend as K
from keras.engine.topology import Layer
```

```python
from keras import initializers
from keras import regularizers
from keras import constraints
```

```
Using TensorFlow backend.
```

[7]:
```python
overal_maxlen = 82
overal_maxlen_aspect = 7
```

4 Define Attention Network Layer

- Define class for Attention Layer
- You need to finish the code for calculating the attention weights

[8]:
```python
import tensorflow as tf

class Attention(Layer):
    def __init__(self,  **kwargs):
        """
        Keras Layer that implements an Content Attention mechanism.
        Supports Masking.
        """

        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')

        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert type(input_shape) == list

        self.steps = input_shape[0][1]

        self.W = self.add_weight(shape=(input_shape[0][-1], input_shape[1][-1]),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),)

        self.built = True

    def compute_mask(self, input_tensor, mask=None):
        assert type(input_tensor) == list
        assert type(mask) == list
        return None

    def call(self, input_tensor, mask=None):
        x = input_tensor[0]
        aspect = input_tensor[1]
```

```
        mask = mask[0]
        ###YOUR CODE HERE###

        # Masking
        masked_x = x * tf.expand_dims(tf.cast(mask, "float"), -1)

        dotter = Dot(axes=-1)
        beta = tf.tanh(dotter([masked_x @ self.W, aspect]))
        alpha = tf.exp(beta) / tf.reduce_sum(tf.exp(beta), axis=1,␣
↪keepdims=True)

        return alpha


    def compute_output_shape(self, input_shape):
        return (input_shape[0][0], input_shape[0][1])
```

[9]:
```
class Average(Layer):

    def __init__(self, mask_zero=True, **kwargs):
        self.mask_zero = mask_zero
        self.supports_masking = True
        super(Average, self).__init__(**kwargs)

    def call(self, x,mask=None):
        if self.mask_zero:
            mask = K.cast(mask, K.floatx())
            mask = K.expand_dims(mask)
            x = x * mask
            return K.sum(x, axis=1) / (K.sum(mask, axis=1) + K.epsilon())
        else:
            return K.mean(x, axis=1)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[-1])

    def compute_mask(self, x, mask):
        return None
```

5 Establish computation Grah for model

- Input tensors
- Shared WordEmbedding layer
- Attention network layer

- Shared BiLSTM layer

- Shared fully connected layer(prediction layer)

[10]:
```
dropout = 0.5
recurrent_dropout = 0.1
vocab_size = len(vocab)
num_outputs = 3 # labels
```

5.1 Input tensors

[11]:
```
#YOUR CODE HERE ##### Inputs #####
aspect_inputs = Input(shape=(overal_maxlen_aspect,), dtype="int32",␣
↪name="aspect")
sentence_inputs = Input(shape=(overal_maxlen,), dtype="int32", name="sentence")
pretrain_inputs = Input(shape=(None,), dtype="int32", name="pretrain")
```

5.2 Shared WordEmbedding layer

[12]:
```
#YOUR CODE HERE### represent aspect as averaged word embedding ###
emb_layer = Embedding(vocab_size, 300, mask_zero=True)
aspect_embedding = emb_layer(aspect_inputs)
average_aspect_embedding = Average()(aspect_embedding)
```

[13]:
```
#YOUR CODE HERE ### sentence representation from embedding ###
sentence_embedding = emb_layer(sentence_inputs)
pretrain_embedding = emb_layer(pretrain_inputs)
```

5.3 Shared BiLSTM layer

[14]:
```
#YOUR CODE HERE ### sentence representation from embedding ###
bilstm = Bidirectional(LSTM(150, dropout=dropout,␣
↪recurrent_dropout=recurrent_dropout, return_sequences=True))
sentence_bilstm = bilstm(sentence_embedding)
pretrain_bilstm = bilstm(pretrain_embedding)
```

5.4 Attention Layer

[15]:
```
##YOUR CODE HERE
attention_weights = Attention()([sentence_bilstm, average_aspect_embedding])
attention_contex = Dot(axes=1)([attention_weights, sentence_bilstm])
```

5.5 Prediction Layer

[16]:
```
shared_prediction = Dense(3, activation="softmax")
last_pretrained_bilstm = Lambda(lambda x: x[:, -1])(pretrain_bilstm)
```

```
densed_output_aspect = shared_prediction(attention_contex)
densed_output_pretrain = shared_prediction(last_pretrained_bilstm)
```

6 Build Models for document-level and aspect-level data

- The two models shared the embedding, BiLSTM, Prediction Layer

[17]:
```
### YOUR CODE HERE
model1 = Model(inputs=[pretrain_inputs], outputs=[densed_output_pretrain])
model2 = Model(inputs=[sentence_inputs, aspect_inputs],␣
↪outputs=[densed_output_aspect])
```

7 Train Model

- First Train model on document-level data.
- Then Train model on aspect-level data

7.1 Train on document-level data

[18]:
```
import keras.optimizers as opt
optimizer=opt.RMSprop(lr=0.001, rho=0.9, epsilon=1e-06, clipnorm=10,␣
↪clipvalue=0)
model1.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['categorical_accuracy'])
batch_size = 128
train_steps_epoch = len(pretrain_data)/batch_size
batch_train_iter_doc = Dataiterator_doc(pretrain_data, pretrain_label,␣
↪batch_size)
```

[19]:
```
###YOUR CODE HERE###
def train_generator_pretrain(model, batch_train_iter, train_steps_epoch,␣
↪epochs):
    def train_gen():
        while True:
            train_batches = [[X, y] for X, y in batch_train_iter]
            for train_batch in train_batches:
                yield train_batch

    history = model.fit(
        train_gen(),
        steps_per_epoch=train_steps_epoch,
        epochs=epochs
    )
    return history
train_generator_pretrain(model1, batch_train_iter_doc, train_steps_epoch, 20)
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning:
Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may
consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

```
Epoch 1/20
235/234 [==============================] - 318s 1s/step - loss: 0.9833 -
categorical_accuracy: 0.5027
Epoch 2/20
235/234 [==============================] - 322s 1s/step - loss: 0.8679 -
categorical_accuracy: 0.5891
Epoch 3/20
235/234 [==============================] - 319s 1s/step - loss: 0.8282 -
categorical_accuracy: 0.6166
Epoch 4/20
235/234 [==============================] - 329s 1s/step - loss: 0.8046 -
categorical_accuracy: 0.6338
Epoch 5/20
235/234 [==============================] - 345s 1s/step - loss: 0.7596 -
categorical_accuracy: 0.6625
Epoch 6/20
235/234 [==============================] - 370s 2s/step - loss: 0.7551 -
categorical_accuracy: 0.6601
Epoch 7/20
235/234 [==============================] - 323s 1s/step - loss: 0.7432 -
categorical_accuracy: 0.6734
Epoch 8/20
235/234 [==============================] - 320s 1s/step - loss: 0.7407 -
categorical_accuracy: 0.6783
Epoch 9/20
235/234 [==============================] - 326s 1s/step - loss: 0.6949 -
categorical_accuracy: 0.7029
Epoch 10/20
235/234 [==============================] - 384s 2s/step - loss: 0.6736 -
categorical_accuracy: 0.7069
Epoch 11/20
235/234 [==============================] - 382s 2s/step - loss: 0.6991 -
categorical_accuracy: 0.6996
Epoch 12/20
235/234 [==============================] - 384s 2s/step - loss: 0.6785 -
categorical_accuracy: 0.7093
Epoch 13/20
235/234 [==============================] - 385s 2s/step - loss: 0.6243 -
categorical_accuracy: 0.7364
Epoch 14/20
235/234 [==============================] - 384s 2s/step - loss: 0.6396 -
categorical_accuracy: 0.7289
```

```
Epoch 15/20
235/234 [==============================] - 383s 2s/step - loss: 0.6580 -
categorical_accuracy: 0.7178
Epoch 16/20
235/234 [==============================] - 386s 2s/step - loss: 0.6320 -
categorical_accuracy: 0.7332
Epoch 17/20
235/234 [==============================] - 384s 2s/step - loss: 0.5999 -
categorical_accuracy: 0.7551
Epoch 18/20
235/234 [==============================] - 387s 2s/step - loss: 0.6025 -
categorical_accuracy: 0.7497
Epoch 19/20
235/234 [==============================] - 433s 2s/step - loss: 0.5992 -
categorical_accuracy: 0.7504
Epoch 20/20
235/234 [==============================] - 384s 2s/step - loss: 0.5991 -
categorical_accuracy: 0.7503
```

[19]:
```
<keras.callbacks.callbacks.History at 0x7f41cc605b00>
```

7.2 Train on aspect-level data

[20]:
```
train_steps_epoch = len(train_x)/batch_size
batch_train_iter_aspect = Dataiterator_aspect([train_x, train_y, train_aspect],␣
↪batch_size)
val_steps_epoch = len(dev_x)/batch_size
batch_val_iter_aspect = Dataiterator_aspect([dev_x, dev_y, dev_aspect],␣
↪batch_size)

import keras.optimizers as opt
optimizer = opt.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08,␣
↪clipnorm=10, clipvalue=0)
model2.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['categorical_accuracy'])
```

[21]:
```
### YOUR CODE HERE ###
from tensorflow.keras.callbacks import EarlyStopping

def train_generator(
    model: tf.keras.Model,
    batch_train_iter: Dataiterator_aspect,
    batch_val_iter: Dataiterator_aspect,
    train_steps_epoch: float,
    val_steps_epoch: float,
    epochs: int
) -> tf.keras.callbacks.History:
```

```
    callbacks = [EarlyStopping(monitor="val_loss", patience=10,␣
↪restore_best_weights=True)]

    def train_gen():
        while True:
            for batch_x_aspect, batch_y_aspect, batch_aspect_terms in␣
↪batch_train_iter:
                yield [[batch_x_aspect, batch_aspect_terms], batch_y_aspect]

    def val_gen():
        while True:
            for val_batch_x, val_batch_y, val_batch_terms in batch_val_iter:
                yield [[val_batch_x, val_batch_terms], val_batch_y]

    return model.fit(
        train_gen(),
        epochs=epochs,
        validation_data=val_gen(),
        steps_per_epoch=train_steps_epoch,
        validation_steps=val_steps_epoch
    )
train_generator(model2, batch_train_iter_aspect, batch_val_iter_aspect,␣
↪train_steps_epoch, val_steps_epoch, 20)
```

```
Epoch 1/20
15/14 [==============================] - 9s 599ms/step - loss: 1.1345 -
categorical_accuracy: 0.3333 - val_loss: 1.1026 - val_categorical_accuracy:
0.3281
Epoch 2/20
15/14 [==============================] - 7s 478ms/step - loss: 1.0669 -
categorical_accuracy: 0.3812 - val_loss: 1.0864 - val_categorical_accuracy:
0.3438
Epoch 3/20
15/14 [==============================] - 7s 470ms/step - loss: 1.0604 -
categorical_accuracy: 0.4062 - val_loss: 1.0496 - val_categorical_accuracy:
0.4922
Epoch 4/20
15/14 [==============================] - 7s 476ms/step - loss: 1.0234 -
categorical_accuracy: 0.4646 - val_loss: 1.0503 - val_categorical_accuracy:
0.5547
Epoch 5/20
15/14 [==============================] - 7s 472ms/step - loss: 0.9957 -
categorical_accuracy: 0.4938 - val_loss: 0.9317 - val_categorical_accuracy:
0.5938
Epoch 6/20
15/14 [==============================] - 7s 465ms/step - loss: 0.9875 -
```

```
categorical_accuracy: 0.5042 - val_loss: 0.9930 - val_categorical_accuracy:
0.5469
Epoch 7/20
15/14 [==============================] - 7s 480ms/step - loss: 0.9403 -
categorical_accuracy: 0.5375 - val_loss: 1.2910 - val_categorical_accuracy:
0.5312
Epoch 8/20
15/14 [==============================] - 7s 476ms/step - loss: 0.9781 -
categorical_accuracy: 0.5271 - val_loss: 0.8810 - val_categorical_accuracy:
0.6094
Epoch 9/20
15/14 [==============================] - 7s 477ms/step - loss: 0.9223 -
categorical_accuracy: 0.5958 - val_loss: 0.8189 - val_categorical_accuracy:
0.6016
Epoch 10/20
15/14 [==============================] - 7s 468ms/step - loss: 0.8610 -
categorical_accuracy: 0.5958 - val_loss: 0.7322 - val_categorical_accuracy:
0.6719
Epoch 11/20
15/14 [==============================] - 7s 462ms/step - loss: 0.8428 -
categorical_accuracy: 0.6042 - val_loss: 0.8542 - val_categorical_accuracy:
0.5547
Epoch 12/20
15/14 [==============================] - 7s 481ms/step - loss: 0.8693 -
categorical_accuracy: 0.6104 - val_loss: 0.9117 - val_categorical_accuracy:
0.6797
Epoch 13/20
15/14 [==============================] - 7s 467ms/step - loss: 0.8838 -
categorical_accuracy: 0.5854 - val_loss: 0.7544 - val_categorical_accuracy:
0.6875
Epoch 14/20
15/14 [==============================] - 7s 473ms/step - loss: 0.8234 -
categorical_accuracy: 0.6313 - val_loss: 0.8768 - val_categorical_accuracy:
0.6172
Epoch 15/20
15/14 [==============================] - 7s 478ms/step - loss: 0.7187 -
categorical_accuracy: 0.7042 - val_loss: 1.0051 - val_categorical_accuracy:
0.6719
Epoch 16/20
15/14 [==============================] - 7s 487ms/step - loss: 0.8077 -
categorical_accuracy: 0.6500 - val_loss: 0.8826 - val_categorical_accuracy:
0.6562
Epoch 17/20
15/14 [==============================] - 7s 479ms/step - loss: 0.7674 -
categorical_accuracy: 0.6604 - val_loss: 0.7802 - val_categorical_accuracy:
0.5938
Epoch 18/20
15/14 [==============================] - 7s 483ms/step - loss: 0.8163 -
```

```
categorical_accuracy: 0.6521 - val_loss: 0.6691 - val_categorical_accuracy:
0.7266
Epoch 19/20
15/14 [==============================] - 7s 475ms/step - loss: 0.7174 -
categorical_accuracy: 0.6687 - val_loss: 0.8296 - val_categorical_accuracy:
0.7109
Epoch 20/20
15/14 [==============================] - 6s 423ms/step - loss: 0.7446 -
categorical_accuracy: 0.6833 - val_loss: 0.6788 - val_categorical_accuracy:
0.5781
```

[21]:
```
<keras.callbacks.callbacks.History at 0x7f41746a6390>
```

7.3 Evaluating on test set

- show the accuracy

[22]:
```python
##YOUR CODE HERE
[_, accuracy] = model2.evaluate(x=[test_x, test_aspect], y=test_y)
print(f"Model accuracy: {accuracy}")
```

```
638/638 [==============================] - 1s 2ms/step
Model accuracy: 0.6332288384437561
```