Online-Appendix zu

„Word Embedding, Neural Networks and Text Classification: what is the State-of-the-Art?“

Estevan Vilar

ESCP Europe

Junior Management Science 4(1) (2019) 35-62

8 Appendix

8.1 Convolutional Neural Network Implementation in Python

```
import numpy as np
import tensorflow as tf

class TextCNN(object):
    def __init__(self, sequence_length, num_classes, vocab_size, embedding_size,
filter_sizes, num_filters, l2_reg_lambda=0.0):
        # Placeholders for input, output and dropout
        self.input_x = tf.placeholder(tf.int32, [None, sequence_length],
name='input_x')
        self.input_y = tf.placeholder(tf.float32, [None, num_classes], name='input_y')
        self.dropout_keep_prob = tf.placeholder(tf.float32, name='dropout_keep_prob')

        # Keeping track of l2 regularization loss (optional)
        l2_loss = tf.constant(0.0)

        # Embedding layer
        with tf.device('/cpu:0'), tf.name_scope('embedding'):
            self.W = tf.Variable(tf.random_uniform([vocab_size, embedding_size], -1.0,
1.0), name="W")
            self.embedded_chars = tf.nn.embedding_lookup(self.W, self.input_x)
            self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)

        # Create a convolution + maxpool layer for each filter size
        pooled_outputs = []
        for i, filter_size in enumerate(filter_sizes):
            with tf.name_scope('conv-maxpool-%s' % filter_size):
                # Convolution Layer
                filter_shape = [filter_size, embedding_size, 1, num_filters]
                W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name='W')
                b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name='b')
                conv = tf.nn.conv2d(
                    self.embedded_chars_expanded,
                    W,
                    strides=[1, 1, 1, 1],
                    padding='VALID',
                    name='conv')

                # Apply nonlinearity
                h = tf.nn.relu(tf.nn.bias_add(conv, b), name='relu')

                # Maxpooling over the outputs
                pooled = tf.nn.max_pool(
                    h,
                    ksize=[1, sequence_length - filter_size + 1, 1, 1],
                    strides=[1, 1, 1, 1],
                    padding='VALID',
                    name='pool')
                pooled_outputs.append(pooled)

        # Combine all the pooled features
        num_filters_total = num_filters * len(filter_sizes)
        self.h_pool = tf.concat(pooled_outputs,3)
        self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])

        # Add dropout
        with tf.name_scope('dropout'):
            self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)

        # Final (unnormalized) scores and predictions
        with tf.name_scope('output'):
            W = tf.get_variable(
                'W',
                shape=[num_filters_total, num_classes],
                initializer=tf.contrib.layers.xavier_initializer())
            b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name='b')
            l2_loss += tf.nn.l2_loss(W)
            l2_loss += tf.nn.l2_loss(b)
            self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name='scores')
            self.predictions = tf.argmax(self.scores, 1, name='predictions')
```

```
        # Calculate mean cross-entropy loss
        with tf.name_scope('loss'):
            losses = tf.nn.softmax_cross_entropy_with_logits_v2(labels = self.input_y,
logits = self.scores) #  only named arguments accepted
            self.loss = tf.reduce_mean(losses) + l2_reg_lambda * l2_loss

        # Accuracy
        with tf.name_scope('accuracy'):
            correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
            self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, 'float'),
name='accuracy')

        with tf.name_scope('num_correct'):
            correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
            self.num_correct = tf.reduce_sum(tf.cast(correct_predictions, 'float'),
name='num_correct')
```

8.2 Convolutional Neural Network + Long Short Term Memory implementation in Python

```
import numpy as np
import tensorflow as tf

class TextCNNRNN(object):
    def __init__(self,embedding_mat, non_static, hidden_unit, sequence_length,
max_pool_size,
        num_classes, embedding_size, filter_sizes, num_filters, l2_reg_lambda=0.0):

        self.input_x = tf.placeholder(tf.int32, [None, sequence_length],
name='input_x')
        self.input_y = tf.placeholder(tf.float32, [None, num_classes], name='input_y')
        self.dropout_keep_prob = tf.placeholder(tf.float32, name='dropout_keep_prob')
        self.batch_size = tf.placeholder(tf.int32, [])
        self.pad = tf.placeholder(tf.float32, [None, 1, embedding_size, 1], name='pad')
        self.real_len = tf.placeholder(tf.int32, [None], name='real_len')

        l2_loss = tf.constant(0.0)
        #EMBEDDING LAYER
        with tf.device('/cpu:0'), tf.name_scope('embedding'):
            W = tf.Variable(embedding_mat, name="W")
            self.embedded_chars = tf.nn.embedding_lookup(W, self.input_x)
            emb = tf.expand_dims(self.embedded_chars, -1)

        pooled_concat = []
        reduced = np.int32(np.ceil((sequence_length) * 1.0 / max_pool_size))

        for i, filter_size in enumerate(filter_sizes):
            with tf.name_scope('conv-maxpool-%s' % filter_size):

                # Zero paddings so that the convolution output have dimension batch x
sequence_length x emb_size x channel
                num_prio = (filter_size-1) // 2
                num_post = (filter_size-1) - num_prio
                pad_prio = tf.concat([self.pad] * num_prio,1)
                pad_post = tf.concat([self.pad] * num_post,1)
                emb_pad = tf.concat([pad_prio, emb, pad_post],1)

                filter_shape = [filter_size, embedding_size, 1, num_filters]
                W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name='W')
                b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name='b')
                conv = tf.nn.conv2d(emb_pad, W, strides=[1, 1, 1, 1], padding='VALID',
name='conv')

                h = tf.nn.relu(tf.nn.bias_add(conv, b), name='relu')

                # Maxpooling over the outputs

                pooled = tf.nn.max_pool(h, ksize=[1, max_pool_size, 1, 1], strides=[1,
max_pool_size, 1, 1], padding='SAME', name='pool')
                pooled = tf.reshape(h, [-1, reduced, num_filters])

                # Cut the feature sequence at the end based on the maximum filter length
                #h_reshape = h_reshape[:, :max_feature_length, :]
                pooled concat.append(pooled)
```

```
      pooled_concat = tf.concat(pooled_concat,2)
      pooled_concat = tf.nn.dropout(pooled_concat, self.dropout_keep_prob)

      lstm_cell = tf.nn.rnn_cell.LSTMCell(num_units=hidden_unit)

      #lstm_cell = tf.nn.rnn_cell.GRUCell(num_units=hidden_unit)
      #THIS
      #lstm_cell = tf.contrib.rnn.GRUCell(num_units=hidden_unit)

      lstm_cell = tf.nn.rnn_cell.DropoutWrapper(lstm_cell,
output_keep_prob=self.dropout_keep_prob)
      #lstm_cell = tf.contrib.rnn.DropoutWrapper(lstm_cell,
output_keep_prob=self.dropout_keep_prob)


      self._initial_state = lstm_cell.zero_state(self.batch_size, tf.float32)
      #inputs = [tf.squeeze(input_, [1]) for input_ in tf.split(1, reduced,
pooled_concat)]
      inputs = [tf.squeeze(input_, [1]) for input_ in
tf.split(pooled concat,num or size splits=int(reduced),axis=1)]
      #outputs, state = tf.nn.rnn(lstm_cell, inputs,
initial_state=self._initial_state, sequence_length=self.real_len)
      outputs, state = tf.contrib.rnn.static_rnn(lstm_cell, inputs,
initial_state=self._initial_state, sequence_length=self.real_len)

      # Collect the appropriate last words into variable output (dimension = batch x
embedding_size)
      output = outputs[0]
      with tf.variable_scope('Output'):
         tf.get_variable_scope().reuse_variables()
         one = tf.ones([1, hidden_unit], tf.float32)
         for i in range(1,len(outputs)):
            ind = self.real_len < (i+1)
            ind = tf.to_float(ind)
            ind = tf.expand_dims(ind, -1)
            mat = tf.matmul(ind, one)
            output = tf.add(tf.multiply(output, mat),tf.multiply(outputs[i], 1.0 -
mat))

      with tf.name_scope('output'):
         W = tf.Variable(tf.truncated_normal([hidden_unit, num_classes], stddev=0.1),
name='W')
         b = tf.Variable(tf.constant(0.1, shape=[num classes]), name='b')
         l2_loss += tf.nn.l2_loss(W)
         l2_loss += tf.nn.l2_loss(b)
         self.scores = tf.nn.xw_plus_b(output, W, b, name='scores')
         self.predictions = tf.argmax(self.scores, 1, name='predictions')

      with tf.name_scope('loss'):
         losses = tf.nn.softmax_cross_entropy_with_logits_v2(labels = self.input_y,
logits = self.scores) #  only named arguments accepted
         self.loss = tf.reduce_mean(losses) + l2_reg_lambda * l2_loss

      with tf.name_scope('accuracy'):
         correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
         self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"),
name='accuracy')

      with tf.name_scope('num_correct'):
         correct = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
         self.num_correct = tf.reduce_sum(tf.cast(correct, 'float'))
```

8.3 Proposal implementation in Python

```
def load_embedding_vectors_proposal600(vocabulary, filename, binary,
vocabularylist):# load embedding_vectors from the word2vec
   encoding = 'utf-8'
   embedding_vectors_final = np.random.uniform(0, 0, (len(vocabulary),600 ))
   i = 0
   with open(filename, "rb") as f:
      header = f.readline()
      vocab_size, vector_size = map(int, header.split())
      # initial matrix with 0
      embedding_vectors = np.random.uniform(0, 0, (len(vocabulary), vector_size))
```

```
      if binary:
         binary_len = np.dtype('float32').itemsize * vector_size

         for line_no in tqdm(range(vocab_size)):
            word = []
            while True:
               ch = f.read(1)
               if ch == b' ':
                  break
               if ch == b'':
                  raise EOFError("unexpected end of input; is count incorrect or file
otherwise damaged?")
               if ch != b'\n':
                  word.append(ch)
            word = str(b''.join(word), encoding=encoding, errors='strict')
            idx = vocabulary.get(word)
            if idx !=0:

               i+=1
               embedding_vectors[idx] = np.fromstring(f.read(binary_len),
dtype='float32')


            else:
               f.seek(binary_len, 1)

      print('found ',i,' vectors')
   f.close()

   with open('saved.modeltrec.bin',"rb") as g:
      header2=g.readline()
      vocab_size2, vector_size2=map(int, header2.split())
      binary_len2=np.dtype('float32').itemsize * vector_size2
      # initial matrix with 0
      embedding_vectors2 = np.random.uniform(0, 0, (len(vocabulary), vector_size2))

      for line_no in tqdm(range(vocab_size2)):
         word2 = []
         while True:
            ch2 = g.read(1)
            if ch2 == b' ':
               break
            if ch2 == b'':
               raise EOFError("unexpected end of input; is count incorrect or file
otherwise damaged?")
            if ch2 != b'\n':
               word2.append(ch2)
         word2 = str(b''.join(word2), encoding=encoding, errors='strict')
         idx2 = vocabulary.get(word2)
         if idx2!=0:

            embedding_vectors2[idx2]=np.fromstring(g.read(binary_len2),
dtype='float32')
         else:
            g.seek(binary_len2, 1)
   g.close()
   a=0
   b=0
   c=0
   d=0
   e=0
   for token in range(len(vocabularylist)):
      a+=1
      if embedding_vectors[token].all(0)!=False:
         if embedding_vectors2[token].all(0)!=False:
            b+=1
embedding_vectors_final[token]=np.concatenate((embedding_vectors[token],embedding_vec
tors2[token]))
         else:
            c += 1
            embedding_vectors_final[token] =
np.concatenate((embedding_vectors[token]#,np.random.uniform(-0.25,
0.25,vector_size2)))
                                                ,np.random.uniform(0.00, 0.00,
vector_size2)))
      else:
         if embedding_vectors2[token].all(0)!=False:
```

```
            e += 1
            embedding_vectors_final[token] = np.concatenate(
               (np.random.uniform(-0.25, 0.25, vector_size),
embedding_vectors2[token]))
         else:
            embedding_vectors_final[token] = np.concatenate(
               (np.random.uniform(-0.25, 0.25, vector_size), np.random.uniform(-0.25,
0.25, vector_size2)))

            d+=1
   print('Out of ', a, ' vocabulary words:')
   print(b, ' were found in both embedding')
   print(c, ' were only found in Google News embedding')
   print(e, ' were only found in data embedding')
   print(d,' were not found at not at all')
return embedding_vectors
```

8.4 Results

LogMeIn	Model	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Devia	Lower CI	Upper CI
	CNN300	0.9298	0.9035	0.8684	0.9211	0.8947	0.9035	0.0240	0.8794	0.9276
	CNN600	0.8684	0.8859	0.8684	0.8421	0.8421	0.8614	0.0190	0.8423	0.8804
	CNNW2V	0.9649	0.9298	0.9385	0.9386	0.8947	0.9333	0.0253	0.9080	0.9586
	CNNGVE	0.9123	0.9123	0.8947	0.9211	0.9211	0.9123	0.0108	0.9015	0.9231
	CNNFXT	0.9386	0.9211	0.9561	0.9474	0.9298	0.9386	0.0139	0.9247	0.9525
	CNNFXT_SUB	0.9649	0.9123	0.9298	0.8596	0.9298	0.9193	0.0384	0.8807	0.9578
	CNN600NULL	0.9123	0.9561	0.9386	0.9649	0.9035	0.9351	0.0267	0.9083	0.9619
	CNN600_PROPOSAL	0.9649	0.9035	0.9035	0.9437	0.9211	0.9273	0.0267	0.9006	0.9541
	LSTM300	0.8684	0.8421	0.8333	0.9035	0.9035	0.8702	0.0331	0.8370	0.9033
	LSTM600	0.8596	0.9298	0.8772	0.8860	0.9211	0.8947	0.0298	0.8649	0.9246
	LSTMW2V	0.8772	0.9211	0.9211	0.9035	0.8246	0.8895	0.0405	0.8489	0.9301
	LSTMGVE	0.9298	0.9298	0.8684	0.9474	0.9208	0.9192	0.0300	0.8892	0.9493
	LSTMFXT	0.8772	0.9035	0.9474	0.8719	0.9298	0.9060	0.0327	0.8732	0.9387
	LSTMFXT_SUB	0.9474	0.9474	0.8772	0.9211	0.8947	0.9176	0.0314	0.8861	0.9490
	LSTMW2V600_NULL	0.9123	0.9035	0.9298	0.9386	0.9035	0.9175	0.0159	0.9016	0.9335
	LSTMW2V600	0.9123	0.8860	0.9386	0.9649	0.8684	0.9140	0.0389	0.8750	0.9531

Results on TREC dataset. Confidence Interval is 95%

TREC	Model	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Devia	Lower CI	Upper CI
	CNN300	0.7950	0.7575	0.7875	0.7600	0.7825	0.7765	0.0168	0.7596	0.7934
	CNN600	0.7750	0.7775	0.7850	0.7975	0.7700	0.7810	0.0107	0.7703	0.7917
	CNNW2V	0.8325	0.8550	0.8400	0.8450	0.8400	0.8425	0.0083	0.8342	0.8508
	CNNGVE	0.7450	0.7775	0.7850	0.7750	0.7825	0.7730	0.0161	0.7568	0.7892
	CNNFXT	0.8450	0.8375	0.8300	0.8725	0.8425	0.8455	0.0161	0.8293	0.8617
	CNNFXT_SUB	0.8300	0.8100	0.8325	0.8425	0.8350	0.8300	0.0121	0.8179	0.8421
	CNN600NULL	0.8575	0.8625	0.8425	0.8425	0.8175	0.8445	0.0175	0.8269	0.8621
	CNN600_PROPOSAL	0.8325	0.8150	0.8050	0.8150	0.8150	0.8165	0.0099	0.8065	0.8265
	LSTM300	0.7625	0.7975	0.7675	0.8100	0.7900	0.7855	0.0201	0.7653	0.8057
	LSTM600	0.7825	0.7800	0.8000	0.7925	0.7925	0.7895	0.0082	0.7813	0.7977
	LSTMW2V	0.8450	0.8400	0.8050	0.8125	0.8725	0.8350	0.0271	0.8078	0.8622
	LSTMGVE	0.7975	0.8125	0.8325	0.8275	0.8100	0.8160	0.0141	0.8019	0.8301
	LSTMFXT	0.8325	0.8350	0.8575	0.8475	0.8100	0.8365	0.0179	0.8186	0.8544
	LSTMFXT_SUB	0.8300	0.8475	0.8025	0.8000	0.7925	0.8145	0.0233	0.7912	0.8378
	LSTM600NULL	0.8650	0.8300	0.8250	0.8050	0.8000	0.8250	0.0257	0.8068	0.8432
	LSTMW2V600	0.8500	0.8175	0.8425	0.8025	0.8525	0.8330	0.0220	0.8174	0.8486

	Results using the LSTM model with an RMSprop gradient update function								
LogMeIn	**Model**	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**	**Standard Devia**	**Lower CI**	**Upper CI**
	LSTM300	0.5702	0.6223	0.5439	0.6491	0.6491	0.6069	0.0477	0.5591	0.6548
	LSTM600	0.5789	0.6053	0.5439	0.6228	0.5614	0.5825	0.0320	0.5504	0.6145
	LSTMW2V	0.6754	0.6140	0.6491	0.6228	0.5965	0.6316	0.0310	0.6005	0.6626
	LSTMGVE	0.7632	0.6578	0.6930	0.7368	0.7368	0.7175	0.0418	0.6756	0.7594
	LSTMFXT	0.5088	0.6316	0.6053	0.6491	0.6404	0.6070	0.0573	0.5496	0.6645
	LSTMFXT_SUB	0.4825	0.5000	0.5263	0.5439	0.6053	0.5316	0.0475	0.4840	0.5792
	LSTM600NULL	0.6316	0.6316	0.6140	0.6754	0.6404	0.6386	0.0227	0.6158	0.6614
	LSTM600_PROPOSAL	0.5439	0.5965	0.5965	0.6140	0.6052	0.5912	0.0274	0.5637	0.6187
TREC	**Model**	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**	**Standard Devia**	**Lower CI**	**Upper CI**
	LSTM300	0.6800	0.7450	0.6900	0.6425	0.7425	0.7000	0.0437	0.6562	0.7438
	LSTM600	0.7125	0.7325	0.6775	0.7000	0.6800	0.7005	0.0230	0.6774	0.7236
	LSTMW2V	0.8200	0.7650	0.7475	0.7400	0.8000	0.7745	0.0344	0.7400	0.8090
	LSTMGVE	0.7450	0.7600	0.7175	0.7200	0.7475	0.7380	0.0185	0.7195	0.7565
	LSTMFXT	0.7725	0.7200	0.7150	0.7975	0.7700	0.7550	0.0359	0.7190	0.7910
	LSTMFXT_SUB	0.6700	0.6725	0.7125	0.6875	0.6750	0.6835	0.0176	0.6659	0.7011
	LSTM600NULL	0.7925	0.7925	0.7850	0.7900	0.7625	0.7845	0.0127	0.7718	0.7972
	LSTM600_PROPOSAL	0.7300	0.7300	0.7450	0.7825	0.7800	0.7535	0.0261	0.7274	0.7796