INFO7374 Assignment 4

Summary	In this codelab, we will discuss the basic features about relation extraction and its implementation.
URL	https://github.com/ll1195831146/Infor7374-AI/tree/master/Assignment4
YouTube Video	https://youtu.be/jEGyL6XVqM8
Category	NLP
Environment	TensorFlow, Google Colab
Status	Done
Feedback Link	https://github.com/ll1195831146/Infor7374-AI/tree/master/Assignment4/issues
Author	Yuchen He, Lei Liu, Xiangyu Chen

Introduction

Natural language processing (**NLP**) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation (From Wikipedia).

What is a Relation Extraction?

Relationship Extraction is a very interesting problem in natural language processing. The idea is to link two entities, such as the owner of a company, or the someone's company position and a person in unstructured text sources. An example would be to extract Bill Gates and Microsoft from the following unstructured text:

"<PERSON> Bill Gates </PERSON>, the founder of <ORG> Microsoft </ORG>, hosted a party last night".

Surprisingly the methods that are typically used on large projects are remarkably simple based on feature extraction as a binary based classification problem. More complicated supervised methods using kernel methods that bypass the need for generating explicit features perform better but are not as well suited for larger domain problems. We shall discuss these methods in a later post.

Supervised Feature Extraction

In most cases, it is typical to try and first identify which words are the entities we wish to extract.

First, entity based features can be developed. These include:

1. Gazetteers (i.e. dictionaries of organization names etc)
2. Features based on:
    - Parts of speech tags
    - Regular expressions
    - Word length
    - Word shape
    - Substring
    - Capitalized letters
3. More complicated features such as:
    - Bigrams
    - Sequencing modeling - especially for the words between the two sequences.

Now that we have features generated we can classify every pair of words using typical supervised machine learning algorithms.

Semi-Supervised Seeding

We can also proceed in semi-supervised manner, where if we know certain relationships, say Bill Gates and Microsoft, we can learn the rule:

"<PERSON> X </PERSON>, the founder of <ORG> Y </ORG>, hosted a party last night".

With a large dataset, we can apply this exact sentence pattern to find new Person-Organization relations. Then, say we find *X'* and *Y'* entities, we can then find different sentence structures, and bootstrap this until we have found them all.

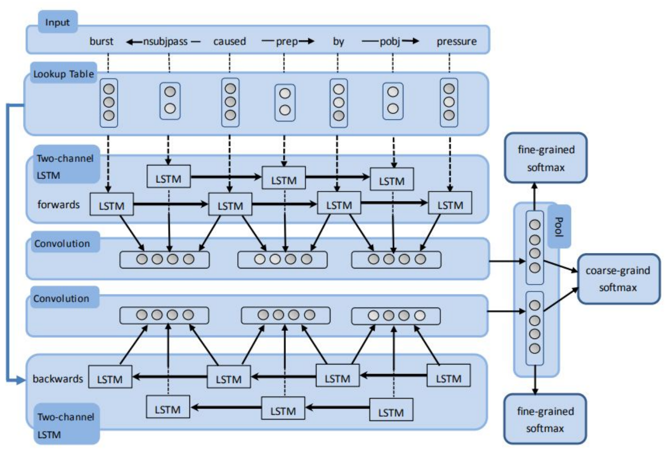
As with a lot of these rule generation methods, the results will typically give high precision but low recall.

Related Work

[Paper 1: Bidirectional Recurrent Convolutional Neural Network for Relation Classification]()

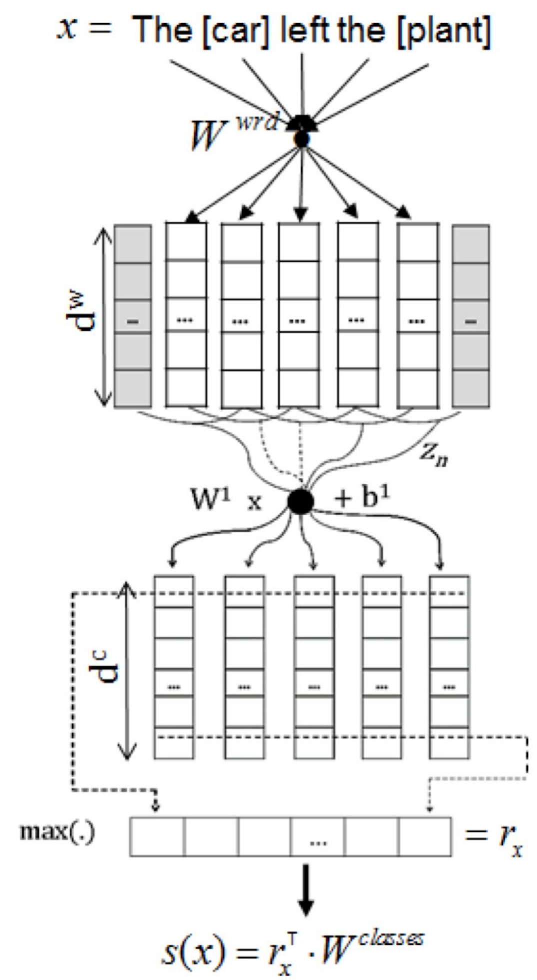
In this paper, the authors used SemEval2010 dataset, which is an established benchmark for relation classification. The dataset contains 8000 sentences for training, and 2717 for testing. We split 800 samples out of the training set for validation.

Their method is to build a BCRNN model is used to learn representations with bidirectional information along the SDP forwards and backwards at the same time. Given a sentence and its dependency tree, they build the neural network on its SDP extracted from the tree. Along the SDP, two recurrent neural networks with long short term memory units are applied to learn hidden representations of words and dependency relations respectively. A convolution layer is applied to capture local features from hidden representations of every two neighbor words and the dependency relations between them. A max pooling layer thereafter gathers information from local features of the SDP or the inverse SDP. They have a softmax output layer after pooling layer for classification in the unidirectional model RCNN. On the basis of RCNN model, we build a bidirectional architecture BRCNN taking the SDP and the inverse SDP of a sentence as input. During the training stage of a (K+1)-relation task, 757 two fine-grained softmax classifiers of RCNNs do a (2K + 1)-class classification respectively. The pooling layers of two RCNNs are concatenated and a coarse-grained softmax output layer is followed to do a (K + 1)-class classification. The final (2K+1)-class distribution is the combination of two (2K+1)-class distributions provided by fine-grained classifiers respectively during the testing stage.



In their conclusion, they emphasized that the BRCNN model, consisting of two RCNNs, learns features along SDP and inversely at the same time. Information of words and dependency relations are used utilizing a two-channel recurrent neural network with LSTM units. The features of dependency units in SDP are extracted by a convolution layer.

Paper 2: Classifying Relations by Ranking with Convolutional Neural Networks



In this paper, the authors used SemEval2010 dataset as well. They used a CNN that performs classification by ranking, which is a improvement of the CNN with softmax model that is called CR-CNN. The main contributions of this paper are: (1) the definition of new CNN for classification especially for the SemEval-2010 Task 8 dataset without using any costly handcrafted features, which is called CR-CNN; (2) an effective method to deal with artificial classes by omitting their embeddings in CR-CNN; (3) the demonstration that using only the text between target nominals is almost as effective as using word position embeddings; and 4) a method to extract from the CR-CNN model the most representative contexts of each relation type which is matrix-vector recursive neural network.

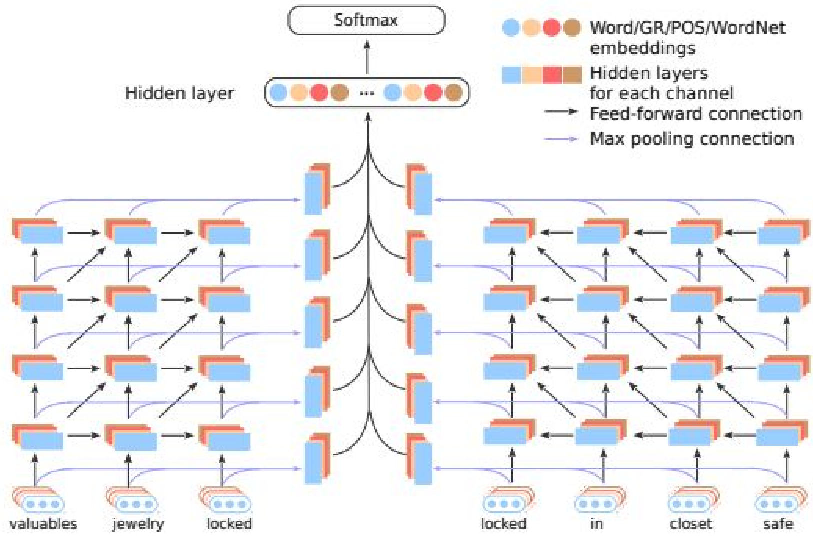
Relation	(e1,e2)	(e2,e1)
Cause-Effect	$e1$ resulted in, $e1$ caused a, had caused the, poverty cause $e2$, caused a $e2$	$e2$ caused by, was caused by, are caused by, been caused by, $e2$ from $e1$
Component-Whole	$e1$ of the, of the $e2$, part of the, in the $e2$, $e1$ on the	$e2$'s $e1$, with its $e1$, $e2$ has a, $e2$ comprises the, $e2$ with $e1$
Content-Container	was in a, was hidden in, were in a, was inside a, was contained in	$e2$ full of, $e2$ with $e1$, $e2$ was full, $e2$ contained a, $e2$ with cold
Entity-Destination	$e1$ into the, $e1$ into a, $e1$ to the, was put inside, imported into the	-
Entity-Origin	away from the, derived from a, had left the, derived from an, $e1$ from the	the source of, $e2$ grape $e1$, $e2$ butter $e1$
Instrument-Agency	are used by, $e1$ for $e2$, is used by, trade for $e2$, with the $e2$	with a $e1$, by using $e1$, $e2$ finds a, $e2$ with a, $e2$, who
Member-Collection	of the $e2$, in the $e2$, of this $e2$, the political $e2$, $e1$ collected in	$e2$ of $e1$, of wild $e1$, of elven $e1$, $e2$ of different, of 0000 $e1$
Message-Topic	$e1$ is the, $e1$ asserts the, $e1$ that the, on the $e2$, $e1$ inform about	described in the, discussed in the, featured in numerous, discussed in cabinet, documented in two,
Product-Producer	$e1$ by the, by a $e2$, of the $e2$, by the $e2$, from the $e2$	$e2$ of the, $e2$ has constructed, $e2$'s $e1$, $e2$ came up, $e2$ who created

Table 6: List of most representative trigrams for each relation type.

Paper 3: Improved Relation Classification by Deep Recurrent Neural Networks with Data Augmentation

In this paper, the authors used SemEval2010 dataset as well. Neural networks, especially deep ones, are likely to be prone to overfitting. The SemEval-2010 relation classification dataset comprises only several thousand samples, which may not fully sustain the training of deep RNNs. To mitigate this problem, they proposed a data augmentation technique for relation classification by making use of the directionality of relationships. The two sub-paths [valuables]e1 → jewelry → locked locked ← in ← closet ← [safe]e2 in Figure 1, for example, can be mapped to the subject-predicate and object- predicate components in the relation Content-Container(e1, e2). If we change the order of these two subpaths, we obtain [safe]e1 → closet → in → locked locked ← jewelry ← [valuables]e2 Then the relationship becomes Container-Content(e1, e2), which is exactly the inverse of Content-Container(e1, e2). In this way, we can augment the dataset without using additional resources.

They decided to build DRNNs on the shortest dependency path (SDP), which serves as a backbone. In particular, an RNN picks up information along each sub-path, separated by the common ancestor of marked entities. Also, they took advantage of four information channels, namely, word embeddings, POS embeddings, grammatical relation embeddings, and WordNet embeddings. They also designed deep RNNs with up to four hidden layers so as to capture information in different levels of abstraction. For each RNN layer, max pooling gathered information from different recurrent nodes. Notice that the four channels (with eight sub-paths) are processed in a similar way. Then all pooling layers are concatenated and fed into a hidden layer for information integration. Finally, they had a softmax output layer for classification.



In their conclusion, they mentioned that the DRNNs model, consisting of several RNN layers, explores the representation space of different abstraction levels. By visualizing DRNNs' units, they demonstrated that high-level layers are more capable of integrating information relevant to target relations. In addition, they had designed a data augmentation strategy by leveraging the directionality of relations. When evaluated on the SemEval dataset, their DRNNs model results in substantial performance boost. The performance generally improves when the depth increases; with a depth of 4, our model reaches the highest F1-measure of 86.1%.

Paper 4: Semantic Relation Classification via Bidirectional LSTM Networks with Entity-aware Attention using Latent Entity Typing

In this paper, the authors used SemEval2010 dataset as well. Entity-aware attention mechanism with latent entity typing and a novel end-to-end recurrent neural model which incorporates this mechanism for relation classification. Their model use only raw sentence and word embeddings without any high-level features from NLP tools, and achieves 85.2% F1-score in SemEval-2010 Task 8.

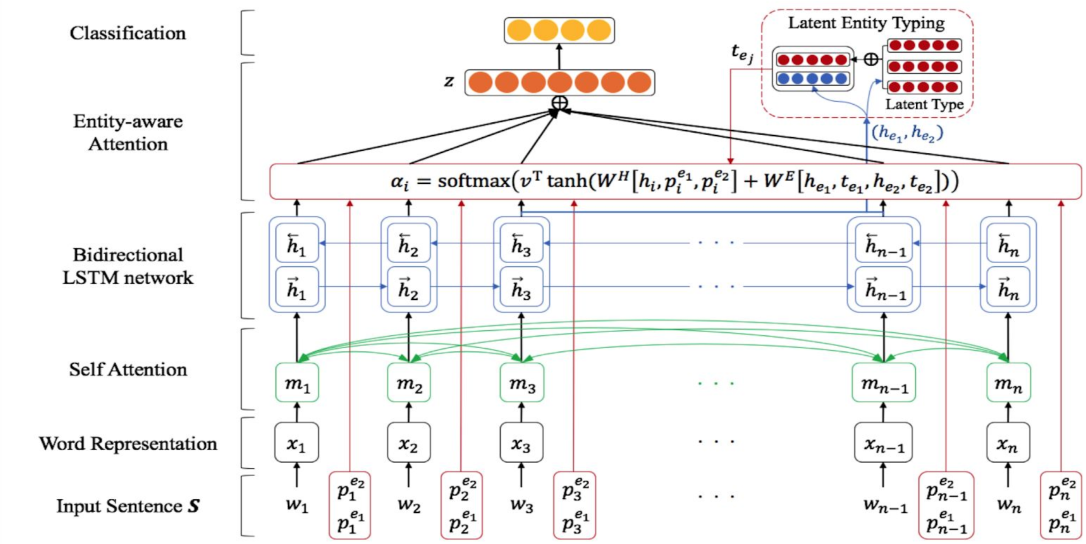


Figure 2: The architecture of our model (best viewed in color). Entity 1 and 2 corresponds to the 3 and $(n-1)$-th words, respectively, which are fed into the LET.

In addition, they have three visualizations of attention mechanisms that applied to the model demonstrate which is more interpretable than previous models. They pretend their model to be extended not only the relation classification task but also other tasks that entity plays an important role. Especially, latent entity typing can be effectively applied to sequence modeling task using entity information without NER.

Type1 : *worker, chairman, author, king, potter, cuisine, spaghetti, restaurant, sugars, bananas, salad, bean*
Type2 : *systems, engine, trucks, valve, hinge, assembly, woofer, mainspring, wriggle, circuit, motor*
Type3 : *virus, tsunami, accident, dust, riot, pandemic, pollution, earthquake, contamination, debt,, congestion, drugs, marijuana*

Figure 7: Sets of Entities grouped by Latent Types

Getting Start

Download the Libraries

[Installing Tensorflow]

```
pip install tensorflow
```

[Installing NLTK]

```
pip install -U nltk
```

[Installing NLTK Data]

```
import nltk
nltk.download('punkt')
```

Pre-trained Word2Vec

glove.6B.100d: Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip

glove.840B.300d: Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip

Prototype Implementation

Model We Used

1. **Entity Attention Bi-LSTM (Lee et al., 2019)**
2. **CNN (Zeng et al., 2014)**

Entity Attention Bi-LSTM (Lee et al., 2019)

Labels

```
class2label = {'Other': 0,
                'Message-Topic(e1,e2)': 1,
'Message-Topic(e2,e1)': 2,
                'Product-Producer(e1,e2)': 3,
'Product-Producer(e2,e1)': 4,
                'Instrument-Agency(e1,e2)': 5,
'Instrument-Agency(e2,e1)': 6,
                'Entity-Destination(e1,e2)': 7,
'Entity-Destination(e2,e1)': 8,
                'Cause-Effect(e1,e2)': 9, 'Cause-Effect(e2,e1)':
10,
                'Component-Whole(e1,e2)': 11,
'Component-Whole(e2,e1)': 12,
                'Entity-Origin(e1,e2)': 13,
'Entity-Origin(e2,e1)': 14,
                'Member-Collection(e1,e2)': 15,
'Member-Collection(e2,e1)': 16,
                'Content-Container(e1,e2)': 17,
'Content-Container(e2,e1)': 18}

label2class = {0: 'Other',
                1: 'Message-Topic(e1,e2)', 2:
'Message-Topic(e2,e1)',
                3: 'Product-Producer(e1,e2)', 4:
'Product-Producer(e2,e1)',
                5: 'Instrument-Agency(e1,e2)', 6:
'Instrument-Agency(e2,e1)',
                7: 'Entity-Destination(e1,e2)', 8:
'Entity-Destination(e2,e1)',
                9: 'Cause-Effect(e1,e2)', 10:
'Cause-Effect(e2,e1)',
                11: 'Component-Whole(e1,e2)', 12:
'Component-Whole(e2,e1)',
                13: 'Entity-Origin(e1,e2)', 14:
'Entity-Origin(e2,e1)',
                15: 'Member-Collection(e1,e2)', 16:
'Member-Collection(e2,e1)',
                17: 'Content-Container(e1,e2)', 18:
'Content-Container(e2,e1)'}
```

Attention Mechanisms

Over the last few years, Attention Mechanisms have found broad application in all kinds of natural language processing (NLP) tasks based on deep learning.

In traditional encode-decode model, the effect of each input on each output is equivalent, which is obviously unreasonable. To solve this problem, The attention mechanism is usually used between encode and decode.
The basic idea: each time the model predicts an output word, it only uses parts of an input where the most relevant information is concentrated instead of an entire sentence.

```
def attention(inputs, e1, e2, p1, p2, attention_size):
    # inputs = (batch, seq_len, hidden)
    # e1, e2 = (batch, seq_len)
    # p1, p2 = (batch, seq_len, dist_emb_size)
    # attention_size = scalar(int)
    def extract_entity(x, e):
        e_idx =
tf.concat([tf.expand_dims(tf.range(tf.shape(e)[0]), axis=-1),
tf.expand_dims(e, axis=-1)], axis=-1)
        return tf.gather_nd(x, e_idx)  # (batch, hidden)
    seq_len = tf.shape(inputs)[1]  # fixed at run-time
    hidden_size = inputs.shape[2].value  # fixed at
compile-time
    latent_size = hidden_size

    # Latent Relation Variable based on Entities
    e1_h = extract_entity(inputs, e1)  # (batch, hidden)
    e2_h = extract_entity(inputs, e2)  # (batch, hidden)
    e1_type, e2_type, e1_alphas, e2_alphas =
latent_type_attention(e1_h, e2_h,

num_type=3,

latent_size=latent_size)  # (batch, hidden)
    e1_h = tf.concat([e1_h, e1_type], axis=-1)  # (batch,
hidden+latent)
    e2_h = tf.concat([e2_h, e2_type], axis=-1)  # (batch,
hidden+latent)

    # v*tanh(W*[h;p1;p2]+W*[e1;e2]) 85.18%? 84.83% 84.55%
    e_h = tf.layers.dense(tf.concat([e1_h, e2_h], -1),
attention_size, use_bias=False,
kernel_initializer=initializer())
    e_h = tf.reshape(tf.tile(e_h, [1, seq_len]), [-1, seq_len,
```

```
attention_size])
    v = tf.layers.dense(tf.concat([inputs, p1, p2], axis=-1),
attention_size, use_bias=False,
kernel_initializer=initializer())
    v = tf.tanh(tf.add(v, e_h))

    u_omega = tf.get_variable("u_omega", [attention_size],
initializer=initializer())
    vu = tf.tensordot(v, u_omega, axes=1, name='vu')  # (batch,
seq_len)
    alphas = tf.nn.softmax(vu, name='alphas')  # (batch,
seq_len)

    # v*tanh(W*[h;p1;p2;e1;e2]) 85.18% 84.41%
    # e1_h = tf.reshape(tf.tile(e1_h, [1, seq_len]), [-1,
seq_len, hidden_size+latent_size])
    # e2_h = tf.reshape(tf.tile(e2_h, [1, seq_len]), [-1,
seq_len, hidden_size+latent_size])
    # v = tf.concat([inputs, p1, p2, e1_h, e2_h], axis=-1)
    # v = tf.layers.dense(v, attention_size,
activation=tf.tanh, kernel_initializer=initializer())
    #
    # u_omega = tf.get_variable("u_omega", [attention_size],
initializer=initializer())
    # vu = tf.tensordot(v, u_omega, axes=1, name='vu')  #
(batch, seq_len)
    # alphas = tf.nn.softmax(vu, name='alphas')  # (batch,
seq_len)

    # output
    output = tf.reduce_sum(inputs * tf.expand_dims(alphas, -1),
1)  # (batch, hidden)

    return output, alphas, e1_alphas, e2_alphas
```

Where, MultiHead attention is taking linear transformation of Q,K and V for h times. Then performing attention, and splicing the results of h times to perform a linear transformation.

```
def multihead_attention(queries, keys, num_units, num_heads,
                        dropout_rate=0,
scope="multihead_attention", reuse=None):
    with tf.variable_scope(scope, reuse=reuse):
```

```
        # Linear projections
        Q = tf.layers.dense(queries, num_units,
kernel_initializer=initializer())  # (N, T_q, C)
        K = tf.layers.dense(keys, num_units,
kernel_initializer=initializer())  # (N, T_k, C)
        V = tf.layers.dense(keys, num_units,
kernel_initializer=initializer())  # (N, T_k, C)

        # Split and concat
        Q_ = tf.concat(tf.split(Q, num_heads, axis=2), axis=0)
# (h*N, T_q, C/h)
        K_ = tf.concat(tf.split(K, num_heads, axis=2), axis=0)
# (h*N, T_k, C/h)
        V_ = tf.concat(tf.split(V, num_heads, axis=2), axis=0)
# (h*N, T_k, C/h)

        # Multiplication
        outputs = tf.matmul(Q_, tf.transpose(K_, [0, 2, 1]))  #
(h*N, T_q, T_k)

        # Scale
        outputs /= K_.get_shape().as_list()[-1] ** 0.5

        # Key Masking
        key_masks = tf.sign(tf.abs(tf.reduce_sum(keys,
axis=-1)))  # (N, T_k)
        key_masks = tf.tile(key_masks, [num_heads, 1])  # (h*N,
T_k)
        key_masks = tf.tile(tf.expand_dims(key_masks, 1), [1,
tf.shape(queries)[1], 1])  # (h*N, T_q, T_k)

        paddings = tf.ones_like(outputs) * (-2 ** 32 + 1)
        outputs = tf.where(tf.equal(key_masks, 0), paddings,
outputs)  # (h*N, T_q, T_k)

        # Activation
        alphas = tf.nn.softmax(outputs)  # (h*N, T_q, T_k)

        # Query Masking
        query_masks = tf.sign(tf.abs(tf.reduce_sum(queries,
axis=-1)))  # (N, T_q)
        query_masks = tf.tile(query_masks, [num_heads, 1])  #
(h*N, T_q)
        query_masks = tf.tile(tf.expand_dims(query_masks, -1),
```

```
[1, 1, tf.shape(keys)[1]])  # (h*N, T_q, T_k)
        alphas *= query_masks  # broadcasting. (N, T_q, C)

        # Dropouts
        alphas = tf.layers.dropout(alphas, rate=dropout_rate,
training=tf.convert_to_tensor(True))

        # Weighted sum
        outputs = tf.matmul(alphas, V_)  # ( h*N, T_q, C/h)
        # Restore shape
        outputs = tf.concat(tf.split(outputs, num_heads,
axis=0), axis=2)  # (N, T_q, C)

        # Linear
        outputs = tf.layers.dense(outputs, num_units,
activation=tf.nn.relu, kernel_initializer=initializer())

        # Residual connection
        outputs += queries
        # Normalize
        outputs = layer_norm(outputs)  # (N, T_q, C)

    return outputs, alphas
```

In 2018, considering the distance between words, Google's machine translation team presented the essay "Self-Attention with Relative Position Representations." and optimized Position encoding.

```
def get_relative_position(df, max_sentence_length):
    # Position data
    pos1 = []
    pos2 = []
    for df_idx in range(len(df)):
        sentence = df.iloc[df_idx]['sentence']
        tokens = nltk.word_tokenize(sentence)
        e1 = df.iloc[df_idx]['e1']
        e2 = df.iloc[df_idx]['e2']

        p1 = ""
        p2 = ""
        for word_idx in range(len(tokens)):
            p1 += str((max_sentence_length - 1) + word_idx -
```

```
e1) + " "
            p2 += str((max_sentence_length - 1) + word_idx -
e2) + " "
        pos1.append(p1)
        pos2.append(p2)

    return pos1, pos2
```

Result

Without word-embedding:

```
Evaluation:
2019-04-03T02:25:57.157410: step 40000, loss 3.74313, acc 0.724005
<<< (9+1)-WAY EVALUATION TAKING DIRECTIONALITY INTO ACCOUNT -- OFFICIAL
>>>:
macro-averaged F1-score = 77.37%, Best = 77.59%
```

With word-embedding glove.6B.300d:

```
Evaluation:
2019-04-04T21:45:58.499478: step 40000, loss 4.80931, acc 0.788711
<<< (9+1)-WAY EVALUATION TAKING DIRECTIONALITY INTO ACCOUNT -- OFFICIAL
>>>:
macro-averaged F1-score = 82.78%, Best = 83.57%
```

With word-embedding glove.840B.300d:

```
Evaluation:
2019-04-04T22:48:58.192466: step 31500, loss 5.23648, acc 0.797967
<<< (9+1)-WAY EVALUATION TAKING DIRECTIONALITY INTO ACCOUNT -- OFFICIAL
>>>:
macro-averaged F1-score = 83.43%, Best = 83.96%
```

TextCNN (Zeng et al.,2014)

Labels

```
class2label = {'Other': 0,
               'Message-Topic(e1,e2)': 1,
'Message-Topic(e2,e1)': 2,
               'Product-Producer(e1,e2)': 3,
'Product-Producer(e2,e1)': 4,
               'Instrument-Agency(e1,e2)': 5,
'Instrument-Agency(e2,e1)': 6,
               'Entity-Destination(e1,e2)': 7,
'Entity-Destination(e2,e1)': 8,
               'Cause-Effect(e1,e2)': 9, 'Cause-Effect(e2,e1)':
10,
               'Component-Whole(e1,e2)': 11,
'Component-Whole(e2,e1)': 12,
               'Entity-Origin(e1,e2)': 13,
'Entity-Origin(e2,e1)': 14,
               'Member-Collection(e1,e2)': 15,
'Member-Collection(e2,e1)': 16,
               'Content-Container(e1,e2)': 17,
'Content-Container(e2,e1)': 18}
```

TextCNN Model

```
class TextCNN:
    def __init__(self, sequence_length, num_classes,
                 text_vocab_size, text_embedding_size,
pos_vocab_size, pos_embedding_size,
                 filter_sizes, num_filters, l2_reg_lambda=0.0):

        # Placeholders for input, output and dropout
        self.input_text = tf.placeholder(tf.int32, shape=[None,
sequence_length], name='input_text')
        self.input_p1 = tf.placeholder(tf.int32, shape=[None,
sequence_length], name='input_p1')
        self.input_p2 = tf.placeholder(tf.int32, shape=[None,
sequence_length], name='input_p2')
        self.input_y = tf.placeholder(tf.float32, shape=[None,
num_classes], name='input_y')
        self.dropout_keep_prob = tf.placeholder(tf.float32,
name='dropout_keep_prob')

        initializer = tf.keras.initializers.glorot_normal
```

```
        # Embedding layer
        with tf.variable_scope("text-embedding"):
            self.W_text =
tf.Variable(tf.random_uniform([text_vocab_size,
text_embedding_size], -0.25, 0.25), name="W_text")
            self.text_embedded_chars =
tf.nn.embedding_lookup(self.W_text, self.input_text)
            self.text_embedded_chars_expanded =
tf.expand_dims(self.text_embedded_chars, -1)

        with tf.variable_scope("position-embedding"):
            self.W_pos = tf.get_variable("W_pos",
[pos_vocab_size, pos_embedding_size],
initializer=initializer())
            self.p1_embedded_chars =
tf.nn.embedding_lookup(self.W_pos, self.input_p1)
            self.p2_embedded_chars =
tf.nn.embedding_lookup(self.W_pos, self.input_p2)
            self.p1_embedded_chars_expanded =
tf.expand_dims(self.p1_embedded_chars, -1)
            self.p2_embedded_chars_expanded =
tf.expand_dims(self.p2_embedded_chars, -1)

        self.embedded_chars_expanded =
tf.concat([self.text_embedded_chars_expanded,

self.p1_embedded_chars_expanded,

self.p2_embedded_chars_expanded], 2)
        _embedding_size = text_embedding_size +
2*pos_embedding_size

        # Create a convolution + maxpool layer for each filter
size
        pooled_outputs = []
        for i, filter_size in enumerate(filter_sizes):
            with tf.variable_scope("conv-maxpool-%s" %
filter_size):
                # Convolution Layer
                conv =
tf.layers.conv2d(self.embedded_chars_expanded, num_filters,
[filter_size, _embedding_size],
```

```
kernel_initializer=initializer(), activation=tf.nn.relu,
name="conv")
                # Maxpooling over the outputs
                pooled = tf.nn.max_pool(conv, 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.variable_scope("dropout"):
            self.h_drop = tf.nn.dropout(self.h_pool_flat,
self.dropout_keep_prob)

        # Final scores and predictions
        with tf.variable_scope("output"):
            self.logits = tf.layers.dense(self.h_drop,
num_classes, kernel_initializer=initializer())
            self.predictions = tf.argmax(self.logits, 1,
name="predictions")

        # Calculate mean cross-entropy loss
        with tf.variable_scope("loss"):
            losses =
tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.logits,
labels=self.input_y)
            self.l2 = tf.add_n([tf.nn.l2_loss(v) for v in
tf.trainable_variables()])
            self.loss = tf.reduce_mean(losses) + l2_reg_lambda
* self.l2

        # 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, tf.float32),
name="accuracy")
```

Result

```
Evaluation:
2019-04-04T20:03:53.361078: step 12200, loss 1.85822, acc 0.690037
[UNOFFICIAL] (2*9+1)-Way Macro-Average F1 Score (excluding Other):
0.609236
```

First 10 Predictions about SemEval2010_task8

Prediction:

```
0    Message-Topic(e1,e2)
1    Product-Producer(e2,e1)
2    Instrument-Agency(e2,e1)
3    Entity-Destination(e1,e2)
4    Cause-Effect(e2,e1)
5    Component-Whole(e1,e2)
6    Product-Producer(e1,e2)
7    Member-Collection(e2,e1)
8    Component-Whole(e1,e2)
9    Message-Topic(e1,e2)
10   Entity-Destination(e1,e2)
```

Ground Truth

```
0    Message-Topic(e1,e2)
1    Product-Producer(e2,e1)
2    Instrument-Agency(e2,e1)
3    Entity-Destination(e1,e2)
4    Cause-Effect(e2,e1)
5    Component-Whole(e1,e2)
6    Product-Producer(e1,e2)
7    Member-Collection(e2,e1)
8    Component-Whole(e1,e2)
```

```
9    Message-Topic(e1,e2)
10   Entity-Destination(e1,e2)
```

Conclusion and Performance

Clearly, our Entity Attention Bi-LSTM model is better than TextCNN model.

With word-embedding glove.840B.300d we had our best model, and the highest F1-score is 83.96%.

With word-embedding glove.840B.300d:

```
Evaluation:
2019-04-04T22:48:58.192466: step 31500, loss 5.23648, acc 0.797967
<<< (9+1)-WAY EVALUATION TAKING DIRECTIONALITY INTO ACCOUNT -- OFFICIAL
>>>:
macro-averaged F1-score = 83.43%, Best = 83.96%
```
