TCN-CRF for NER

Anwen Hu

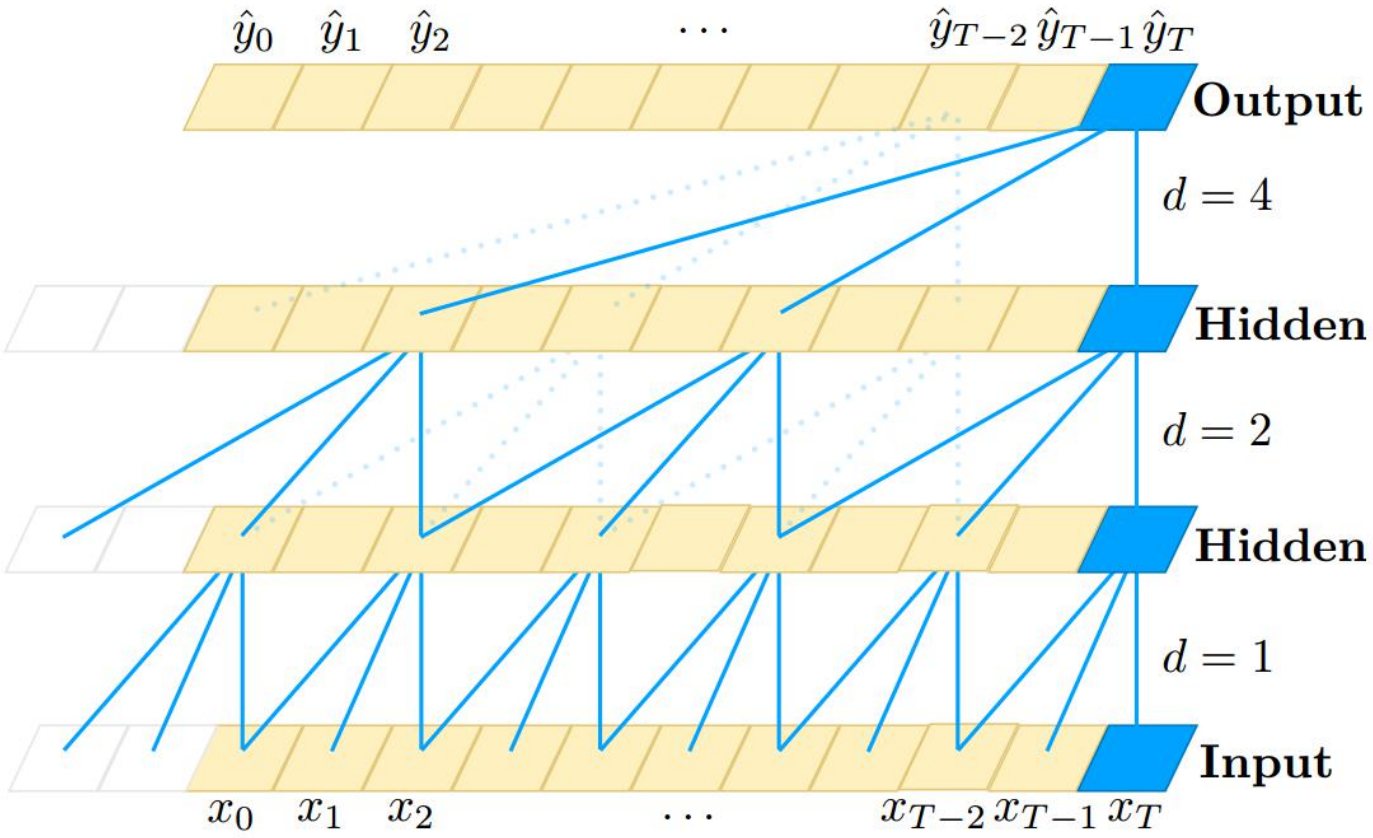
2018/06/12

TCN(Temporal Convolution Network)

Bai S, Kolter J Z, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling[J]. arXiv preprint arXiv:1803.01271, 2018.

- Dilated Convolution(Yu & Koltun 2016)
- Residual block(He et al.2016)

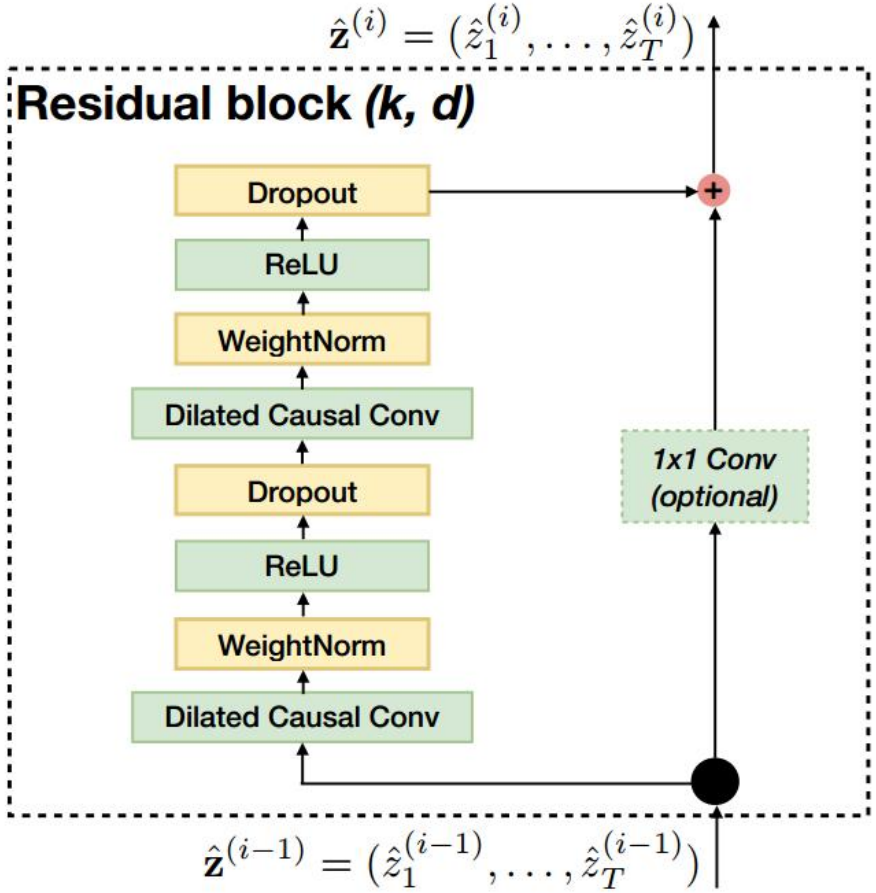
Dilated Convolution

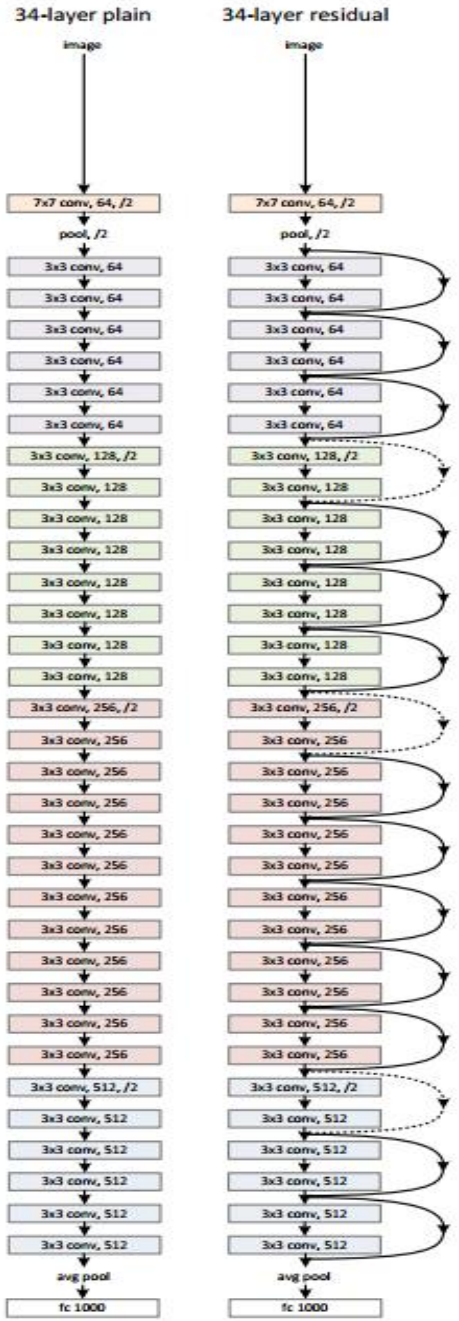


Dilated Convolution

```
relu1 = slim.conv2d(input, output_dim, [self.filter_size, 1], stride=1, padding='SAME', rate=dilation,
                    weights_initializer=tf.initializers.random_normal(mean=0, stddev=0.01),
                    weights_regularizer=contrib.layers.l1_regularizer(0.5),
                    scope="Residual_" + str(block_index) + "_conv1")
```

Residual Block





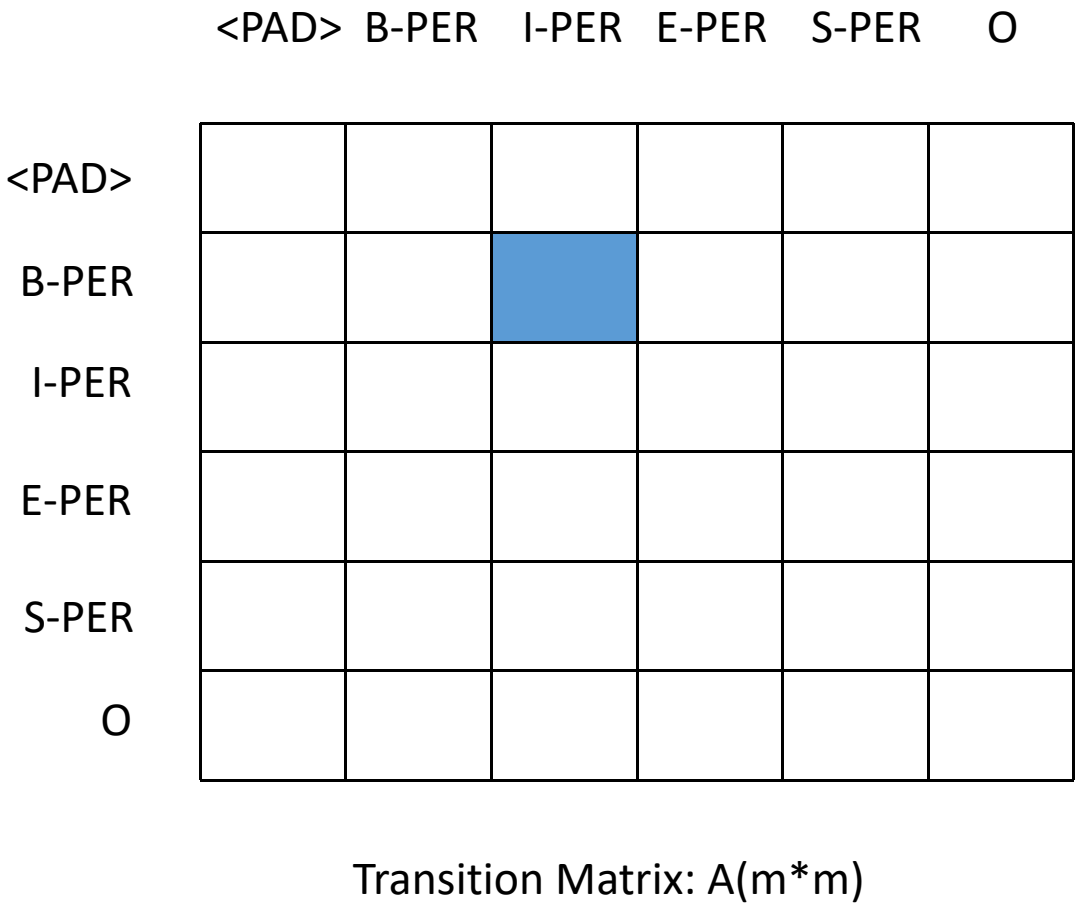
He et al.2016

Residual Block

```
if output_dim == input_dim:
    # identity map
    residual_output = dropout2 + input  # B*L*1*output_dim
else:
    # 1*1 convolution
    one_filter = tf.get_variable("Residual_" + str(block_index) + "_filter_1_1_weight",
                                 initializer=tf.random_normal(
                                     shape=[1, 1, input_dim, output_dim],
                                     mean=0, stddev=0.01, dtype=tf.float32))
    one_conv = tf.nn.conv2d(input, one_filter, strides=[1, 1, 1, 1], padding='SAME')
    one_bias = tf.get_variable("Residual_" + str(block_index) + "_one_bias", [output_dim],
                               initializer=tf.zeros_initializer)
    one_conv_b = tf.nn.bias_add(one_conv, one_bias)
    residual_output = dropout2 + one_conv_b  # B*L*1*output_dim
```

CRF

$$A_{12} = \text{Score}(B-PER => I-PER)$$



Transition Matrix: A(m*m)

CRF

```
self.tcn_pred = tf.reshape(self.tcn_pred, [-1, self.max_sent_len, label_num])
self.transition_matrix = tf.get_variable('transition', [self.label_num, self.label_num])
self.pedict_labels_batch, self.scores_batch = crf.crf_decode(self.tcn_pred,
                                                             self.transition_matrix, self.sents_len_batch)
self.log_likelihood, self.transition_matrix = crf.crf_log_likelihood(self.tcn_pred, self.labels_batch,
                                                                     self.sents_len_batch,
                                                                     self.transition_matrix)
```

TCN-CRF

words: $\mathrm{x} = (x_1, x_2, \ldots, x_n)$

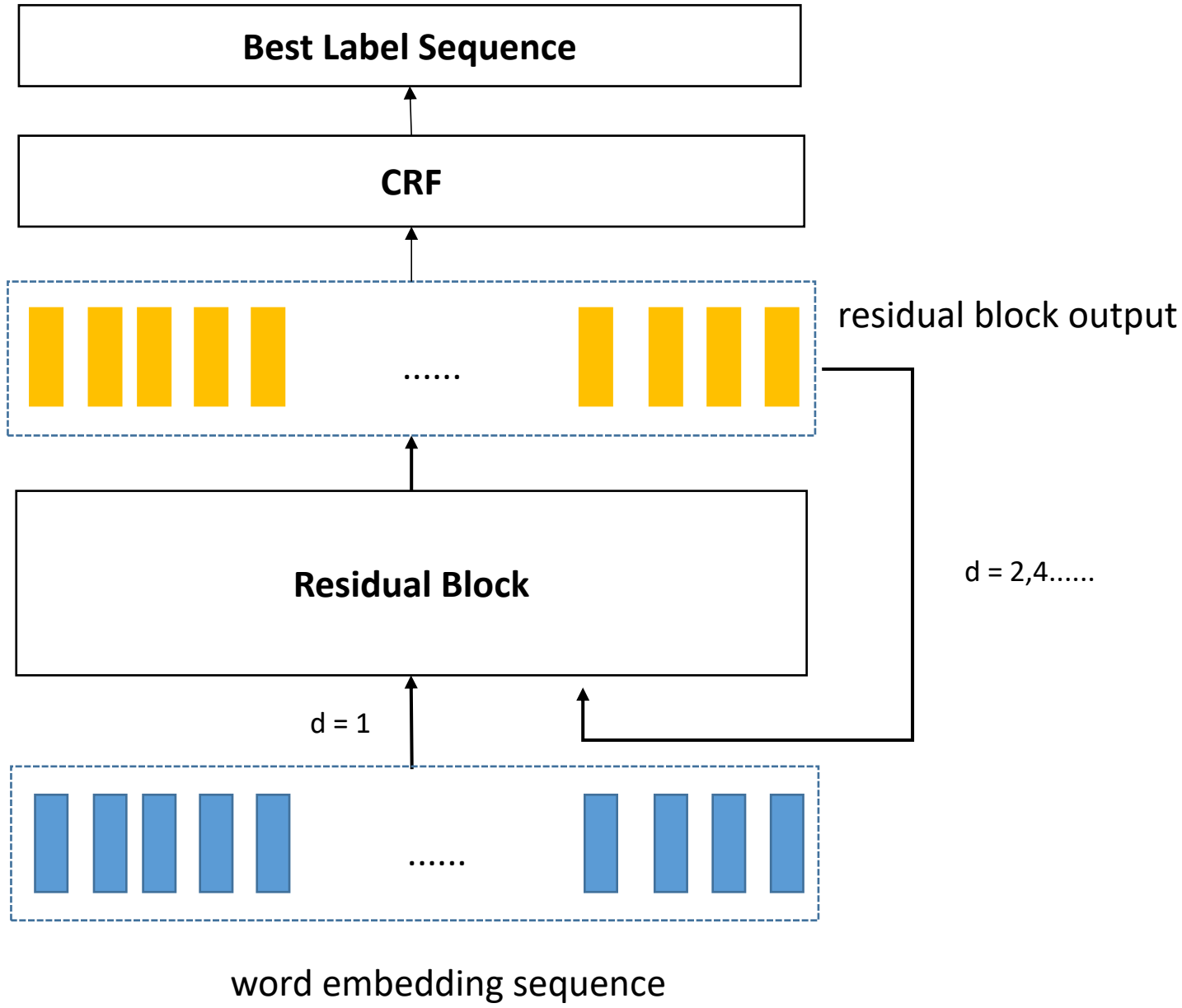
tags: $\mathrm{y} = (y_1, y_2, \ldots, y_n)$

$$s(\mathbf{y}, \mathbf{x}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

$$p(\mathbf{y}_{\text{true}} \mid \mathbf{x}) = \frac{e^{s(\mathbf{y}_{true}, \mathbf{x})}}{\sum_{\tilde{\mathbf{y}} \in Y} e^{s(\tilde{\mathbf{y}}, x)}}$$

$$\max \log(p(\mathbf{y}_{\text{true}} \mid \mathbf{x}))$$

$$\text{Loss} : -\log(p(\mathbf{y}_{\text{true}} \mid \mathbf{x}))$$



TCN-CRF

```
def Multiple_Residual_Block(self, input):
    for i in range(len(self.num_channels)):
        dilation = 2 ** i
        input_dim = self.word_emb_dim if i == 0 else self.num_channels[i - 1]
        output_dim = self.num_channels[i]
        residual_output = self.Residual_Block(input, input_dim, output_dim, dilation, block_index=i)
        input = residual_output
    return input
```

TCN-CRF

```
if emb_tag:
    self.embedding = tf.Variable(embedding_matrix, trainable=True, name="emb", dtype=tf.float32)
else:
    self.embedding = tf.get_variable("emb", [self.word_num + 1, self.word_emb_dim])
self.embeddings_batch = tf.nn.embedding_lookup(self.embedding, self.words_batch)  # B*L*word_dim
self.input = tf.reshape(self.embeddings_batch, [-1, self.max_sent_len, 1, self.word_emb_dim])  # B*L*1*word_dim
# multiple residual blocks
self.multi_residual_output = self.Multiple_Residual_Block(self.input)  # B*L*1*output_dim

# B*L*1*last_filter_num => (B*L)*last_filter_num(label_num)
self.residual_output = tf.reshape(self.multi_residual_output, [-1, self.tcn_output_dim])
self.tcn_pred = self.residual_output
if not crf_tag:
    # self.nerlabel_embedding_batch = tf.nn.embedding_lookup(self.nerlabels_matrix, self.labels_batch)
    self.nerlabel_one_hot_batch = tf.one_hot(self.labels_batch, self.label_num, on_value=1)
    self.loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        logits=self.tcn_pred, labels=tf.reshape(self.nerlabel_one_hot_batch, [-1, self.label_num])))
    self.optimizer = tf.train.AdamOptimizer(learning_rate=0.015).minimize(self.loss)
else:
    self.tcn_pred = tf.reshape(self.tcn_pred, [-1, self.max_sent_len, label_num])
    self.transition_matrix = tf.get_variable('transition', [self.label_num, self.label_num])
    self.pedict_labels_batch, self.scores_batch = crf.crf_decode(self.tcn_pred,
                                                                 self.transition_matrix, self.sents_len_batch)
    self.log_likelihood, self.transition_matrix = crf.crf_log_likelihood(self.tcn_pred, self.labels_batch,
                                                                         self.sents_len_batch,
                                                                         self.transition_matrix)

    self.loss = tf.reduce_mean(-self.log_likelihood)
    # control the gradient
    self.trainable_variables = tf.trainable_variables()
    self.grads, _ = tf.clip_by_global_norm(tf.gradients(self.loss, self.trainable_variables), 0.35)
    self.optimizer = tf.train.GradientDescentOptimizer(learning_rate=self.learning_rate).apply_gradients(
        zip(self.grads, self.trainable_variables)
    )
```

DataSet-Conll2003

- **From**: Reuters Corpora
- **Language**: English Language News stories
- **Date**: 1996-08-20 to 1997-08-19
- **The longest sentence** : 124 tokens
- **The longest document** : 1335 tokens
- **Entity**: PER,LOC,ORG,MISC
- **Taggging scheme**: BIO (=>BIOES)
  - e.g. (Welsh National Farmers ' Union)
    BIO: O B-ORG I-ORG I-ORG I-ORG O
    BIOES: O B-ORG I-ORG I-ORG E-ORG O

DataSet		CoNLL2003
Train	SENT DOC	14041 946
Dev	SENT DOC	3250 216
Test	SENT DOC	3453 231

Hyper-parameters

- **filter size**: k=5 / **residual blocks**: sent-level: i=4 doc-level:i=7

$$\text{len} = 2(2^i - 1)(k - 1) + 1$$

- **learning rate**:

  init:4.0 (annealed by a factor of 0.5 when validation loss plateaus)

- **batch size**: sent-level:8 doc-level:2
- **epoch** : 100
- **dropout**: 0.5
- **optimizer**: SGD
- **clip_norm**:global 0.35

Experiment

- sentence-level

BIO dev ; **PER** p:0.9304 ; r:0.9213 ; f1:0.9258
BIO dev ; **ORG** p:0.8708 ; r:0.8292 ; f1:0.8495
BIO dev ; **LOC** p:0.9474 ; r:0.9314 ; f1:0.9393
BIO dev ; **ALL** p:0.9212 ; r:0.8888 ; f1:0.9047
BIO dev ; **MISC** p:0.9213 ; r:0.8254 ; f1:0.8707

BIO test ; **PER** p:0.8958 ; r:0.8670 ; f1:0.8812
BIO test ; **ORG** p:0.8362 ; r:0.7652 ; f1:0.7991
BIO test ; **LOC** p:0.8814 ; r:0.9089 ; f1:0.8949
BIO test ; **ALL** p:0.8621 ; r:0.8313 ; f1:0.8464
BIO test ; **MISC** p:0.7894 ; r:0.7208 ; f1:0.7535

- document-level

BIO dev ; **LOC** p:0.9391 ; r:0.9314 ; f1:0.9352
BIO dev ; **ORG** p:0.8752 ; r:0.8262 ; f1:0.8500
BIO dev ; **PER** p:0.9264 ; r:0.9159 ; f1:0.9211
BIO dev ; **MISC** p:0.9023 ; r:0.8210 ; f1:0.8597
BIO dev ; **ALL** p:0.9156 ; r:0.8857 ; f1:0.9004

BIO test ; **LOC** p:0.8557 ; r:0.8993 ; f1:0.8769
BIO test ; **ORG** p:0.8181 ; r:0.7339 ; f1:0.7737
BIO test ; **PER** p:0.8978 ; r:0.8472 ; f1:0.8718
BIO test ; **MISC** p:0.7972 ; r:0.7279 ; f1:0.7610
BIO test ; **ALL** p:0.8503 ; r:0.8144 ; f1:0.8320

THANKS