【NLP基础】手把手带你fastText文本分类(附代码)

深度学习自然语言处理 3月10日

以下文章来源于NewBeeNLP ，作者kaiyuan

NewBeeNLP

一个自然语言处理&人工智能的原创杂货铺子，希望能找到你喜欢的小玩意儿

点击上方，选择**星标**或**置顶**，每天给你送干货🤔！

> 阅读大概需要6分钟🤤
> 跟随小博主，每天进步一丢丢😄

来自：NewBeeNLP

写在前面

今天的教程是基于FAIR的Bag of Tricks for Efficient Text Classification[1]。也就是我们常说的fastText。

最让人欣喜的这篇论文配套提供了fasttext工具包。这个工具包代码质量非常高，论文结果一键还原，目前已经是包装地非常专业了，这是fastText官网和其github代码库，以及提供了python接口，可以直接通过pip安装。这样准确率高又快的模型绝对是实战利器。

为了更好地理解fasttext原理，我们现在直接复现来一遍，但是代码中仅仅实现了最简单的基于单词的词向量求平均，并未使用b-gram的词向量，所以自己实现的文本分类效果会低于facebook开源的库。

论文概览

> We can train fastText on more than one billion words in less than ten minutes using a standard multicore CPU, and classify half a million sentences among 312K classes in less than a minute.

首先引用论文中的一段话来看看作者们是怎么评价fasttext模型的表现的。

这篇论文的模型非常之简单，之前了解过word2vec的同学可以发现这跟CBOW的模型框架非常相似。

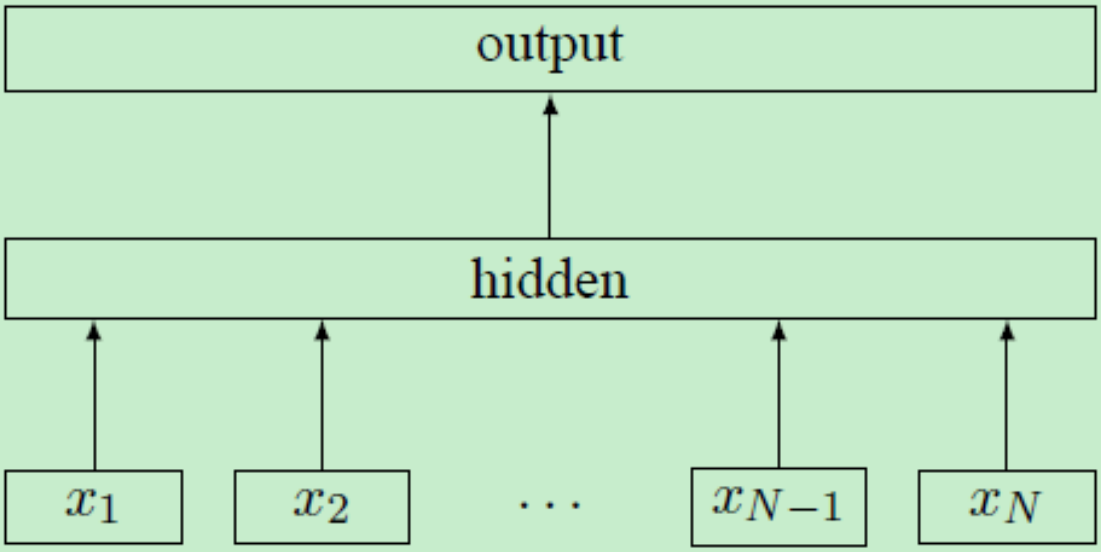

Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

对应上面这个模型，比如输入是一句话，x_1到x_n就是这句话的单词或者是n-gram。每一个都对应一个向量，然后对这些向量取平均就得到了文本向量，然后用这个平均向量取预测标签。当类别不多的时候，就是最简单的softmax；当标签数量巨大的时候，就要用到**「hierarchical softmax」**了。

模型真的很简单，也没什么可以说的了。下面提一下论文中的两个tricks：

- **「hierarchical softmax」**
  类别数较多时，通过构建一个霍夫曼编码树来加速softmax layer的计算，和之前word2vec中的trick相同
- **「N-gram features」**
  只用unigram的话会丢掉word order信息，所以通过加入N-gram features进行补充 用hashing来减少N-gram的存储

看了论文的实验部分，如此简单的模型竟然能取得这么好的效果 ！

但是也有人指出论文中选取的数据集都是对句子词序不是很敏感的数据集，所以得到文中的试验结果并不奇怪。

代码实现

看完阉割版代码大家记得去看看源码噢~ 跟之前系列的一样，定义一个 fastTextModel类 ，然后写网络框架，输入输出placeholder，损失，训练步骤等。

```
class fastTextModel(BaseModel):
    """
    A simple implementation of fasttext for text classification
    """
    def __init__(self, sequence_length, num_classes, vocab_size,
                 embedding_size, learning_rate, decay_steps, decay_rate,
                 l2_reg_lambda, is_training=True,
                 initializer=tf.random_normal_initializer(stddev=0.1)):
        self.vocab_size = vocab_size
        self.embedding_size = embedding_size
        self.num_classes = num_classes
        self.sequence_length = sequence_length
        self.learning_rate = learning_rate
        self.decay_steps = decay_steps
        self.decay_rate = decay_rate
        self.is_training = is_training
        self.l2_reg_lambda = l2_reg_lambda
        self.initializer = initializer

        self.input_x = tf.placeholder(tf.int32, [None, self.sequence_length], name='input_x')
        self.input_y = tf.placeholder(tf.int32, [None, self.num_classes], name='input_y')

        self.global_step = tf.Variable(0, trainable=False, name='global_step')
        self.instantiate_weight()
        self.logits = self.inference()
        self.loss_val = self.loss()
        self.train_op = self.train()

        self.predictions = tf.argmax(self.logits, axis=1, name='predictions')
        correct_prediction = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
        self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'), name='accuracy')

    def instantiate_weight(self):
        with tf.name_scope('weights'):
            self.Embedding = tf.get_variable('Embedding', shape=[self.vocab_size, self.embedd
                                             initializer=self.initializer)
            self.W_projection = tf.get_variable('W_projection', shape=[self.embedding_size, s
                                                initializer=self.initializer)
            self.b_projection = tf.get_variable('b_projection', shape=[self.num_classes])
```

```
    def inference(self):
        """
        1. word embedding
        2. average embedding
        3. linear classifier
        :return:
        """
        # embedding layer
        with tf.name_scope('embedding'):
            words_embedding = tf.nn.embedding_lookup(self.Embedding, self.input_x)
            self.average_embedding = tf.reduce_mean(words_embedding, axis=1)

        logits = tf.matmul(self.average_embedding, self.W_projection) +self.b_projection

        return logits


    def loss(self):
        # loss
        with tf.name_scope('loss'):
            losses = tf.nn.softmax_cross_entropy_with_logits(labels=self.input_y, logits=self
            data_loss = tf.reduce_mean(losses)
            l2_loss = tf.add_n([tf.nn.l2_loss(cand_var) for cand_var in tf.trainable_variable
                                if'bias'notin cand_var.name]) * self.l2_reg_lambda
            data_loss += l2_loss * self.l2_reg_lambda
            return data_loss

    def train(self):
        with tf.name_scope('train'):
            learning_rate = tf.train.exponential_decay(self.learning_rate, self.global_step,
                                                       self.decay_steps, self.decay_rate,
                                                       staircase=True)

            train_op = tf.contrib.layers.optimize_loss(self.loss_val, global_step=self.global
                                                      learning_rate=learning_rate, optimizer=

        return train_op
```

```
def prepocess():
    """
    For load and process data
    :return:
```

```
    """
    print("Loading data...")
    x_text, y = data_process.load_data_and_labels(FLAGS.positive_data_file, FLAGS.negative_da
    # bulid vocabulary
    max_document_length = max(len(x.split(' ')) for x in x_text)
    vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
    x = np.array(list(vocab_processor.fit_transform(x_text)))

    # shuffle
    np.random.seed(10)
    shuffle_indices = np.random.permutation(np.arange(len(y)))
    x_shuffled = x[shuffle_indices]
    y_shuffled = y[shuffle_indices]

    # split train/test dataset
    dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
    x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
    y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]
    del x, y, x_shuffled, y_shuffled

    print('Vocabulary Size: {:d}'.format(len(vocab_processor.vocabulary_)))
    print('Train/Dev split: {:d}/{:d}'.format(len(y_train), len(y_dev)))
    return x_train, y_train, vocab_processor, x_dev, y_dev


def train(x_train, y_train, vocab_processor, x_dev, y_dev):
    with tf.Graph().as_default():
        session_conf = tf.ConfigProto(
            # allows TensorFlow to fall back on a device with a certain operation implemented
            allow_soft_placement= FLAGS.allow_soft_placement,
            # allows TensorFlow log on which devices (CPU or GPU) it places operations
            log_device_placement=FLAGS.log_device_placement
        )
        sess = tf.Session(config=session_conf)
        with sess.as_default():
            # initialize cnn
            fasttext = fastTextModel(sequence_length=x_train.shape[1],
                      num_classes=y_train.shape[1],
                      vocab_size=len(vocab_processor.vocabulary_),
                      embedding_size=FLAGS.embedding_size,
                      l2_reg_lambda=FLAGS.l2_reg_lambda,
                      is_training=True,
                      learning_rate=FLAGS.learning_rate,
                      decay_steps=FLAGS.decay_steps,
                      decay_rate=FLAGS.decay_rate
                     )
```

```
# output dir for models and summaries
timestamp = str(time.time())
out_dir = os.path.abspath(os.path.join(os.path.curdir, 'run', timestamp))
ifnot os.path.exists(out_dir):
    os.makedirs(out_dir)
print('Writing to {} \n'.format(out_dir))

# checkpoint dir. checkpointing - saving the parameters of your model to restore
checkpoint_dir = os.path.abspath(os.path.join(out_dir, FLAGS.ckpt_dir))
checkpoint_prefix = os.path.join(checkpoint_dir, 'model')
ifnot os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
saver = tf.train.Saver(tf.global_variables(), max_to_keep=FLAGS.num_checkpoints)

# Write vocabulary
vocab_processor.save(os.path.join(out_dir, 'vocab'))

# Initialize all
sess.run(tf.global_variables_initializer())


def train_step(x_batch, y_batch):
    """
    A single training step
    :param x_batch:
    :param y_batch:
    :return:
    """
    feed_dict = {
        fasttext.input_x: x_batch,
        fasttext.input_y: y_batch,
    }
    _, step, loss, accuracy = sess.run(
        [fasttext.train_op, fasttext.global_step, fasttext.loss_val, fasttext.acc
        feed_dict=feed_dict
    )
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accurac

def dev_step(x_batch, y_batch):
    """
    Evaluate model on a dev set
    Disable dropout
    :param x_batch:
    :param y_batch:
```

```
            :param writer:
            :return:
            """
            feed_dict = {
                fasttext.input_x: x_batch,
                fasttext.input_y: y_batch,
            }
            step, loss, accuracy = sess.run(
                [fasttext.global_step, fasttext.loss_val, fasttext.accuracy],
                feed_dict=feed_dict
            )
            time_str = datetime.datetime.now().isoformat()
            print("dev results:{}: step {}, loss {:g}, acc {:g}".format(time_str, step, l

        # generate batches
        batches = data_process.batch_iter(list(zip(x_train, y_train)), FLAGS.batch_size,
        # training loop
        for batch in batches:
            x_batch, y_batch = zip(*batch)
            train_step(x_batch, y_batch)
            current_step = tf.train.global_step(sess, fasttext.global_step)
            if current_step % FLAGS.validate_every == 0:
                print('\n Evaluation:')
                dev_step(x_dev, y_dev)
                print('')

        path = saver.save(sess, checkpoint_prefix, global_step=current_step)
        print('Save model checkpoint to {} \n'.format(path))

def main(argv=None):
    x_train, y_train, vocab_processor, x_dev, y_dev = prepocess()
    train(x_train, y_train, vocab_processor, x_dev, y_dev)

if __name__ == '__main__':
    tf.app.run()
```

对啦，我这里使用的数据集还是之前训练CNN时的那一份

本文参考资料

[1] **Bag of Tricks for Efficient Text Classification:** https://arxiv.org/abs/1607.01759