风速预测-LSTM模型

zty，V1.0

1. API

1.1. 预处理生成的数据格式

文件命名

预处理后生成四个文件，分别用 “train_x.dat”，“train_y.dat”，“test_x.dat”，“test_y.dat” 命名。

train_x 和 **test_x** 的格式

train_x 和 test_x 分别是 numpy 三维数组，形如

```
train_x = np.zeros((train_size_, seq_len_, x_dim_))
```

其中，第一个参数表示有 train_size_ 个样本点（每个样本点表示，比如，用于预测12日12点风速的 x ）；

第二个参数表示每个样本点有 seq_len_ 个时间序列点（每个时序点表示，比如，用于预测12日12点风速的 x 中的第某个时刻数据，比如12日11点15分的数据）；

第三个参数表示每个时间序列点的数据是一个 x_dim_ 维的向量（比如12日11点15分的数据由18台风机的风速组成，则维数为18）。

train_y 和 **test_y** 格式

train_y 和 test_y 分别是二位数组，形如

```
train_y = np.zeros((train_size_, y_dim_))
```

其中，第一个参数表示有 train_size_ 个样本点（每个样本点表示，比如，12日12点风速的 ground truth : y）；train_x[i] 应与 train_y[i] 对应；

第二个参数表示每个时间序列点的数据是一个 y_dim_ 维的向量（比如12日12点的 ground truth数据由6台风机的风速组成，则维数为6）。

用 **numpy.tofile** 写文件

```
train_x.tofile('./train_x.dat')
```

1.2. 模型的训练和测试方法

指定数据格式参数

train_size，test_size，x_dim，y_dim，seq_len 需要与数据格式吻合。

指定训练模式

toPlot = 'y'，表示画学习曲线，在不支持可视化的设备上训练时请将本变量设为'n'，否则将发生错误；

toRestore = 'y'，表示用存储好的上次训练后的模型参数继续优化，模型结构更换后请将本变量设为'n'，否则将发生错误。

调整网络结构和训练算法超参数

调整超参数 batch_size，num_iters，learning_rate，h_size 以得到最优训练效果。

训练

在命令行用 $python2.7$ 或 $python3.6$ 运行程序即可，请不要使用任何 IDE 运行代码，IDE 相比命令行都是垃圾，可能造成各种各样意想不到的奇怪错误。

运行的机器上应该装有 $python$，$anaconda$，$tensorflow$ 等依赖关系库。深度学习库间的依赖关系较为复杂，任何版本不协调都可能造成各种各样的奇怪错误，请使用者耐心调试。

2. 模型和代码说明

我们使用一个 LSTM （long-short term memory） 模型进行学习。

LSTM 的简介

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

使用的一些优化技巧

L2 正则化：

```
# regularization
def variable_with_weight_loss(shape, stddev, wl):
    var = tf.Variable(tf.truncated_normal(shape, stddev = stddev))
    if(wl is not None):
        weight_loss = tf.multiply(tf.nn.l2_loss(var), wl, name = 'weight_loss')
        tf.add_to_collection('losses', weight_loss)
    return var

# regularized loss
def regularized_loss(loss):
    tf.add_to_collection('losses', loss)
    return tf.add_n(tf.get_collection('losses'), name = 'total_loss')

def Graph(trainset, testset):
    # ...
    losses = tf.abs(ypred - Y)
    total_loss = tf.reduce_mean(losses)
    total_loss = regularized_loss(total_loss)
    # ...
```

gradient_clipping：

Deep Learning，Bengio，367页。

```
def Graph(trainset, testset):
    # ...
    # gradient clipping
    grads_and_vars = train_step.compute_gradients(total_loss, [Wf, bf, Wi, bi, Wc, bc,
Wo, bo, Wout, bout])
    capped_grads_and_vars = [(tf.clip_by_norm(gv[0], 5), gv[1]) for gv in
grads_and_vars]
    train_step.apply_gradients(capped_grads_and_vars)
    train_step = train_step.minimize(total_loss)
    # ...
```

代码实现

minibatch SGD：

```
class dataset:
    # ...
    def shuffle(self):
        # ...
    def get_batch(self):
        start = self.idx
        self.idx += batch_size_
        if self.idx > self.size:
            self.shuffle()
            start = 0
            self.idx = batch_size_
        end = self.idx
        # seperate into different slices
        batch_x = []
        batch_y = []
        for i in range(start, end):
            batch_x.append(self.set[i].x)
            batch_y.append(self.set[i].y)
        return(batch_x, batch_y)
```

LSTM：

```
def Graph(trainset, testset):
    # ...
    # Forward pass for training
    current_h = init_h
    current_c = init_c
    for i in range(seq_len_):
        # input
        current_input = tf.reshape(input_series[i], [batch_size_, x_dim_])

        # LSTM
        h_concat_x = tf.concat([current_input, current_h], 1)
        f = tf.nn.sigmoid(tf.matmul(h_concat_x, Wf) + bf)
        i = tf.nn.sigmoid(tf.matmul(h_concat_x, Wi) + bi)
        c = tf.tanh(tf.matmul(h_concat_x, Wc) + bc)
        next_c = tf.multiply(current_c, f) + tf.multiply(c, i)
        o = tf.nn.sigmoid(tf.matmul(h_concat_x, Wo) + bo)
        next_h = tf.multiply(tf.tanh(next_c), o)

        # renew
        current_h = next_h
        current_c = next_c

    # output given output h
    ypred = tf.matmul(current_h, Wout) + bout
    # ...
```