Weekly Report of Research Work

WR-ABS-TEMP-2015A-No.010

汤吉(Ji TANG)

Number: WR-ABS-TEMP-2015A, E-mail: tangji08@hotmail.com

Date: 25/1/2016 - 31/1/2016

January 31, 2016

Contents

1 Work

1. Translating the paper
2. Learning the concept of Deep Learning
3. Downloading the code for Deep Learning and analysing them
4. Studying about the logistic regression

2 Translation of the paper

Hurst指数和金融市场预测

Bo Qian, Khaled Rasheed
计算机科学系，佐治亚大学
Athens, GA 30601
USA
mailto:khaled]@cs.uga.edu

摘要

Hurst指数(H)是一个统计学测量用来分类时间序列。当H=0.5时，表示一个完全随机的序列。而当H>0.5时，表示了一个具有保持趋势倾向能力的序列。H的值越大，这个序列的倾向也越强。我们接下来将要研究如何利用Hurst指数来将不同时期的金融序列数据进行分类。BP神经网络的实验表明，具有高Hurst指数的序列比那些Hurst指数接近于0.50的序列能够被更加精确的预测。因此，Hurst指数提供了一种预测方法。

关键词 Hurst指数，时间序列分析，神经网络，蒙特卡诺模拟，预测

I．介绍 Hurst指数是H. E. Hurst提出用来作分形分析的，现在已经被用在许多研究领域。最近，由于Peter的相关工作，它在金融领域也变的十分热门。Hurst指数为长期记忆和时间序列的分形提供了一种方法。由于它是高鲁棒性的基本系统的几个假设，现在已经被广泛用于时间序列分析。Hurst指数的值在0和1之间。基于Hurst指数H，一个时间序列能够被分为三种类型：(1)H=0.5表明了序列可以用随机游走来描述。(2)0<H<0.5表明了序列具有反持续性。(3)0.5<H<1表明序列具有持续性。一个反持续性序列具有均值回复的特性，即意味着一个上升的值更有可能紧接着一个下降的值，反之亦然。H的值越接近于0.0，序列均值回复的能力也越强。而一个持续性序列具有保持倾向的能力，即下一时刻的值相对于现在值的变化，更有可能与这一时刻相对于上一时刻值的变化一致。H的值越接近于1.0，序列保持倾向的能力也越强。大多数的经济和金融时间序列具有持续性，即H>0.5。

在时间序列的预测当中，我们首先需要解决的问题是我们想要研究的这个时间序列是否可以被预测。如果这个时间序列是随机的，一切的方法都是无效的。我们想要确定这些序列具

有一定的可预测等级。我们知道一个具有很高H值的时间序列是具有很强的倾向性的，所以我们自然地认为这样的时间序列要比那些H值接近于0.5的时间序列更可能被预测。接下来，我们将要使用神经网络来测试这个假设。

神经网络是无参数的通用函数逼近，可以无假设地从数据中进行学习。在过去的十年里，神经网络预测模型已经被广泛应用于金融时间序列分析。神经网络可以被用来代替通用函数逼近，进行预测。在同样的条件下，一个时间序列如果比另外一个时间序列具有更小的预测误差，我们便说它更容易被预测。从1930年1月2日到2004年5月14日，我们研究每日的道琼斯指数，计算每1024交易日的Hurst指数。从当中选出30个具有最大的Hurst指数与30个Hurst指数接近于随机序列的周期，然后我们用这些数据来训练我们的神经网络。我们对比这两组数据的预测误差，发现他们的预测误差完全不同。这个研究是通过Matlab来实现的，这篇文章所有的Matlab程序生成的结果都可以从www.arches.uga.edu/ qianbo/research下载。

3 The concept of Deep Learning

3.1 Definitions

There are a number of ways that the field of deep learning has been characterized. Deep learning is a class of machine learning algorithms that use a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised). are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation. are part of the broader machine learning field of learning representations of data. learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

These definitions have in common (1) multiple layers of nonlinear processing units and (2) the supervised or unsupervised learning of feature representations in each layer, with the layers forming a hierarchy from low-level to high-level features. The composition of a layer of nonlinear processing units used in a deep learning algorithm depends on the problem to be solved. Layers that have been used in deep learning include hidden layers of an artificial neural network and sets of complicated propositional formulas. They may also include latent variables organized layer-wise in deep generative models such as the nodes in Deep Belief Networks and Deep Boltzmann Machines.

Deep learning algorithms are contrasted with shallow learning algorithms by the number of parameterized transformations a signal encounters as it propagates from the input layer to the output layer, where a parameterized transformation is a processing unit that has trainable parameters, such as weights and thresholds. A chain of transformations from input to output is a credit assignment path (CAP). CAPs describe potentially causal connections between input and output and may vary in length. For a feedforward neural network, the depth of the CAPs, and thus the depth of the network, is the number of hidden layers plus one (the output layer is also parameterized). For recurrent neural networks, in which a signal may propagate through a layer more than once, the CAP is potentially unlimited in length. There is no universally agreed upon threshold of depth dividing shallow learning from deep learning, but most researchers in the field agree that deep learning has multiple nonlinear layers (CAP > 2) and Schmidhuber considers CAP > 10 to be very deep learning.

3.2 Fundamental concepts

Deep learning algorithms are based on distributed representations. The underlying assumption behind distributed representations is that observed data is generated by the interactions of factors organized in layers. Deep learning adds the assumption that these layers of factors correspond to levels of abstraction or composition. Varying numbers of layers and layer sizes can be used to provide different amounts of abstraction.

Deep learning exploits this idea of hierarchical explanatory factors where higher level, more abstract concepts being learned from the lower level ones. These architectures are often constructed with a greedy layer-by-layer method. Deep learning helps to disentangle these abstractions and pick out which features are useful for learning.

For supervised learning tasks, deep learning methods obviate feature engineering, by translating the data into compact intermediate representations akin to principal components, and derive layered structures which remove redundancy in representation.

Many deep learning algorithms are applied to unsupervised learning tasks. This is an important benefit because unlabeled data is usually more abundant than labeled data. An example of a deep structure that can be trained in an unsupervised manner is a deep belief network.

4 The code for logistic regression

```
"""
This tutorial introduces logistic regression using Theano and stochastic
gradient descent.

Logistic regression is a probabilistic, linear classifier. It is parametrized
by a weight matrix :math:W and a bias vector :math:b. Classification is
```

```
done by projecting data points onto a set of hyperplanes, the distance to
which is used to determine a class membership probability.

Mathematically, this can be written as:

.. math::
  P(Y=i|x, W,b) &= softmax_i(W x + b) \\
                &= \frac {e^{W_i x + b_i}} {\sum_j e^{W_j x + b_j}}


The output of the model or prediction is then done by taking the argmax of
the vector whose i'th element is P(Y=i|x).

.. math::

  y_{pred} = argmax_i P(Y=i|x,W,b)


This tutorial presents a stochastic gradient descent optimization method
suitable for large datasets.


References:

    - textbooks: "Pattern Recognition and Machine Learning" -
                 Christopher M. Bishop, section 4.3.2

"""
__docformat__ = 'restructedtext en'

"""
import ...


class LogisticRegression(object):
    """Multi-class Logistic Regression Class

    The logistic regression is fully described by a weight matrix :math:W
    and bias vector :math:b. Classification is done by projecting data
    points onto a set of hyperplanes, the distance to which is used to
    determine a class membership probability.
    """

    def negative_log_likelihood(self, y):
        """Return the mean of the negative log-likelihood of the prediction
        of this model under a given target distribution.

        .. math::

            \frac{1}{|\mathcal{D}|} \mathcal{L} (\theta=\{W,b\}, \mathcal{D}) =
            \frac{1}{|\mathcal{D}|} \sum_{i=0}^{|\mathcal{D}|}
                \log(P(Y=y^{(i)}|x^{(i)}, W,b)) \\
            \ell (\theta=\{W,b\}, \mathcal{D})

        :type y: theano.tensor.TensorType
        :param y: corresponds to a vector that gives for each example the
                  correct label

        Note: we use the mean instead of the sum so that
              the learning rate is less dependent on the batch size
        """
        # start-snippet-2
        # y.shape[0] is (symbolically) the number of rows in y, i.e.,
        # number of examples (call it n) in the minibatch
        # T.arange(y.shape[0]) is a symbolic vector which will contain
        # [0,1,2,... n-1] T.log(self.p_y_given_x) is a matrix of
        # Log-Probabilities (call it LP) with one row per example and
        # one column per class LP[T.arange(y.shape[0]),y] is a vector
        # v containing [LP[0,y[0]], LP[1,y[1]], LP[2,y[2]], ...,
        # LP[n-1,y[n-1]]] and T.mean(LP[T.arange(y.shape[0]),y]) is
        # the mean (across minibatch examples) of the elements in v,
        # i.e., the mean log-likelihood across the minibatch.
        return -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]), y])
        # end-snippet-2

    def errors(self, y):
        """Return a float representing the number of errors in the minibatch
        over the total number of examples of the minibatch ; zero one
        loss over the size of the minibatch

        :type y: theano.tensor.TensorType
        :param y: corresponds to a vector that gives for each example the
                  correct label
        """
```

```

def load_data(dataset):
    ''' Loads the dataset

    :type dataset: string
    :param dataset: the path to the dataset (here MNIST)
    '''

    #############
    # LOAD DATA #
    #############

    # Download the MNIST dataset if it is not present
    data_dir, data_file = os.path.split(dataset)
    if data_dir == "" and not os.path.isfile(dataset):
        # Check if dataset is in the data directory.
        new_path = os.path.join(
            os.path.split(__file__)[0],
            "..",
            "data",
            dataset
        )
        if os.path.isfile(new_path) or data_file == 'mnist.pkl.gz':
            dataset = new_path

    if (not os.path.isfile(dataset)) and data_file == 'mnist.pkl.gz':
        import urllib
        origin = (
            'http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz'
        )
        print 'Downloading data from %s' % origin
        urllib.urlretrieve(origin, dataset)

    print '... loading data'

    # Load the dataset
    f = gzip.open(dataset, 'rb')
    train_set, valid_set, test_set = cPickle.load(f)
    f.close()
    #train_set, valid_set, test_set format: tuple(input, target)
    #input is an numpy.ndarray of 2 dimensions (a matrix)
    #witch row's correspond to an example. target is a
    #numpy.ndarray of 1 dimensions (vector)) that have the same length as
    #the number of rows in the input. It should give the target
    #target to the example with the same index in the input.

    def shared_dataset(data_xy, borrow=True):
        """ Function that loads the dataset into shared variables

        The reason we store our dataset in shared variables is to allow
        Theano to copy it into the GPU memory (when code is run on GPU).
        Since copying data into the GPU is slow, copying a minibatch everytime
        is needed (the default behaviour if the data is not in a shared
        variable) would lead to a large decrease in performance.
        """

def sgd_optimization_mnist(learning_rate=0.13, n_epochs=1000,
                           dataset='mnist.pkl.gz',
                           batch_size=600):
    """
    Demonstrate stochastic gradient descent optimization of a log-linear
    model

    This is demonstrated on MNIST.

    :type learning_rate: float
    :param learning_rate: learning rate used (factor for the stochastic
                          gradient)

    :type n_epochs: int
    :param n_epochs: maximal number of epochs to run the optimizer

    :type dataset: string
    :param dataset: the path of the MNIST dataset file from
                 http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz

    """
    ######################
    # BUILD ACTUAL MODEL #
    ######################
    print '... building the model'

    ###############
    # TRAIN MODEL #
    ################
    print '... training the model'
    # early-stopping parameters

def predict():
    """
    An example of how to load a trained model and use it
    to predict labels.
    """

    # load the saved model
    classifier = cPickle.load(open('best_model.pkl'))

    # compile a predictor function
    predict_model = theano.function(
        inputs=[classifier.input],
        outputs=classifier.y_pred)

    # We can test it on some examples from test test
    dataset='mnist.pkl.gz'
    datasets = load_data(dataset)
    test_set_x, test_set_y = datasets[2]
    test_set_x = test_set_x.get_value()

    predicted_values = predict_model(test_set_x[:10])
    print ("Predicted values for the first 10 examples in test set:")
    print predicted_values


if __name__ == '__main__':
    sgd_optimization_mnist()
```

5 My feeling about the codes

The codes I written always not have many notes. But the codes I downloaded from Internet are always with so many notes, which makes me feel more comfortable to read the codes