NN Splice algorithm.
Implementation and testing

31.05.2016

Introduction

Gene prediction is the process of finding the location of genes and other meaningful subsequences in DNA sequences. This process is time consuming and expensive when done by biochemical methods and genetics. There is existed another approach that makes genes to be predicted by analysing the sequence of nucleotides in the DNA using a statistical method. Then the process is carried out in a computer system, which is faster and less expensive.
The DNA molecule contains subsequences that codes protein chains. The proteins form the functionality of the organism. The subsequences that code to these proteins are called genes. In eukaryotic cells, the gene sequences consist of exons and introns. The exon part is coding to proteins while the intron part is rejected in the splicing process. The transition between exon and intron, and the transitions between intron and exon are called *donor splice site* and a*cceptor splice site*, respectively. The discussed algorithm tries to predict the splice sites in the genes. As a function approximator artificial neural network is used. It is connected to a gene sequence and the system tries to predict the slice sites in the sequence.

The purpose of the project

The purpose of the project is to implement and test the Neural Network Splice algorithm with k-fold cross-validation. To evaluate the successes of obtained model there was computed such statistical measures as accuracy, recall, sensitivity, specificity. The Python language (v 2.7) is used for reaching the goal.

Description of the Neural Network algorithm

Artificial neural network is a mathematical method known from artificial intelligence and pattern recognition. Here the neural network algorithm with backpropagation presented. It iteratively learns a set of weights for prediction of the class label of tuples. A multilayer feed-forward neural network consists of an input layer, one hiddenlayer, and an output layer.

Each layer is made up of units. The inputs to the network correspond to the attributes measured for each training sample. The inputs are fed simultaneously into the units making up the input layer. These inputs pass through the input layer and are then weighted and fed simultaneously to a second, hidden, layer. The weighted outputs of the hidden layer are input to units making up the output layer, which emits the network's prediction for given sample. Here there is only one hidden layer, hence that it is a two-layer neural network.

Each output unit takes, as input, a weighted sum of the outputs from units in the previous layer. It applies a non-linear (activation) function to the weighted input. Multilayer feed-forward neural networks are able to model the class prediction as a non-linear combination of the inputs. From a statistical point of view, they perform non-linear regression.

As it was mentioned ANN implemented using backpropagation algorithm. Backpropagation learns by iteratively processing a data set of training samples, comparing the network's prediction for each sample with the actual known target value. The target value is the known class label of the training sample (0 or 1). For each training observation, the weights are modified so as to minimize the mean-squared error between the network's prediction and the actual target value. The algorithm is summarized below.

```
Initialize all weights and biases in neural network;
while (two consecutive accuracies $\leq$ current accuracy && a pre-specified number of epochs has not expired) {
    for each training sample in the given training set {
        for each hidden or output layer unit $j$ {
            // Compute the net input of unit $j$ with respect to the previous layer $i$
            $I_j = \sum_i w_{ij} O_i + \theta_j$;
            // Compute the output of each unit $j$, i.e. apply activation function to it
            $O_j = \frac{1}{1+e^{-I_j}}$};
        for each unit $j$ in the output layer
            $Err_j = O_j(1 - O_j)(T_j - O_j)$; //Compute the error
        for each unit $j$ in the output layer
            // Compute the error with respect to the next higher layer
            $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$;
        for each weight $w_{ij} in$neural network {
            $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
            $w_{ij} = w_{ij} + \Delta w_{ij}$;} // weight update
        for each bias $\theta_j in$neural network {
            $\Delta \theta_j = (l) Err_j$; // bias increment
            $\theta_j = \theta_j + \Delta \theta_j$;} // bias update
    }
}
```

Learning rate is computed after each epoch:
$l = \frac{l_0}{1+t/N}$,
l_0 is the initial value of learning rate;
t is the current epoch;
N is the number of samples in the dataset.

The briefly description of each step is represented below:
Initialize the weights: The weights in the network are initialized to small random numbers, ranging from 1.0 to 1.0. Each unit has a bias associated with it. The biases are similarly initialized to small random numbers.

Each training sample, X, is processed by the following steps.
Propagate the inputs forward: First, the training sample is fed to the network's input layer. The inputs pass through the input units, unchanged. That is, for an input unit, j, its output, O_j , is equal to its input value, I_j . Next, the net input and output of each unit in the hidden and output layers are computed. The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs. Each such unit has a number of inputs to it that are, in fact, the outputs of the units connected to it in the previous layer. Each connection has a weight. To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit, j in a hidden or output layer, the net input, I_j , to unit j is

$$I_j = \sum_i w_{ij} O_i + \theta_j,$$

where w_{ij} is the weight of the connection from unit i in the previous layer to unit j; O_i is the output of unit i from the previous layer; and $theta_j$ is the bias of the unit. The bias acts as a threshold in that it serves to vary the activity of the unit.

Each unit in the hidden and output layers takes its net input and then applies an activation function to it. The function symbolizes the activation of the neuron represented by the unit. The logistic, or sigmoid, function is used. Given the net input I_j to unit j, then O_j, the output of unit j, is computed as

$$O_j = \frac{1}{1+e^{-I_j}}.$$

Backpropagate the error: The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction. For a unit j in the output layer, the error Err_j is computed by

$$Err_j = O_j(1 - O_j)(T_j - O_j),$$

where O_j is the actual output of unit j, and T_j is the known target value of the given training sample. $O_j(1 - O_j)$ is the derivative of the logistic function.
To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk},$$

where w_{jk} is the weight of the connection from unit j to a unit k in the next output layer, and Err_k is the error of unit k.

The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where Δw_{ij} is the change in weight $w_i j$:

$$\Delta w_{ij} = (l) Err_j O_i,$$
$$w_{ij} = w_{ij} + \Delta w_{ij}.$$

Learning rate: The variable l is the learning rate, a constant having a value between 0.0 and 1.0. Backpropagation learns using a gradient descent method to search for a set of weights that fits the training data so as to minimize the meansquared distance between the network's class prediction and the known target value of the samples. The learning rate helps avoid getting stuck at a local minimum in decision space (i.e., where the weights appear to converge, but are not the optimum solution) and encourages finding the global minimum. If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur.
Biases are updated by the following equations, where $\Delta Theta_j$ is the change in bias Θ_j :

$$\Delta\theta_j = (l) Err_j,\ \theta_j = \theta_j + \Delta\theta_j.$$

Here we are updating the weights and biases after the presentation of each sample. This is referred to as case updating. This strategy is chosen because it tends to yield more accurate results.
Terminating condition: Training stops when

- A pre-specified number of epochs has expired;
- Two consecutive worse prediction coefficients are obtained.

The neural network will evaluate the pattern in the window of n nucleotides slides to predict if there are any splices sites in this pattern. The output of this evaluation is a numeric value, and the values are accumulated for each nucleotide as the window slides over the gene. The accumulated score is used as an indicator of where the splice sites are located.

K-Fold Cross-Validation for Neural Networks

Because neural networks are universal function approximators, given enough time, it is always possible (in theory) to find a set of weights and biases so that computed outputs exactly match training data outputs. But if you use those weights and bias values on new, previously unseen data, your neural network will predict very poorly. This is called over-fitting.

So the problem is over-fitting. There are many ways to deal with over-fitting. K-fold cross validation is one. The idea is to break the training data into k subsets, where k is 10 was taken. Then run the training algorithm 10 times. On the first training run we use the 9/10 of the training data to train, and then compute the network's accuracy using the 1/10 of the remaining data. This process is repeated, so that each 1/10 subset is used exactly once as the validation set. When finished we take the average of the 10 accuracies and use it as the overall estimate of the accuracy of the network. In short, k-fold cross-validation gives an estimate of a neural network's accuracy when the network was constructed using particular values for number of hidden nodes and training parameters.

After we do k-fold cross-validation repeatedly, and during the training phase use different values for the training technique's parameters and also try different numbers of hidden nodes, we will find the best values for number of hidden nodes and training parameters. Then with these in hand we can finally train the network using all data, with the best number of hidden nodes and training parameters. The algorithm is summarized below.

```
loop $N$ times:
    pick a number of hidden nodes
    pick training parameters (for example learning rate)
    divide train data into $k$ parts
    for i = 1 to $k$:
        train network using $k - 1$ parts
        compute accuracy using 1 part
    end for
    compute average accuracy of the $k$ runs
    if avg accuracy best found so far
        save number hidden nodes used
        save training parameters used
        save best average accuracy value
    end if
end loop
train network using all data
    (using best number hidden nodes,
    and best training parameters)
estimated accuracy is best accuracy found above
```

Training dataset and test dataset

The data are used in the project represent two dataset, acceptor dataset and donor dataset. Each original datasets consist of sequences of nucleotides (genes) and binary targets (”1” means that the sequence is either acceptor or donor, ”0” - something else). The example of the acceptor input file is presented below:

1
GGGCCCCTAGCGGAA . . . TTCAGGATGGGGAACCCCCTCAGCA

```
1
GAGGACAGGTGTCTC...CCCAGGTGATTGAACAGAGCTACAA
1
TGGGGGAAACAGGAA...GCCAGGAAAAGGGCCATGCTGGTCA
 ...
0
CTACTGGAGGCTCCCCGCT...GTTGCAGAAGATGAAGTGGTT
0
GAGGCTCCCCGCTAACAGG...GAAGATGAAGTGGTTTGTGAG
0
GCTCCCCGCTAACAGGTAG...GATGAAGTGGTTTGTGAGATT
```

Acceptor dataset includes 5578 genes, each of which consists of 90 nucleotides ('A', 'C', 'G', 'T'), donor dataset includes 5256 genes with length of 15 nucleotides. So there are two datsets and as a result there is built two models, two neuron networks, for each dataset. The goal is to build high quality models for recognition acceptor and donor splice sites.

After preprocessing the data (it is described in the next section) each dataset was splited into training ans test ones. The acceptor training and test datasets have 3905 (70% of total observations) and 1673 randomly chosen observations, respectively. The donor training and test sets have 3679 and 1577 observations, respectively. The experiments were repeated for different training and test sets.

Data preprocessing and input to the neural network

Before feed the model with data, they were preprocessed in a few steps.

At first, the missing values were removed. Here missing values are observations, genes, that contain gaps or letters different from nucleotides abbreviations (A - adenine, C - cytosine, G - guanine, T - thymine).

Further, after removing, each gene was transformed into an orthogonal binary vector (A = 1000, C = 0100, G = 0010, T = 0001). This input description has been used in several other studies [12, 14]. This input system is called orthogonal input, due to the orthogonal binary vectors. This is the most used input representation for neural networks in the application of gene prediction. Such scheme also has the advantage that each nucleotide input is separated from each other, such that no arithmetic correlation between the monomers needs to be constructed.

Measurement of performance

The above method is used to predict if the sequence is acceptor or donor splice site. The obtained result can be compared with the actual genes. There are four different outcomes of this comparison. These outcomes are summarized in the next table:

	Predicted positive	Predicted negative
Real positive	TP	FN
Real negative	FP	TN

Table 4.1. Confusion matrix

- True positive (TP): A sequence is correctly identified as either acceptor(donor) splice site
- False positive (FP): An acceptor (a donor) splice site is incorrectly identified as not such
- True negative (TN): A sequence is correctly identified as not acceptor (not donor)

- False negative (TN): A sequence is incorrectly identified as acceptor (donor) splice site

The counts of each comparison outcome are used to compute standard measurement indicators to benchmark the performance of the predictor. The *mean squared error (MSE), Accuracy, Error rate, Recall, Precision and Specificity* have been chosen for measuring the performance of prediction tools.

Accuracy. Accuracy is the ratio of correctly classified instances to whole instances:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

Error rate. Error rate is the ratio of incorrectly classified instances to whole instances:

$$Error = \frac{FP+FN}{TP+FP+TN+FN} = 1 - Accuracy$$

Recall. Recall is the ratio of correctly predicted acceptor (donor) splice sites to all actual nucleotides sequences. It defines how many relevant genes are selected:

$$Recall = \frac{TP}{TP+FN}$$

Precision. Precision is the ratio of correctly predicted acceptor (donor) splice sites to all positive predicted sequences. It defines how many selected genes are relevant:

$$Precision = \frac{TP}{TP+FP}$$

Specificity. Specificity is the ratio of negatives that are correctly identified as such:

$$Specificity = \frac{TN}{TN+FP}$$