Übung: Secondary Structure Prediction

Exercise: 28 Nov, 5 Dec 2011

Assignment due date: 11 Dec 2011

It is likely that this Übung will really require the allocated two weeks. Don't lose too much time on the first task during exercise time in the student pool, it is advisable to start on the second and third tasks once you are confident that you understand the first task. Please e-mail (**`ast_uebung [at] zbh.uni-hamburg.de`**) a brief written report (in .pdf or .txt plaintext format) answering the following questions together with your source code for the third task.

Each person should hand in their own report, no group submissions please.

Tasks:

1. In the lectures, we considered neural networks which use a "logistic" function for switching. There is also a strong school of thought based on other activation functions. Before considering neural networks applied to proteins and secondary structure, we consider a purely theoretical example, based on a simpler, linear classifier.

   Frank Rosenblatt's perceptron can be seen as a simple kind of feed-forward neural network. It is a linear classifier mapping its input vector $\boldsymbol{x}$ to a linear function $f(x)=\sum_i w_i x_i - b$, where $\boldsymbol{w}$ is a vector of weights w_i and b is a bias. This version of an artificial neuron can be used as a binary classifier using the sign of $f(\boldsymbol{x})$ as decision basis. The output is not $f(\boldsymbol{x})$, but 1 or -1.

   In this exercise you will use a slightly modified model that calculates the weighted sum of two inputs x_1, x_2 and compares it to a threshold b.

   If the weighted sum is greater than the threshold then the neuron fires +1 otherwise -1, i.e. either $y = +1$ or $y = -1$. This is essentially the same as if you used Rosenblatt's perceptron as a binary classifier. The illustrated neuron can be trained (or it can be made learned) through error correction from training examples. If $d(k)$ is the desired and $y(k)$ is the obtained output of training example k, then the error is given by $e(k) = d(k) - y(k)$. The synaptic weight w_i changes during the learning procedure according to the learning rule $w_i(k+1) = w_i(k) + \eta e(k)x_i(k)$, where η is the learning rate.

The following training set specifies the truth table for the logic **AND** operator:

$x_1(k)$	$x_2(k)$	$d(k)$	k
1	1	1	1
-1	1	-1	2
-1	-1	-1	3
1	-1	-1	4

Given the initial conditions $w_1 = -0.2$, $w_2 = +0.1$, $b = +0.2$ and $\eta = +0.1$, find the synaptic weights that solve the problem.

2. The aim is to try out secondary structure prediction programs on some interesting examples, where one knows the answer. The answers come from:
   * STRIDE or DSSP – use coordinates to estimate the secondary structure.
   *AUTHOR ASSIGNMENTS - whatever the authors think is correct after looking at their structures

   The question is, how close are the predictions (GOR-IV, HNN, JPRED, PHD) to the estimates based on the structure. The examples here use public web servers. These are all free, but do remember that somebody has built the server and provided the computer time. Be careful not to flood any of the servers with requests.

   There are two weeks allocated for this Übung. Note that some of the servers listed below do not send results back instantly. It might be safe to submit queries in the first week and look at their answers later (next week).

Have a look at the following three protein pairs:

Protein 1	Protein 2	identical sequence
`1ial` (292-300)	`1pky` (413-421)	KGVVPQLVK
`1cgu` (121-127)	`1bgl` (835-841)	LITTAHA
`1efv` (119-124)	`1p04` (114-119)	LLPRVA

You might remember few of the proteins from the lectures. There are three pairs of proteins with known structure. There is an identical chunk of sequence in each of three sequences which adopts a different secondary structure in each protein.

For all three pairs of proteins, retrieve their (single-lettered) sequences and tertiary structures (PDB-files) from the RCSB protein data bank (PDB) found at **`www.pdb.org or www.rcsb.org`**.

Now use GOR IV, information theory based method, (**http://npsa-pbil.ibcp.fr/** and follow the link to "GOR IV") and HNN (same address as GOR IV), based on a hierarchical neural network to predict the secondary structures. Compare the prediction results to the secondary structure assignment based on the 3D coordinates, for example made by the STRIDE services (**`webclu.bio.wzw.tum.de/stride/`**). Compare also the DSSP and STRIDE assignments against each other and against the "author approved" assignments (all found under the “Sequence Details” tab at **`www.rcsb.org`**). Note that the PDB website includes the DSSP and STRIDE assignments (in the Sequence tab, then “Add Annotations”).

In your report, write about:

- the differences and similarities in assignments of the complete sequences and especially in the ambiguous parts.
- Do the servers make the same prediction for each member of the protein pair? Why (not)? Describe your observations, compare and discuss them.
- The servers often give an estimate of confidence or reliability. How do they treat the regions where the same sequence sometimes adopts different secondary structure?

- If you are brave and a little patient (30 – 60 min.), you should include predictions from the JPRED **http://www.compbio.dundee.ac.uk/www-jpred/** or PHD **http://www.predictprotein.org/** services in your report. The PHD server requires an unfriendly, but harmless registration. Both servers take multiple sequence alignments into account and reflect the state of art. Why might these methods make not much sense here?

3. Investigate the following hypothesis:

   "*The number of secondary structure elements found in random sequences is significantly lower and/or their length is significantly shorter than in protein primary structures (with the same amino acid composition).*"

   If this hypothesis is true, secondary structure prediction could be used as a starting point to find structural genes coding for proteins.

   The first part of this task is to write a small program to generate random protein sequences, and then present a way the hypothesis could be tested quantitatively using this program. You should perform your proposed test on some examples and present your findings for these example tests.

   Please hand in your source code as a separate file together with instructions for compiling it if necessary (for example a Makefile) together with your answers. If you are unable to write this program there is a section at the end describing the use of a pre-made program for generating random sequences.

   Writing your program for generating random sequences

   You should try and write your own program to generate random sequences. You may use any language you like, but the program should be runnable in the student pool under Linux.

   The plan is to shuffle a given protein sequence by swapping amino acids, generating a permutation of the original sequence. A shuffled sequence naturally has the same amino acid composition as the original sequence. One of the oldest and simplest algorithms for shuffling a sequence is the Fisher-Yates shuffle (sometimes also called the Knuth shuffle), which is the algorithm you should use here.

The Fisher-Yates shuffle in pseudocode for an array `a[]` of `n` elements with indices from `0` to `n-1`:

```
for i from n − 1 downto 1 ( 1 ≤ i ≤ n-1 )
    j = random integer in range [0, i] ( 0 ≤ j ≤ i )
    swap a[i], a[j]
```

Your program should take the number of shuffled sequences to output and the original sequence from the command-line. Example output should be (assuming you called your program shuffle):

```
$ ./shuffle 4 abc
bac
abc
cba
acb
```

You should test that your program generates all permutations with equal probability. A simple test is the following:

```
./shuffle 100000 abc | sort | uniq -c
```

All six possible permutations of the string “abc” should occur the same number of times on average.

Also make sure that you seed your random number generator (the function `srand()` in C) so that calling your program multiple times won't generate the same set of permuted sequences.

If you use a language like C, make sure that your program runs under `valgrind` (a program that can find certain kinds of errors while running your program) without errors. You can run your program under `valgrind` like so:

```
valgrind ./shuffle 4 abc
```

Using an existing program for generating random sequences

Only read this part if you are unable to write your own program.

Under ‘`/home/torda/uebung_sec_struct/`’ directory, you can find a little command-line tool that

takes a protein sequence and produces (pseudo) random sequences. These pseudo-random sequences are of the same composition and length as the input sequence, but the order of the amino acids has changed.

In the text file `/home/torda/uebung_sec_struct/data/1tu7_random`, you can find random sequences derived from the primary structure of the major cytosolic Glutathione-S-transferase from the parasitic nematode *Onchocerca volvulus*, the causing organism of river blindness.
But, of course, you are free to use the command-line tool on your own to produce your own random sequences. To do so you can launch the tool by typing, for example:

```
/home/torda/uebung_sec_struct/bin/shuffle_seq.x 3 MSYKLTYFSIRGLAEP
```

at command prompt (e.g. *konsole*). This would print the original sequence and three random sequences to stdout. To save this output to a file, use the redirection operator `> random_seq.out` after your typed command to store the output in the file `random_seq.out`. You can find the source code in the *src* subdirectory.[1]

[1] Code originally written by Gundolf Schenk