WS 2006/07 Übung zu Anwendungen der Strukturanalyse

Originally written by Gundolf Schenk

Übung III: Secondary Structure Prediction

29 Nov 2007

It is likely that this Übung will really require the two weeks allocated. Read the entire document and maybe decide to have only a quick look at task 1 and then perform (partly) the other tasks at first. Please hand in (or e-mail `torda@zbh.uni-hamburg.de`) a brief written report answering the following questions not later then 20. December 2007.

Tasks:

1. In the lectures, we concentrated on neural networks which used 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 the simplest 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.
   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 bigger 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 learn) through error correction from training examples. If $d(k)$ is the desired and $y(k)$ is the obtained output of training example k 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, usually where one knows the answer.
   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 back results instantly. It may be best to submit a query in the first week and look at it later (or the next week).
   Have a look at the following 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

   Some of them you might remember from the lectures. These are three pairs of proteins with known structure. Within each pair, there is an identical stretch of primary structure, which adopts a different secondary structure in each protein.
   Retrieve their (single-lettered) sequence and their tertiary structures (PDB-files) from the RCSB protein data bank (**`www.rcsb.org`**).
   Now, use GOR IV (based on information theory,
   `npsa-pbil.ibcp.fr` and follow the link to "GOR IV") and NNPREDICT (based on a feed-forward neural network, (**`alexander.compbio.ucsf.edu/~nomi/nnpredict.html`**) 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`**).
   Report on the differences and similarities in assignments of the complete sequences and especially

write about 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 (30-60min.), you should include predictions from the JPRED (**`www.compbio.dundee.ac.uk/~www-jpred`**) or PHD (**`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.*"

   If this hypothesis is true the secondary structure prediction could be used as a starting point to find structural genes

   In `/home/torda/uebung_sec_struct/` 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 textfile `/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
   ```

   in a 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]

   Develop and perform (partly) an experiment that would allow to verify the hypothesis. How did you investigate the hypothesis? What is your result ?

[1] Code originally written by Gundolf Schenk