Neural network tutorial instructions

June 14, 2017

The archive file for the tutorial contains the following files:

CG_Network_Streamlined.cpp

L2_error.cpp

INPUT.txt

ala-dip-data_all.txt

Makefile

The file "CG_Network_Streamlined.cpp" contains the source code for optimizing a neural network using the conjugate gradient algorithm discussed in the lecture. The file "L2_error.cpp" contains a short program for computing the L^2 error between a benchmark free energy surface and that generated by the neural network from the training data. The file "ala-dip-data_all.txt" contains all of the data generated by a TAMD-AFED calculation of the Ramachandran surface for the alanine dipeptide in aqueous solution. The file "INPUT.txt" contains the input parameters for the calculation, and the file "Makefile" is a makefile for compiling the source code.

In order to carry out the tutorial, you will need to follow these steps:

1. Unpack the tar file:

   ```
   tar -xzvf Neural.tar.gz
   ```

2. Change to the directory just created and compile the source code. First, edit 'Makefile' and make sure the C and C++ compilers correspond to the ones you have available on your system, e.g., 'gcc' and 'g++' or 'icc'. The compile the code by typing

   ```
   make
   ```

3. Create a training data set from the full data set. To do this, you can use one of two commands:

   ```
   head -n ala-dip-data_all.txt > ala-dip-data.txt
   ```

   or

```
tail -nl ala-dip-data_all.txt > ala-dip-data.txt
```

Here n is the number of training points you wish to extract from the full data set.

3. Set the calculation type, number of conjugate gradient steps, checkpointing frequency for weights, and number of conjugate gradient line-minimization steps. These are the 2nd, 3rd, 4th, and 5th lines in the file "INPUT.txt". Edit the file "INPUT.txt". On the second line, three options are possible: "1" indicates a calculation of neural network parameters starting from scratch. "-1" indicates a calculation of neural network parameters starting from an old set contained in a file called "weight.txt". "0" indicates that you wish to perform a validation calculation of the neural network. Start by setting this parameter to "1". On the next line, the number of CG steps is set. A good value for this number is between 100 and 200. The next lines contains the frequency with which the weights are written, and a frequency of 20 is likely fine. Finally, the number of CG substeps appears on the last line. Here, a value between 50 and 100 should suffice. Note that the more CG steps and substeps you set, the longer the calculation will take.

4. set the number of hidden layers and the number of nodes within each hidden layer. To do this, you need to edit the source code "CG_Network_Streamlined.cpp". On line 16 of this file is a `#define` command for the number of layers. Set the integer value on this line to the desired number of hidden layers, e.g. 2. Three lines below this is a variable declaration "`int const amount`" In the definition of this variable is a bracket "{`input` $n1, n2, n3, ...$`output`}". The parameters $n1, n2, n3, ...$ are the number of nodes within layer 1, 2, 3,... Make sure the number of integers set here corresponds to the number of hidden layers, e.g., if you want 2 hidden layers, then set 2 integers only between 'input' and 'output'. Note that these numbers need not be the same.

   If you have changed any of lines in the source code, you will need to recompile by typing

   ```
   make
   ```

5. The compilation will produce an executable called "Machine_Learning.exe". Next, run the calculation by typing

   ```
   ./Machine_Learning.exe
   ```

   The screen output will indicate the progress of your calculation in both CG macrosteps and CG substeps, reporting the value of the error or cost function $E(\mathbf{w})$.

6. When the calculation has finished, run a second round of minimization. This is done by editing the file "INPUT.txt", changing the second line from "1" to "-1" and inserting at the end of the file "INPUT.txt" the file "weight.txt" generated from the run in step 5. Save the file. When this is done, type

   ```
   ./Machine_Learning.exe
   ```

   which will run the second round of minimization. Check the value of the cost function, which should have dropped below 0.001. If not, you can run another round of minimization following this step again.

7. Now you are ready to validate the final neural network you have generated. To do this, you need to generate a validation set. Use the commands in step 3 to create this set in the following manner: If, in performing step 3, you used the "head" command, use "tail" for creating the validation set, and vice versa. In this case, n will represent the number of validation points you wish to use, and this value of n need not be the same as used in step 3. Once you've created the validation set, edit the file "INPUT.txt" and change the second line to "0". Delete all of the lines below the 5th line, and insert the contents of the newly generated "weight.txt" file in place of these deleted lines. Save the file. When this is done, type

   ```
   ./Machine_Learning.exe
   ```

   This will generate a file called "output.txt". which contains the benchmark free energy surface together with that generated by your neural network. In order to compute the L^2 error, compile the utility "L2_error.cpp" via

   ```
   g++ -o L2_error.x L2_error.cpp -lm
   ```

   (if "g++" is not your C++ compiler, then substitute the name of your compiler in for whichever one you are using on your system). Run the utility

   ```
   ./L2_error.x
   ```

   You will be asked to input the number of validation points you are using, and it will compute and write to the screen the value of the L^2 error. Record this value together with the parameters of the network you used.

Now that you have completed one full run, try changing the parameters and see how the L^2 error changes. Things you can try to change include

The number of training data points.

The number of hidden layers.

The number of nodes in each layer.

The number of CG macrosteps.

8. If time permits, you can also try changing the activation function. The definitions of these $h(x)$ and its derivative, $h'(x)$, are coded between lines 33 - 50 in the source code `CG_Network_Streamlined.cpp`. Some different activation functions you can try are

$$\begin{aligned} h(x) &= \frac{1}{1+a^2x^2} \\ h(x) &= \tanh(ax) \\ h(x) &= \ln\left(1+e^{ax}\right) \\ h(x) &= \sqrt{\frac{\pi}{a}}e^{-a^2x^2} \end{aligned}$$

   Here, a is a parameter you can choose and vary at will. Are the results you obtain sensitive to the choice of the activation function?