Metodi Computazionali della Fisica
Secondo Modulo: C++

Quarta Lezione

Note sulle parametrizzazioni

Implementazione di una rete neurale

Introduzione alla minimizzazione

Definizione della classe Minimization

In generale

La parametrizzazione di una funzione $f(x)$ può essere scritta come

$$f(x;\mathbf{c}) = c_0 f_0(x) + c_1 f_1(x) + \ldots + c_m f_m(x)\,,$$

dove c_i sono alcuni dei parametri

- $f_i = (x - x_0)^i$ nel caso di un polinomio di Taylor;
- $f_i = \frac{1}{1+e^{-\omega_i x+\theta_i}}$ nel caso di una rete neurale;
- $f_i = e^{-\frac{(x-x_i)^2}{2\sigma_i^2}}$ nel caso di una funzione radiale.

Dalla fine della scorsa lezione:

```
class Mlp : public Parametrization{
public:

  Mlp();
  Mlp(const Data&);
  Mlp(const Mlp&);
  ~Mlp();

  vector<double> Output(const vector<double>&);

private:

  void Read();
  double Activation_function(const size_t,double&);

  double xi[MXNODES][MXLAYERS];
  size_t n[MXLAYERS];
  size_t tnl,ilast_layer,iaf_type;
};
```

$$\xi_i^{(l)} = g\left(\sum_{j=1}^{n_{l-1}+1} \omega_{ij}^{(l-1)}\xi_j^{(l-1)}\right) .$$

Note

Per poter *adattare* una parametrizzazione ad un set di dati abbiamo bisogno di:

- una funzione di errore [figure di merito];
- un algoritmo di minimizzazione.

Il χ^2

Esistono diverse possibili funzione di errore. Noi utilizzeremo questa:

$$\chi^2 = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{x_{\text{data}} - x_{\text{fit}}}{\sigma} \right)^2 ,$$

dove σ può essere sia l'errore sperimentale delle misure, sia l'accuratezza numerica che vogliamo.
In statistica minimizzare il χ^2 corrisponde a masimizzare la *likelihood*, ovvero la probabilità condizionata che un parametro (o un insieme di parametri) assuma un certo valore a partire da una misura (un insieme di misure).

Un algoritmo genetico

In breve:

1. si fanno n_{ind} cloni del set di parametri iniziali;
2. da ogni clone si generano n_{mut} mutanti;
3. si valuta la funzione errore per ognuno dei mutanti;
4. si selezionano gli n_{ind} mutanti che hanno dato i migliori valori della funzione errore;
5. si ripete il tutto dal punto 2 fino a quando non viene soddisfatto un criterio di stopping.

minimization.hpp [#1]

```
#ifndef _MINIMIZATION_HPP
#define _MINIMIZATION_HPP

#include <iostream>
#include <vector>
#include <fstream>
#include <limits>
#include <cmath>
#include "parametrization.hpp"

enum {GA,SD};

using namespace std;
```

minimization.hpp [#2]

```
class Minimization {
public:

  Minimization(const Data&,Parametrization*&);
  ~Minimization();

  size_t initialize(vector<double>);

  double dump_error_function(const Data &,Parametrization*&);
  void dump_data_vs_fit(const Data &,Parametrization*&);
  void minimize(const Data&,Parametrization*&);
```

minimization.hpp [#3]

```
private:

  Minimization();
  Minimization(const Minimization&);

  void read();
  double chi2d(vector<double>&,vector<double>&,vector<double>&);

  void genetic_algorithm(const Data&,Parametrization*&);
  void mutation();
  void sort(vector<double>&);
  vector<double> mutate(vector<double>&);
  vector<double> assign_best();

   vector<double> normalize(vector<double>, const vector<double>,
   const vector<double>);

   vector<double> unormalize(vector<double>, const vector<double>,
   const vector<double>);
```

minimization.hpp [#4]

```
  vector<double>* mutants;
  vector<double>* individual;

  size_t iminim_type;
  size_t iterations, ndump, nsmear;
  double chi2tr_max, t_frac;

  size_t indsize, mutsize, n_mut_mul, nclones;
  double mut_rate;

  vector<double> chi2t,chi2v;

};

#endif
```