Metodi Computazionali della Fisica
Secondo Modulo: C++

Terza Lezione

Soluzione esercizio seconda lezione

Una bozza della classe Parametrization

Il comando `make`

Definizione della classe Parametrization

Introduzione alla reti neurali

Definizione

- Una parametrizzazione è una mappa tra input ed output.
- Un parametrizzazione è definita da un insieme di parametri e dal modo in cui questi agiscono per produrre l'output per ogni dato input.
- Esempi di parametrizzazione sono un polinomio di grado n (o sviluppo di Taylor di ordine n), uno sviluppo di Fourier di un certo ordine, una rete neurale.

Definizione

- Una parametrizzazione è una mappa tra input ed output.
- Un parametrizzazione è definita da un insieme di parametri e dal modo in cui questi agiscono per produrre l'output per ogni dato input.
- Esempi di parametrizzazione sono un polinomio di grado n (o sviluppo di Taylor di ordine n), uno sviluppo di Fourier di un certo ordine, una rete neurale.

Definizione

- Una parametrizzazione è una mappa tra input ed output.
- Un parametrizzazione è definita da un insieme di parametri e dal modo in cui questi agiscono per produrre l'output per ogni dato input.
- Esempi di parametrizzazione sono un polinomio di grado n (o sviluppo di Taylor di ordine n), uno sviluppo di Fourier di un certo ordine, una rete neurale.

In pratica

Vogliamo definire una classe `Parametrization` con carateristiche generali:

- variabili: `ninput`, `noutput` e `parameters`;
- metodi: `initialize`, `get`, `set`, `dump`,etc.

Vogliamo definire un insieme di classi derivate che implementino alcuni aspetti particolari:

- variabili peculiari della parametrizzazione;
- metodi: `read`, `output`.

Vogliamo poter definire un oggetto di tipo `Parametrization` e fare in modo che questo di volta in volta corrisponda alla particolare scelta effetuata.

Una bozza [#1]

```
#include <iostream>
#include <vector>
using namespace std;

class Parametrization {
public:

  Parametrization();
  Parametrization(size_t,size_t);
  Parametrization(const Parametrization&);
  virtual ~Parametrization();

  virtual void output() = 0;
  void init_parameters();
  void set_parameter(size_t i, double x){parameters[i]=x;};
  vector<double> get_parameters(){return parameters;};
  size_t get_ninput(){return ninput;};
  size_t get_noutput(){return noutput;};
  size_t get_npar(){return npar;};
```

Una bozza [#2]

```
protected:

  virtual void read() = 0 ;
  vector<double> parameters;
  size_t npar;

private:

  size_t ninput, noutput;

};
```

Una bozza [#3]

```
class Pol : public Parametrization{
public:

  Pol();
  Pol(size_t,size_t);
  Pol(const Pol&);
  ~Pol();

  void output();

private:

  void read(){npar=4;cout<<"Pol read"<<endl;};

};
```

Thinking in C++, cap. 16 'Design patterns'

>man make

```
NAME
       make - GNU make utility to maintain groups of programs

[...]

DESCRIPTION
       The  purpose  of  the make utility is to determine
       automatically which pieces of a large program need
       to be recompiled, and issue the commands to recompile
       them.  [...]

       To  prepare  to  use make, you must write a file called
       the makefile that describes the relationships among files
       in your program, and the states the commands for updating
       each file. [...]
```

Makefile: un possibile esempio [# 1]

```
all: rmxeq  mkxeq
.PHONY: all

# main programs and modules to be compiled

MAIN = main
MOD = data ran parametrization

MODULES = $(MOD)
MDIR = modules/
VPATH = .:$(MDIR)

CXX = g++
OFLAGS = -O3 -Wall

PGMS = $(MAIN) $(MODULES)
```

Makefile: un possibile esempio [# 2]

```
$(addsuffix .o,$(PGMS)): %.o: %.cpp Makefile
$(CXX) $< -c $(OFLAGS)
$(MAIN): %: %.o $(addsuffix .o,$(MODULES)) Makefile
$(CXX) $< $(addsuffix .o,$(MODULES)) $(OFLAGS) -o $@

# produce executables
mkxeq: $(MAIN)

# remove old executables
rmxeq:
@ -rm -f $(MAIN); \
        echo "delete old executables"

# clean directory
clean:
@ -rm -rf *.o *~  $(MAIN)
.PHONY: clean
```

parametrization.hpp [#1]

```
#ifndef _PARAMETRIZATION_HPP
#define _PARAMETRIZATION_HPP

#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include "../include/data.hpp"
#define MXNODES 50
#define MXLAYERS 5

enum {INIT,FINAL};
enum {LINEAR,FERMI,SINUS};
enum {GAUSS,SPLINE};

using namespace std;
```

parametrization.hpp [#2]

```
class Parametrization {
public:

  Parametrization();
  Parametrization(const Data&);
  Parametrization(const Parametrization&);
  virtual ~Parametrization();

  virtual vector<double> output(vector<double>) = 0;
  void init_parameters();
  void set_parameters(vector<double> x){parameters=x;};
  void set_parameter(size_t i, double x){parameters[i]=x;};
  vector<double> get_parameters(){return parameters;};
  int dump_parameters(size_t);
  size_t get_ninput(){return ninput;};
  size_t get_noutput(){return noutput;};
  size_t get_npar(){return npar;};
```

parametrization.hpp [#3]

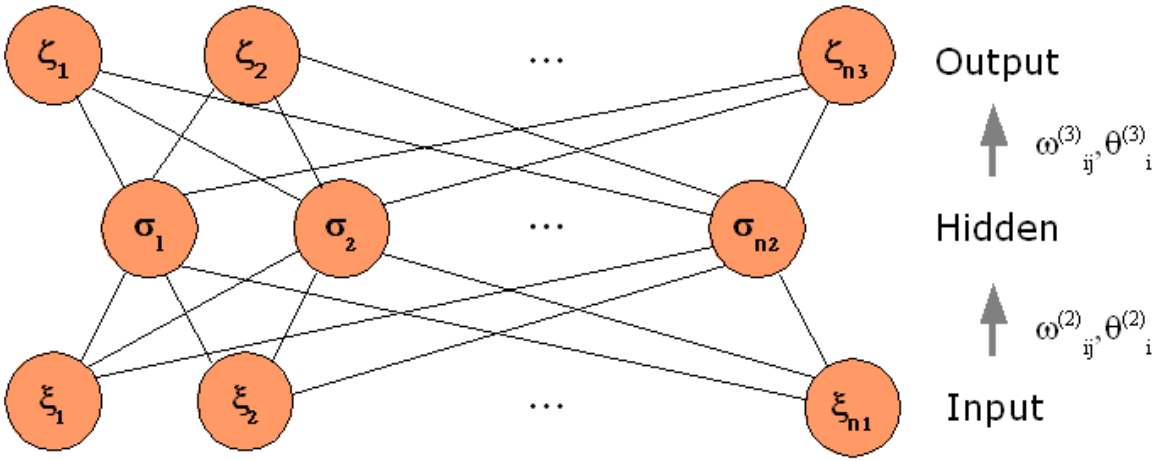
```
protected:
  virtual void read() = 0 ;
  vector<double> parameters;
  size_t npar;

private:

  size_t ninput, noutput;

};
```

parametrization.hpp [#4]

```
class Pol : public Parametrization{
public:

  Pol();
  Pol(const Data&);
  Pol(const Pol&);
  ~Pol();

  vector<double> output(vector<double>);

private:

  void read();
  size_t n[MXLAYERS];
};
```

Definizione

Una rete neurale è una struttura di nodi interconnessi.



Nate per simulare il funzionamento del cervello umano, le reti neurali si sono rivelate utili algoritmi per la ricostruzione di informazione da dati rumorosi o incompleti.

MultiLayer Perceptron

- Funzione di attivazione:

$$\xi_i^{(l)} = g\left(\sum_{j=1}^{n_{l-1}} \omega_{ij}^{(l-1)} \xi_j^{(l-1)} - \theta_i^{(l)}\right), \qquad g(x) = \frac{1}{1+e^{-x}}$$

- In un caso semplice abbiamo:

$$\xi_1^{(3)} = \frac{1}{1+e^{\theta_1^{(3)} - \frac{\omega_{11}^{(2)}}{1+e^{\theta_1^{(2)} - \xi_1^{(1)}\omega_{11}^{(1)}}} - \frac{\omega_{12}^{(2)}}{1+e^{\theta_2^{(2)} - \xi_1^{(1)}\omega_{21}^{(1)}}}}}$$

- In pratica:

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

parametrization.hpp [#5]

```
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;
};
```