Last updated on Wednesday, August 15, 2018

2018 IFN680 - Assignment One

Differential Evolution for Hyperparameter Search

Assessment Information

- Code and report submission due on **Sunday 16th September, 23:59pm**
- Use **Blackboard** to submit your work (report and code)
- Max **group size**: three people per submission. Smaller group sizes allowed (1 or 2 people).

Quick Overview

- You will implement a simple generic Differential Evolution search
- You will apply this generic algorithm to two problems
- You are provided with scaffolding code that you will need to complete
- You will also perform experiments and report on your results

Introduction

In machine learning, a ***hyperparameter*** is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters like the weights of a neural network are derived via training. Examples of hyperparameters for neural networks include the number of layers, the number of neurons in each layer and the learning rate.

Different training algorithms require different hyperparameters. Some simple algorithms (such as ordinary least squares regression) require none. The optimal values for the hyperparameters depend also on the dataset. In this assignment, you will implement a search algorithm to find good values for a small set of hyperparameters of a neural network.

Differential Evolution (DE) is an efficient and high performing optimizer for real-valued functions. DE is the search technique that you are required to use in this assignment to solve the optimization problems. As DE is based on evolutionary computing, it performs well on multi-modal, discontinuous optimization landscapes. Another appealing feature of DE is that it is still applicable even if the function we try to optimize is not differentiable. This is often the case when some simulation is involved in the computation of the real-valued function being optimized. In these scenarios, a gradient descent approach is not applicable. DE performs a parallel search over a population of fixed size, where each population member is a higher dimensional vector.

In this assignment, the core problem is to adapt DE to perform a search for four hyperparameters of a neural network. In the next section, we describe in more details DE.

Differential Evolution

Let $f(w)$ be the cost function which must be minimized. Note that maximization can be performed by considering the function $-f$. The function $f(w)$ takes a candidate solution as argument in the form of a vector w of real numbers and produces a real number as output which indicates the cost of the

given candidate solution *w*.

There exist many variations of the DE algorithm. The variation that you have to implement has been chosen for its simplicity. In the rest of this section, variable names written in *blue italic* correspond to Python program variables used in the provided scaffolding code.

We constrain the search by specifying bounds for each component of *w* with a list of pairs called *bounds* in the scaffolding code. That is, we impose

bounds[k][0] <= *w[k]* <= *bounds[k][1]* for k in range(len(*w*))

For example, if *bounds* is [(1,100),(1,100),(-6,2),(-6,1)] , then w[1] is constrained to the interval [1, 100] and w[2] is constrained to the interval [-6, 2].

To make the code generic, we normalize each component of *w* to the interval [0, 1] by mapping with an affine function the interval [*bounds[k][0]*, *bounds[k][1]*] to the interval [0, 1].

We first initialize all individuals *w* of the initial population with random points from the search-space.

Until a termination criterion is met (maximum number of iterations/generations done, or acceptable cost is reached), we repeat the following loop:

For each individual *w* in the population do:

- Pick three distinct individuals *a*, *b* and *c* from the current population at random. The individuals *a*, *b* and *c* must be distinct from *w* as well.
- Create a *mutant* vector *a* + *mut* * *(b – c)* where *mut* is a constant (passed as an argument to the *differential_evolution* function)
- Clip the *mutant* entries to the interval [0, 1]
- Create a *trial* vector by assigning to *trial[k]* with probability *crossp* the value *mutant[k]* and probability *1-crossp* the value of *w[k]*
- Evaluate the cost of the *trial* vector. If the *trial* vector is better than *w*, then replace *w* by the *trial* vector

In other words, for each individual *w* in the current population, we create a challenger *trial* vector. If this challenger is better than *w*, then *w* is replaced by the *trial* vector .

The mutation factor *mut* with and the crossover constant *crossp* are numerically defined in the scaffolding code.

Scaffolding Code

The file "*my_submission.py*" contains scaffolding code. The key function is *differential_evolution*. This function return a generator. You should have seen generators in the Python tutorial https://docs.python.org/3/tutorial/classes.html#generators .

To validate your implementation of the function *differential_evolution*, you should first test and debug it with the function *task_1* as it tries to solve a simpler problem than *task_2*.

In **Task 1**, you fit a polynomial to a noisy dataset. This is the same problem as you have seen in the early pracs where you solve this problem with the pseudo-inverse and a gradient descent approach.

Your output should look similar to the figures below:

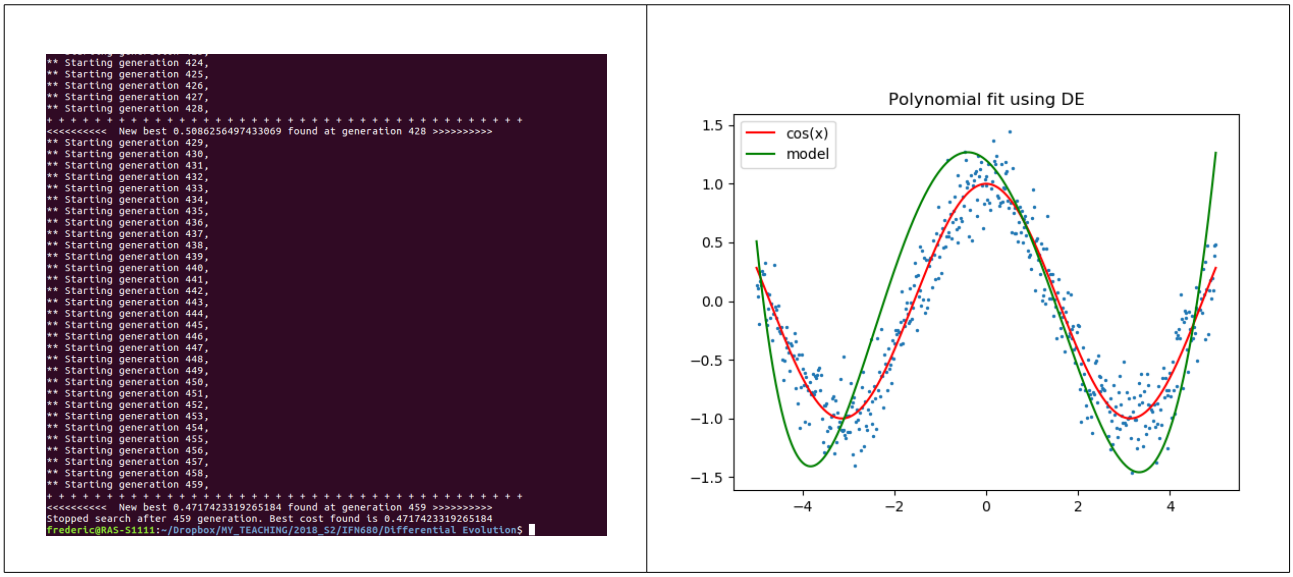

Table 1: Example of expected output for Task 1.

Task 2 is less of a toy problem. In this second task, you perform a search for some hyperparameters of a neural network using DE. This is a realistic scenario. There are no theoretical results to tell us what should be the architecture of a neural network for a given dataset. In practice, you can start by looking at what has worked on similar problems. But to explore more exhaustively the space of the potential neural architectures, it is better to perform a search using DE. The limiting factor of this approach is the training time of the neural networks.

The search space for Task 2 is a 4-dimensional vector space. The entries of a vector *w* of this space are:

- the number *nh1* of neurons in the first hidden layer of the neural network
- the number *nh2* of neurons in the second hidden layer of the neural network
- log10 (*alpha*) where *alpha* is the coefficient of the L2 regularization
- log10(*learning_rate_init*) where *learning_rate_init* is the initial learning rate

These hyperparameters are passed to the neural network constructor *MLPClassifier.*

You Job

- Complete the missing code for in the file "*my_submission.py*". You are not allowed to change the interface of the provided functions, but you can add extra functions and comments anywhere.
- In the scaffolding code for Task 2, we use a population size of 10 with 20 generations. That is we gave ourselves a computational budget of 200 = 10*20 neural training runs for the search. It is not obvious what is the best strategy. Should we use a population size of 5 with 40 generations, or maybe a population size of 40 with 5 generations?! To answer this question, write code (a new function *task_3*) to run experiments to compare the following (*population_size*, *max_iter*) allocations in the list [(5,40), (10,20),(20,10),(40,5)]. Write a section in your report to document your findings.

Submission

You should submit via Blackboard

- A **report in pdf format** strictly **limited to 4 pages in total** in which you present your experimental results for the three tasks using tables and figures. Only one person per group needs has to submit the assignment. Make sure you list the members of your group in the report and in the code.
- Your Python file **my_submission.py**

Marking Criteria

- **Report**: 4 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
  - Evidence based recommendations (Task 3).

Levels of Achievement

4 Marks	3 Marks	2 Marks	1 Mark	0 Mark
Report written at the highest professional standard with respect to spelling, grammar, formatting, structure, language, and terminology.	Report is very-well written and understandable throughout, with only a few insignificant presentation errors. Methodology, experiments and recommendations are clear.	The report is generally well-written and understandable but with a few small presentation errors that make one of two points unclear. Clear figures and tables.	Large parts of the report are poorly-written, making many parts difficult to understand. Use of sections with proper section titles.	The entire report is poorly-written and/or incomplete and/or impossible to understand. The report is in pdf format.

To get "i Marks", the report needs to satisfy all the positive items and none of the negative items of the columns "j Marks" for all j<i.

- **Code quality**: 4 marks
  - Readability, meaningful variable names.
  - Proper use of Python constructs like numpy arrays, dictionaries and list comprehension when applicable.
  - Header comments in all classes and functions.
  - Function parameter documentation.
  - In-line comments.

Levels of Achievement

4 Marks	3 Marks	2 Marks	1 Mark	0 Mark
Code is generic. Minimal changes would be needed to run same experiments on a different dataset.	Proper use of numpy array operations. Avoid unnecessary loops. Useful in-line comments. Code structured so that it is straightforward to repeat the experiments	No magic numbers (that is, all numerical constants have been assigned to variables). Appropriate use of auxiliary functions. Each function parameter documented (including type and shape of its parameters)	Header comments with instructions on how to run the code to repeat the experiments.	Code looks like a high-entropy spaghetti plate

To get "i Marks", the report needs to satisfy all the positive items and none of the negative items of the columns "j Marks" for all j<i.

- **Code functionality**: 12 marks in total
  - differential_evolution(): 6 marks (if fully working)
  - task_1: 2 marks (if fully working)
  - task_2: 2 marks (if fully working)
  - task_3: 2 marks (if fully working)

Final Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the practical sessions.
- Email questions to f.maire@qut.edu.au