A short write-up on the implementation of Particle Swarm optimization.

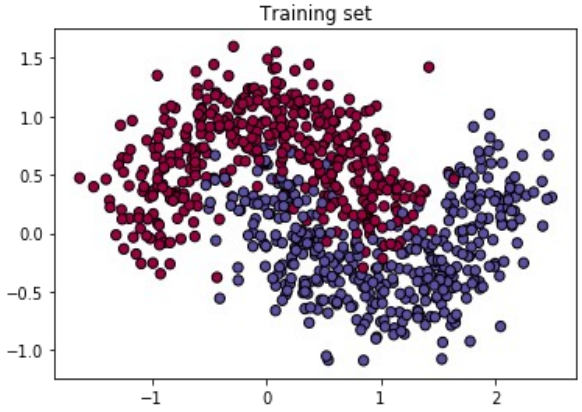
Algorithm:

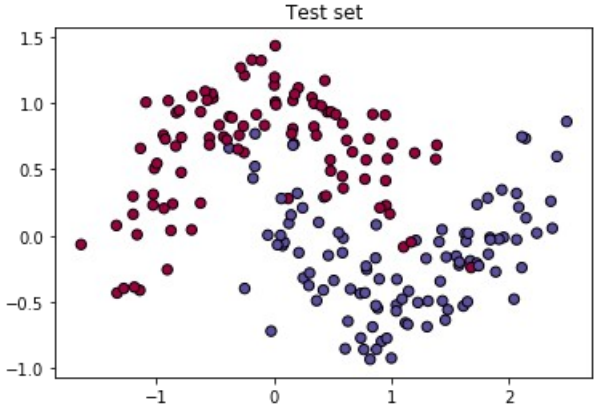
Particle Swarm Optimization with momentum

Application:

I have implemented the said algorithm for optimization of neural network parameters to minimize its loss and maximize classification accuracy. For data-set I've made use of already present library function 'make_moons' from sklearn library. The data-set was chosen keeping in consideration the complexity and visualization of problem.

Hence, I finally have a set of 2D data points and I build the model to classify whether the data point belongs to red or blue category. Below is the pictorial representation of the data sets (both training as well as test set) (80:20 split)





I am using a 5-layer architecture build ordinarily by using numpy. The model has the following architecture:

Input Layer/Inputs	2 Neurons
Hidden_Layer_1	3 Neurons
Hidden_Layer_2	4 Neurons
Hidden_Layer_3	3 Neurons
Hidden_Layer_4	2 Neurons
Output_Layer/Output	1 Neuron

So the overall architecture consist of 51 parameters (Weights + Biases), initially initialized randomly and are then optimized using PSO to minimize loss of NN and thus maximizing its accuracy.

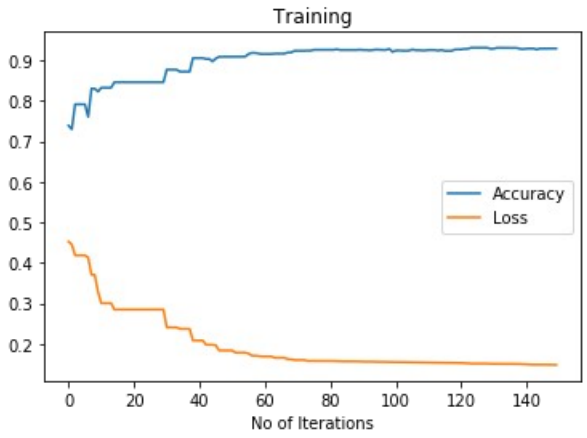
Assumptions:

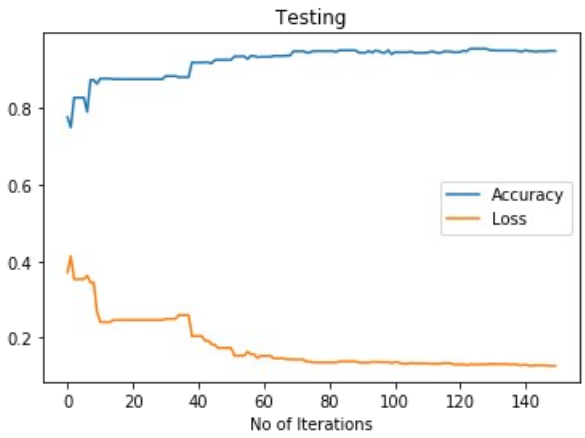
1. Dataset is created as 2D only from already Library
2. The parameters are initialized randomly
2. I am using Binary Cross Entropy as loss function and thus is also used for fitness measure in the PSO

Results:

I calculate accuracies while training at the end of each iteration. This helped me to see if the model is converging. I ran the model for total of 150 iterations.

The following graphs depict more clearly our models behavior.





So, the final training results were satisfying and hence achieved an overall test-time accuracy of 95.3% which shows the model performed really well on unseen data.

Code Execution:

1. Install Python 3.x
2. Install dependencies:
    1. numpy: for vector operations and complex function calculations
    2. matplotlib: for visualization
    3. jupyter notebook for seamless visualization
    4 sklearn : for dataset creation

4. Cd to the directory where attached '.ipnb' is located and start the jupyter notebook.
5. Follow the onscreen instructions on the notebook.
6. I have tried to keep the implementation simple and steady. I have also tried to keep comments wherein a user can understand and tickle with the parameters.

Notes:

The initialization of parameters in random and so the user may not get the desired accuracy in the first run. It is recommended to re-run the implementation several times to get better results.
In my case, I ran the model 2 times to get to a marginal accuracy. I will leave the said parameters in references.

Moreover, the model is flexible to tune for variety of application and can be extended to a large set of classification tasks with few changes in the hyperparameters.

I have tried to keep the model simple, procedural and without much abstraction in it. The sole purpose of which is to make any user mold or change it according to its own problem statement.

References:

1. Medium - DecisionTree Classifier — Working on Moons Dataset using GridSearchCV to find best Hyperparameters (For Dataset Creation)

2. Python Numpy Tutorial (https://cs231n.github.io/python-numpy-tutorial/)

2. # 51 parameters loss 0.148 iter 150

```
gbest = np.array([  4.97068145,  -2.68814875,  -2.28928195,   0.86862162, 2.85398692,   1.33864996,   3.79240839,  -7.51317275,
                   -3.86429077,  -4.31387192,  -4.11028348,   1.5646129 , 0.4029814 ,  -0.92741715,   0.85748413,   2.84747029,
                    0.04160663,   0.56312397,   1.06214566,   1.53984206, 0.724875  ,  -2.22347727,  -8.0293366 ,   1.77186317,
                   -1.92731939,  -2.20343603,   4.68690343,   0.7726582 , 2.92830511,  -1.47057823,   1.15980866,  -4.98607546,
                    3.33487488,   1.58834295, -15.32838022,   8.4776463 , 10.16347111,   2.89998419,   4.24816663,   3.83446135,
                   -2.20423708,   4.3371909 ,  -3.46285   ,  -2.22208246, 4.01846773,   2.37110802,   6.08027008,   7.35848904,
                   -1.0818903 ,   3.91443871,  -7.74464867])
```