Lecture 13: Heuristic Optimization

Goals for Today:

- Heuristic optimization overview
- Particle Swarm Optimization
- Genetic Algorithms

Why are we talking about this?

All of the methods we have covered for optimization are based on evaluating the gradient of a loss function (**L**) with respect to model parameters (**β**).

Gradient Descent :

$$\beta_{i+1} = \beta_i - \alpha \nabla_\beta \mathbf{L}$$

However, there are many situations where the gradient is unavailable to us either because (i) our features are non-differentiable, or (ii) the analytical form of our loss function is unknown.

For situation (ii), we can always numerically evaluate our gradient (e.g., by taking small steps in β_i) but this is usually much more expensive and we won't go into further detail here.

Heuristic* algorithms have been developed to address various situations where gradient-based optimization fails. This includes, (i) gradient-free methods of various flavors, (ii) methods designed to escape shallow minima, (iii) methods

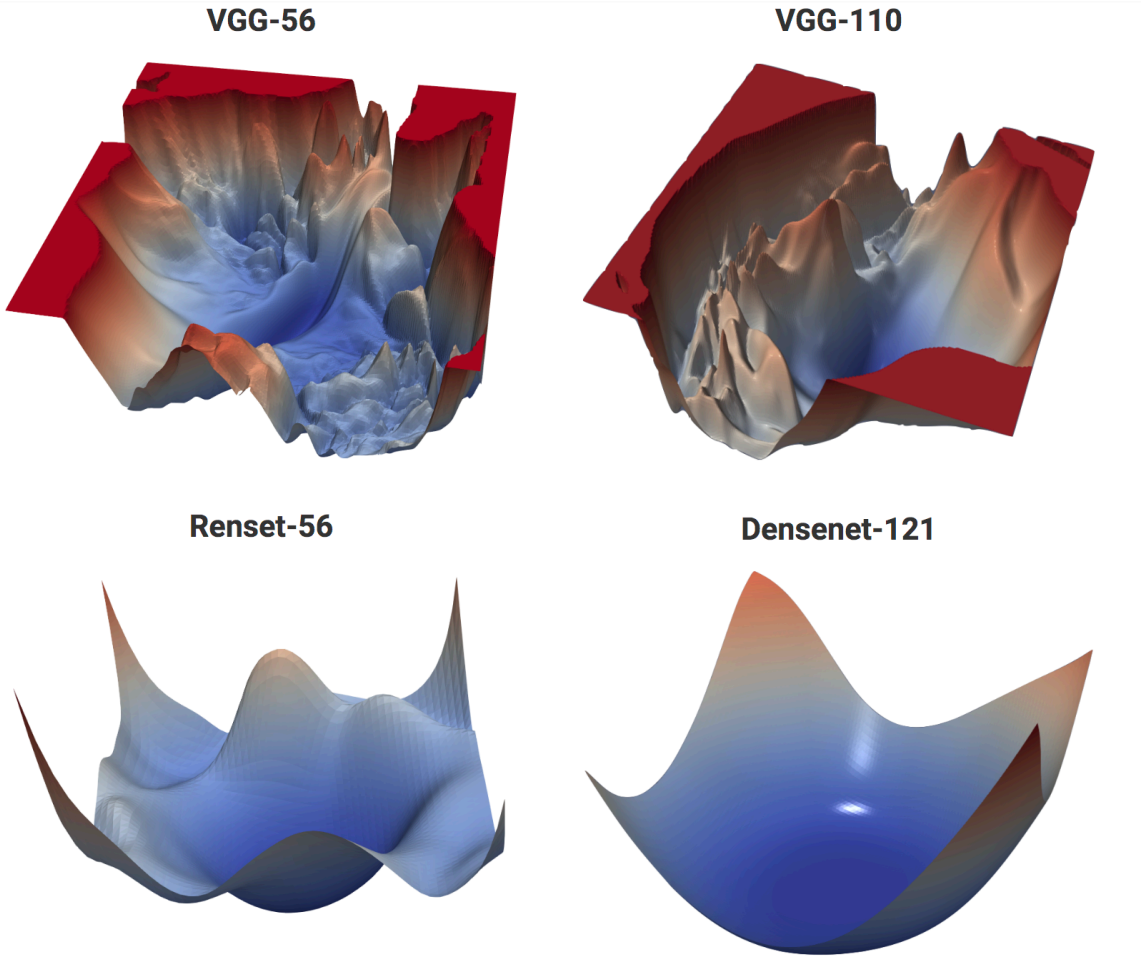
*An informal "rule-of-thumb" or intuition based approach known to yield a solution in certain situations.

Problem 1: Non-convexity

There is no formal guarantee on finding a global minimum for non-convex problems

In many cases, we settle for a solution that is “good” enough, because gradient based methods are so cheap.

We introduced several “heuristics” for escaping shallow minima, like momentum and adaptive learning rates. But one major motivation for alternative strategies is to more systematically escape shallow minima.

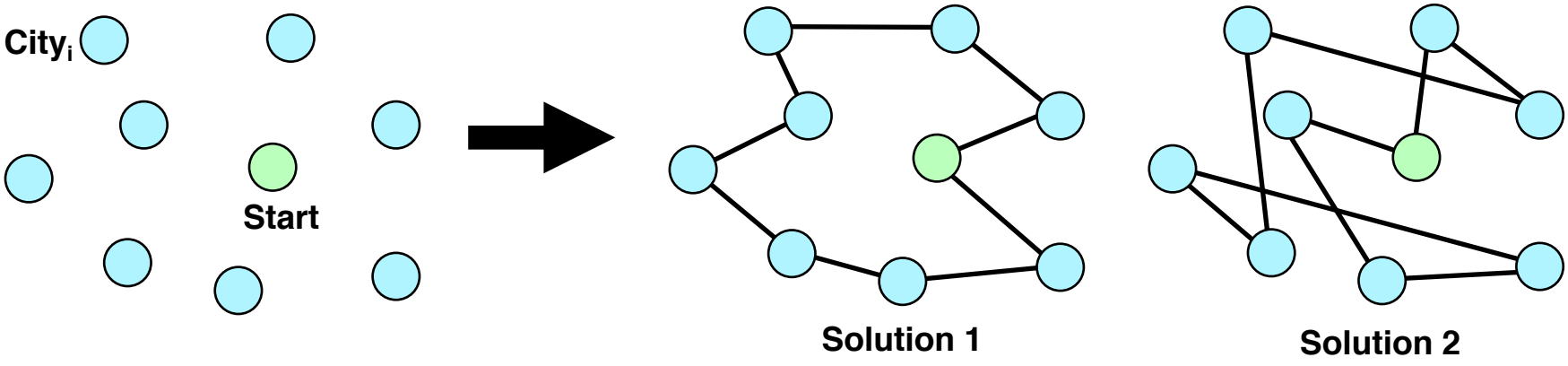


3D projection of a high dimensional loss surface associated with 4 neural networks for image recognition tasks. The lower two are designed to have relatively better convexity and smoothness. Easier to optimize but not necessarily better. https://www.cs.umd.edu/~tomg/projects/landscapes/

Problem 2: Missing Gradient

The second common situation where heuristic optimization is used, is when the analytical form of the gradient is unavailable to us either because the features are undifferentiable or because the mapping between parameters and loss function is unknown.

Consider the traveling salesman problem:



This is a case where the loss function is easy to evaluate (total distance), but the features (i.e., a graph) are non-differentiable. Another pertinent example for this audience would be molecular structures.

A hard optimization problem

For testing our methods today, we will use the **Rastrigin function**:

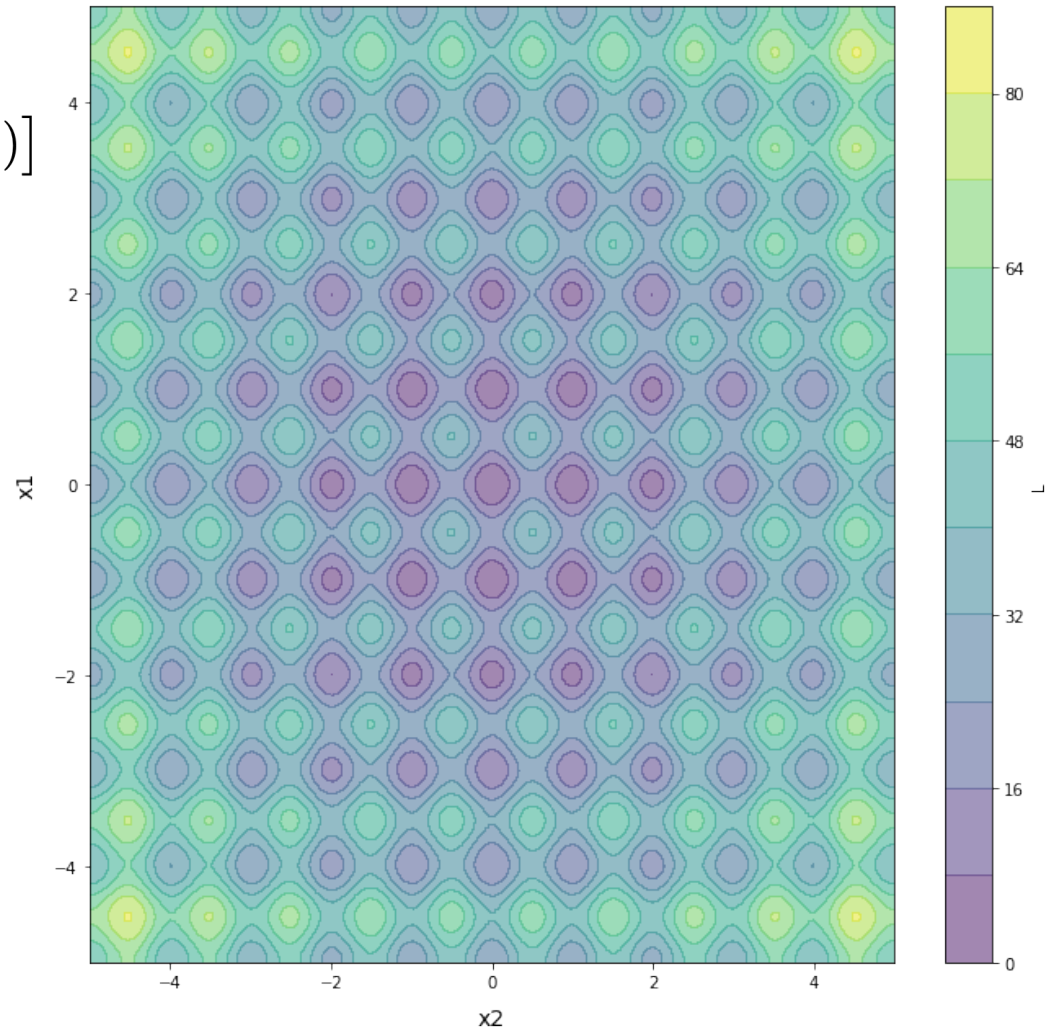
$$f(\mathbf{x}) = 10n + \sum_{i}^{n} \left[x_i^2 - 10\cos\left(2\pi x_i\right) \right]$$

Lots of local minima that would challenge any gradient-based approach.

Known solution at (0,0) in two dimensions.

Although we know the gradient in this case, we won't use it.

Sidenote: people design functions like this specifically for testing out various optimization algorithms. There are a ton of them.

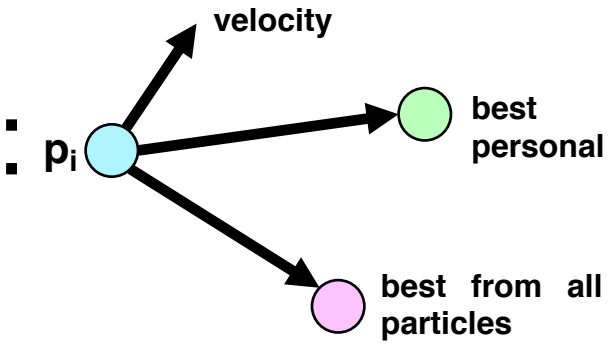


Method 1: Particle Swarm Optimization

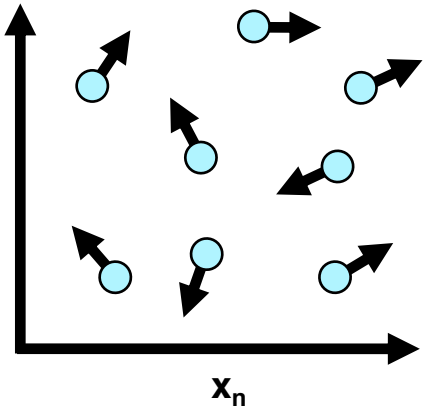
The first algorithm we will illustrate is based on the concepts of **momentum**, **random walks**, and **communication**.

The idea is to use a lot of "**walkers**", or "**particles**", to sample the optimization surface, where each particle will move in a manner that explores space while also being pushed in the direction of the best spot discovered so far **by itself or another particle**.

Each particle moves according to



The "swarm" of particles explores solution space



Governing Equations (for each particle):

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha \mathbf{v}_i^{t+1}$$

$$\mathbf{v}_i^{t+1} = \eta \mathbf{v}_i^t + \phi_1 \mathbf{r}_1 \left(\mathbf{x}_{i,\text{best}} - \mathbf{x}_i^t \right) + \phi_2 \mathbf{r}_2 \left(\mathbf{x}_{\text{all,best}} - \mathbf{x}_i^t \right)$$

η, ϕ_1, and ϕ_2 are hyperparameters that control the relative weight for momentum, local best, and global best, in directing the particle search.

PSO Implementation

Top half (initialization part) of the function:

```
# Simple PSO implementation
def pso(L,n=20,nx=2,xmin=None,xmax=None,omega=0.2,p1=0.5,p2=0.5,tmax=1000):

  # Fix the seed for reproducibility
  np.random.seed(10104041)

  # Make sure the user has supplied initialization information
  if xmin is None or xmax is None:
    print("ERROR: min/max for each variable must be supplied as arrays.")
    return

  # x initializations
  x = np.zeros([n,nx,tmax+1])
  x[:,:,0] = (np.random.random(size=(n,nx))-0.5) * np.array(xmax-xmin) + \
             np.array((xmax+xmin)/2.0)
  x_best = deepcopy(x[:,:,0])

  # v initializations
  v = np.zeros([n,nx,tmax+1])
  v[:,:,0] = ((np.random.random(size=(n,nx))-0.5)*2) * np.array(xmax-xmin) * 0.1

  # individual and global best initializations
  L_best = L(x[:,:,0])
  L_g_best = L_best[np.argmin(L_best)]
  x_g_best = x_best[np.argmin(L_best),:]
```

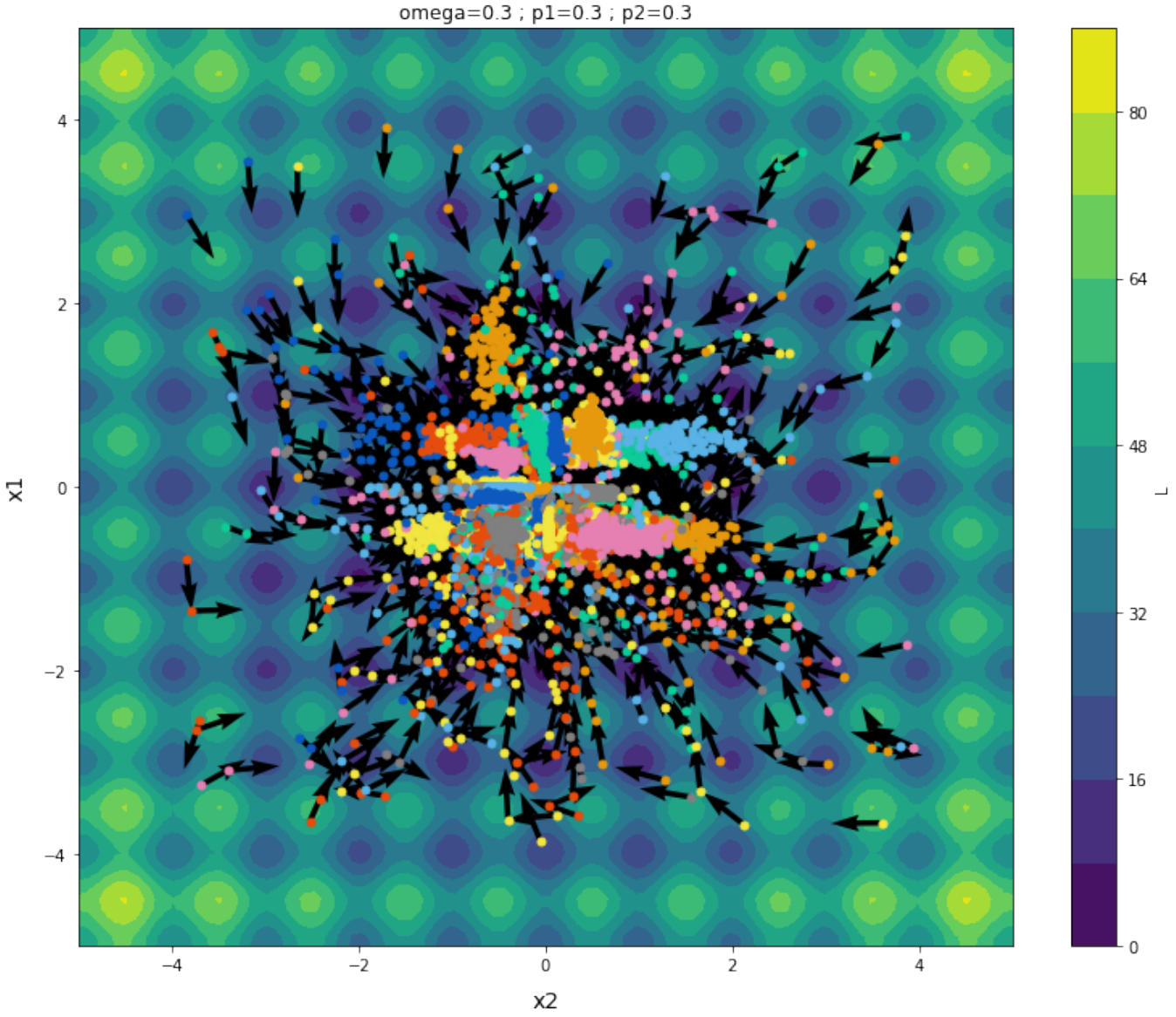
PSO Implementation

Bottom half (loop) of the function:

```
# Simple PSO implementation
def pso(L,n=20,nx=2,xmin=None,xmax=None,omega=0.2,p1=0.5,p2=0.5,tmax=1000):

  … initialization code …

  # run tmax steps
  for t in range(0,tmax):
    # Update velocities
    v[:,:,t+1] = omega*v[:,:,t] +\
                 np.random.random(size=(n,nx))*p1*(x_best - x[:,:,t]) +\
                 np.random.random(size=(n,nx))*p2*(x_g_best - x[:,:,t])

    # Update positions
    x[:,:,t+1] = x[:,:,t] + v[:,:,t+1]

    # Update best lists
    L_here = L(x[:,:,t+1])
    inds = np.where((L_here - L_best) < 0)
    x_best[inds] = x[:,:,t+1][inds]
    L_best[inds] = L_here[inds]

    # Update global
    if min(L_best) < L_g_best:
    x_g_best = x_best[np.argmin(L_best),:]
    L_g_best = min(L_best)

  return x,v,x_g_best,L_g_best
```

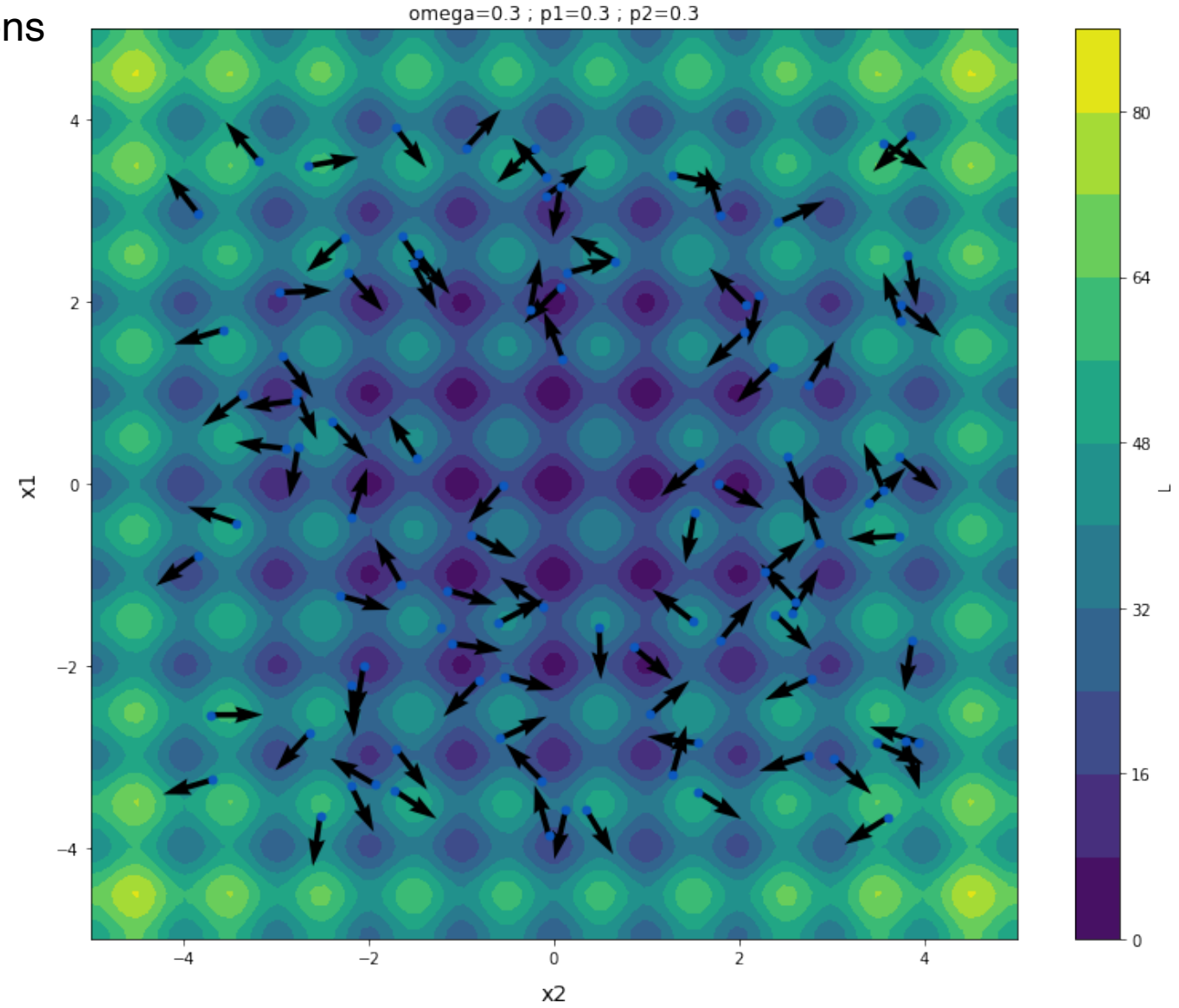
PSO Applied to the Rastrigin Function

Trajectories



PSO Applied to the Rastrigin Function

Initial positions



PSO Applied to the Rastrigin Function

Final positions

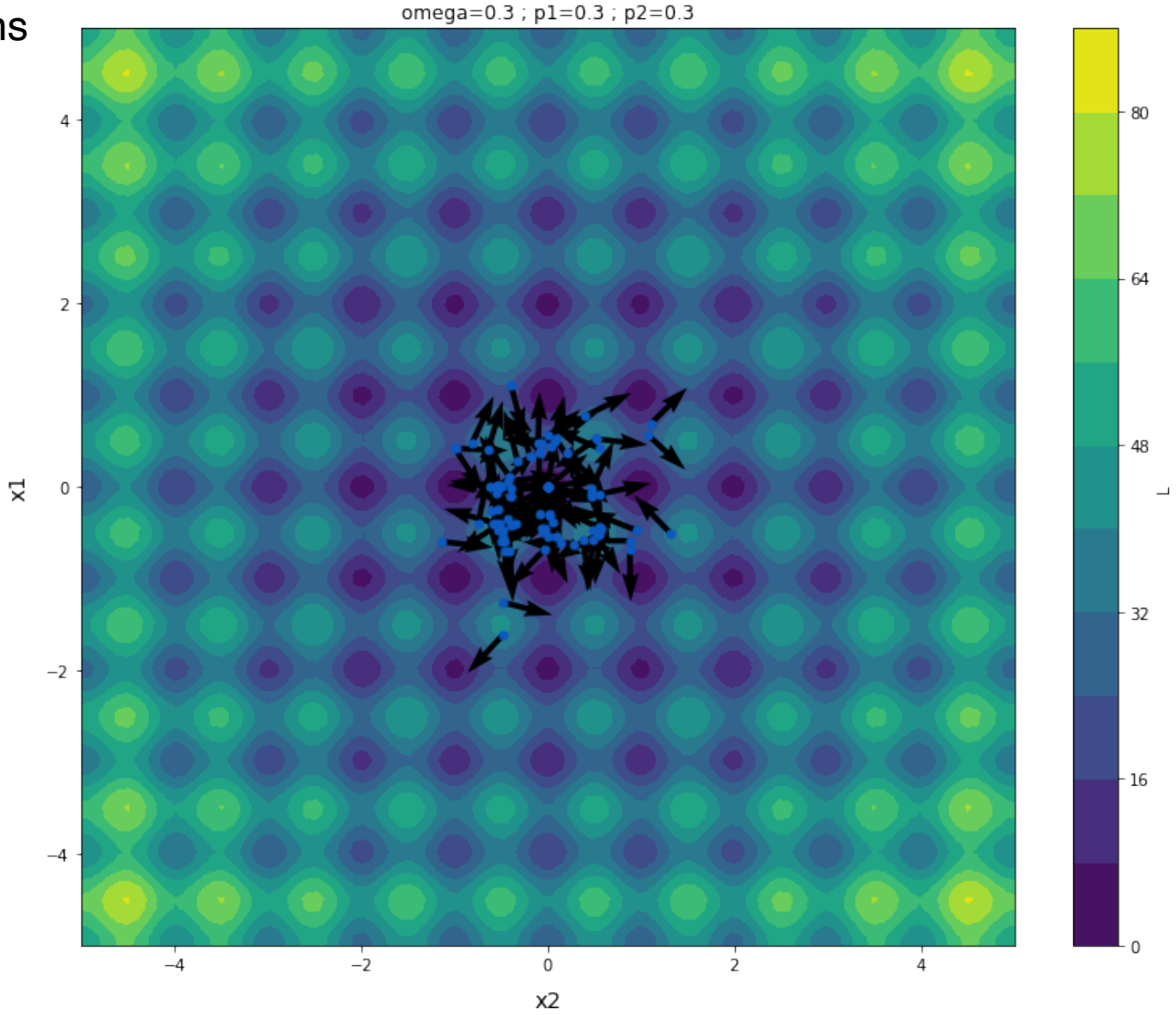
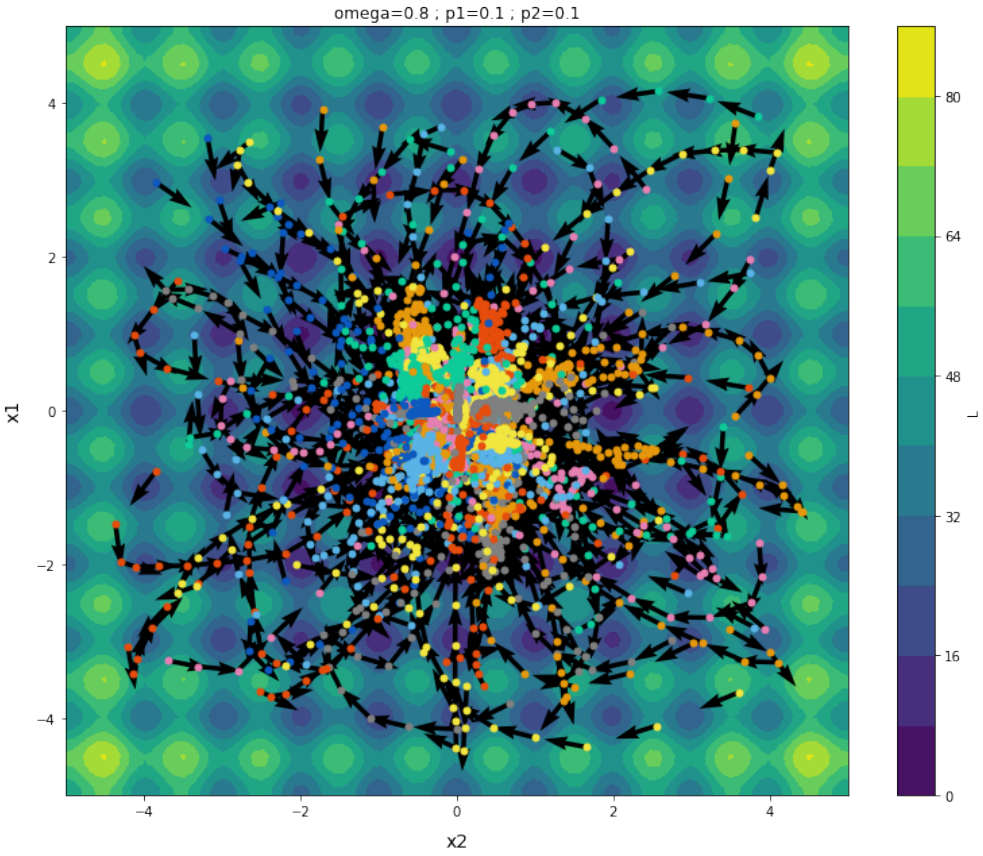


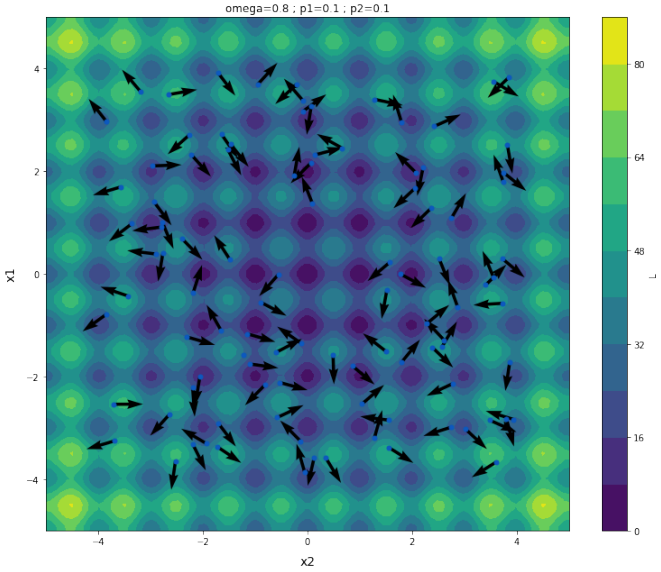
Illustration of Hyperparameter Effects

High η, low ϕ_1, low ϕ_2



Trajectories look more Newtonian. Particles slowly respond to influences of personal and global best

Initial Positions



Final Positions

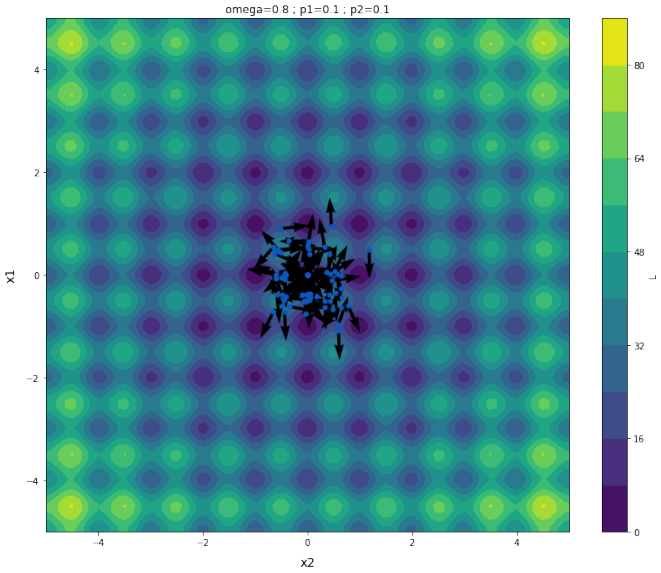
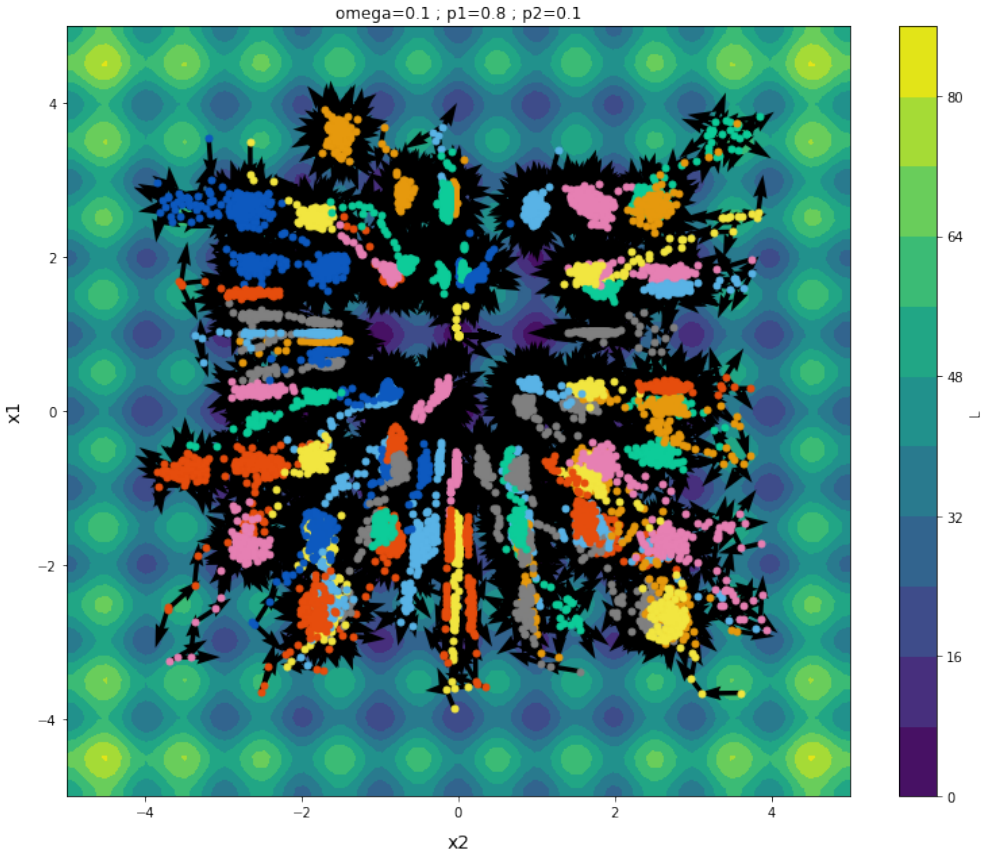


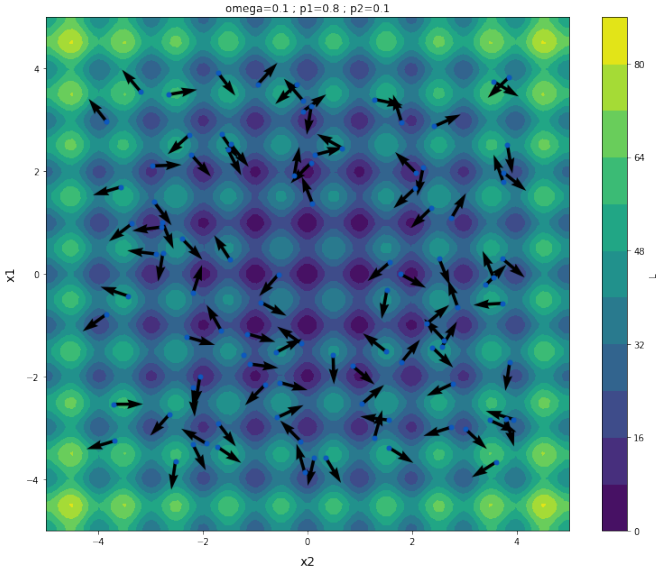
Illustration of Hyperparameter Effects

Low η, high ϕ_1, low ϕ_2



Trajectories are much more local. The particles give up exploration to stay near local best.

Initial Positions



Final Positions

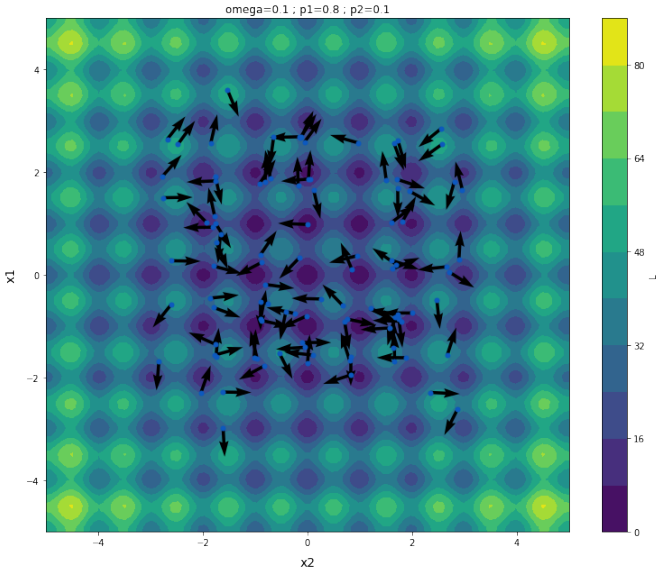
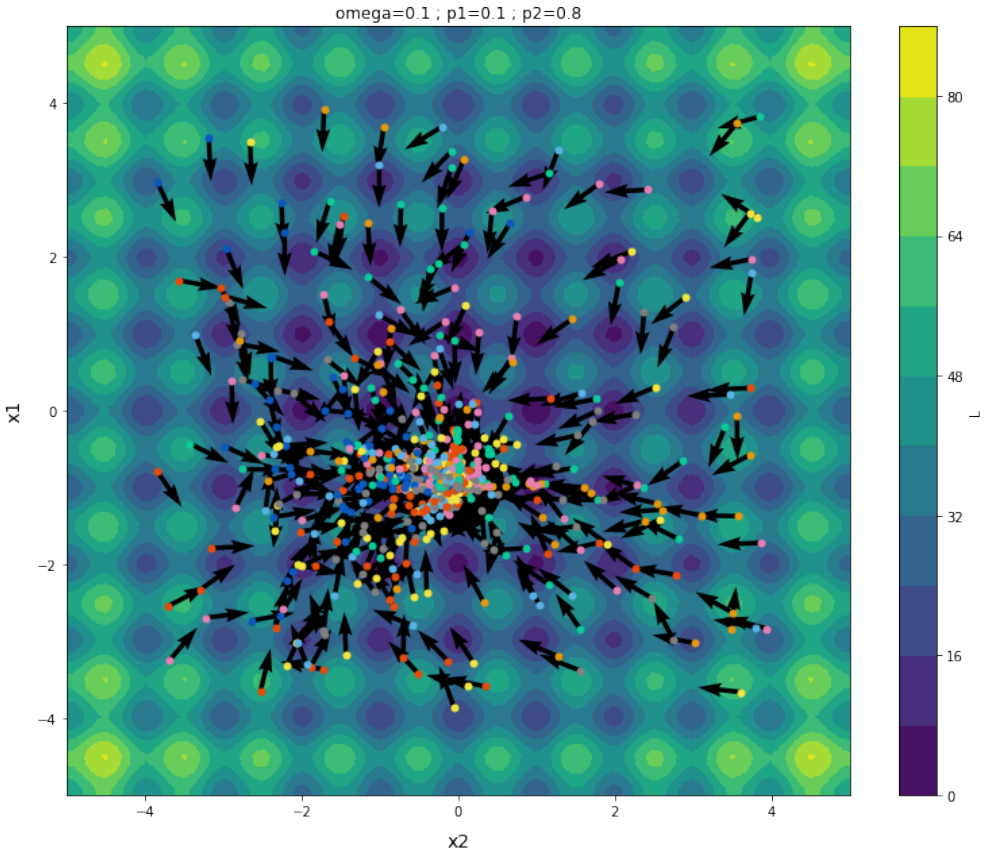


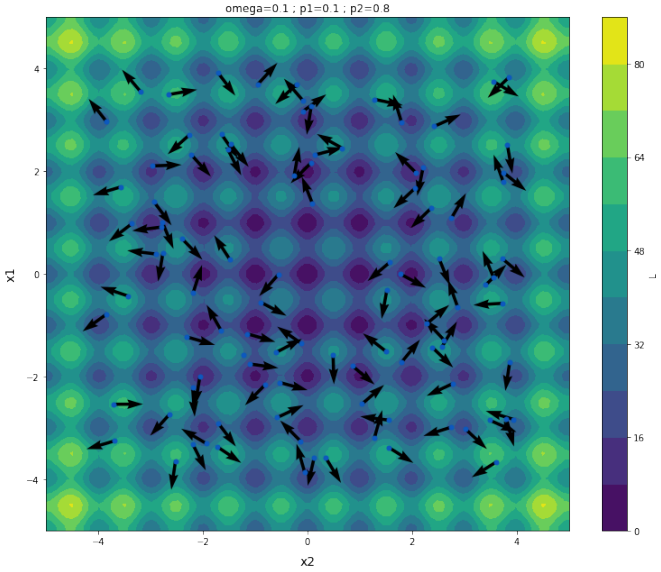
Illustration of Hyperparameter Effects

Low η, low ϕ_1, high ϕ_2

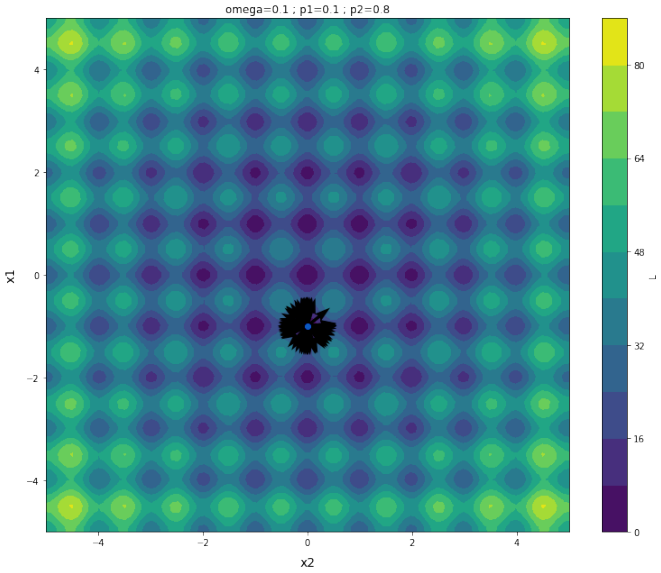


Trajectories collapse immediately to the current global best. The particles stop providing independent information after only a few iterations.

Initial Positions



Final Positions



Genetic Algorithms

See Jupyter notebook for discussion and implementation of a genetic algorithm with application to Rastrigin function.