Swarming in NuPIC

September 6, 2013

What swarming does

- Automatically determines the best CLA model for a given dataset
  - Determines which components should go into the model (encoders, spatial & temporal pooler, classifier, etc.)
  - Determines best parameters for each component

Input and output

- Input to the swarm:
  - Dataset to optimize over
  - Task you are trying to perform (i.e. “predict consumption 4 steps in advance”)
  - Optional custom error metric to measure effectiveness of the model
- Output from the swarm:
  - An OPF (Online Prediction Framework) format text file that completely describes the best model found

Behind the scenes

- Automatically builds and evaluates 100's (typically) of models using multiple processes in parallel
- Outputs the best model (that which produced the lowest error score on the provided dataset)
- Intelligently and dynamically searches the parameter space
  - A typical swarm might be searching a *huge* parameter space (ex. 5 optional fields, 2 optional model components, 15 scalar parameters, 4 enumerated parameters, …)
  - Search space is dynamically adjusted as candidate models complete

Example

- Data:

Timestamp	Temperature	Consumption
2010-07-02 00:00:00.0	68	5.3
2010-07-02 00:15:00.0	69	5.5
2010-07-02 00:30:00.0	70	5.1
…	…	…

- Swarm description:
  - Aggregate records hourly
  - Prediction “Consumption” 2 steps (2 hours) in advance
  - Consider using Timestamp, Temperature, and Consumption fields as inputs to the model

Demo

- NOTE: Swarming requires access to a MySQL database
  - Default is to look for the MySQL server on “localhost” with a username of “root” and an empty password
  - Different MySQL credentials can be entered by editing <nupic>/conf/default/nupic-default.xml
- To run demo

```
run_swarm.py examples/swarm/simple/search_def.json
      --maxWorkers=4
```

Swarm description file

- JSON format text file containing swarming parameters
- Example can be found at <nupic>/examples/swarm/simple/search_def.json

Section	Description
includedFields	Describes which fields of the csv data file to consider as inputs to the model
streamDef	Name of the csv data file and amount of aggregation (if any) to perform
inferenceType	Type of model you want to create
inferenceArgs	Name of the predicted field and number of steps in advance to predict
iterationCount	Number of aggregated records to run through each model
swarmSize	How exhaustive of a search to perform

Example CSV data file

```
gym,address,timestamp,consumption
string,string,datetime,float
S,,T,
Condamine Street Balgowlah 2093,2010-07-02 00:00:00.0,5.3
Condamine Street Balgowlah 2093,2010-07-02 00:15:00.0,5.5
Condamine Street Balgowlah 2093,2010-07-02 00:30:00.0,5.1
Condamine Street Balgowlah 2093,2010-07-02 00:45:00.0,5.3
Condamine Street Balgowlah 2093,2010-07-02 01:00:00.0,5.2
…
```

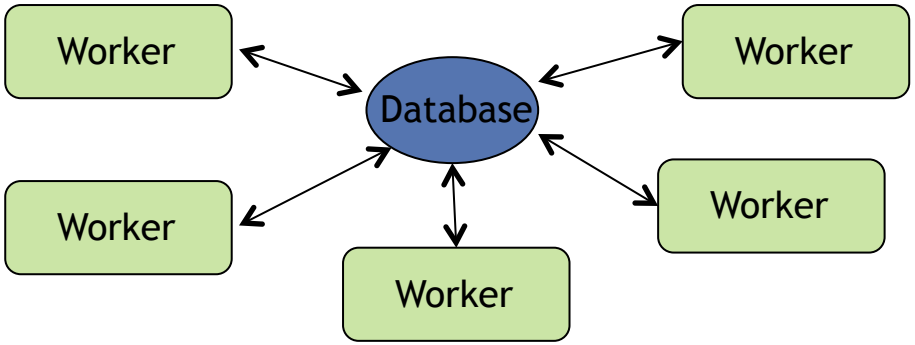
- First 3 rows are special header rows
  - Field names
  - Field types
  - Field flags

Leveraging multiple processes

- The more processes that swarming has access to, the more models it can run simultaneously and the faster the swarm can complete

Leveraging multiple processes

- The more processes that swarming has access to, the more models it can run simultaneously and the faster the swarm can complete
- Every process (called a “swarm worker”) is an equal citizen, there is no central coordinator.
  - No single point of failure
  - Swarm workers can even be added/removed as the swarm progresses
- A MySQL database is used to maintain results obtained from each candidate model



Swarm worker pseudo-code

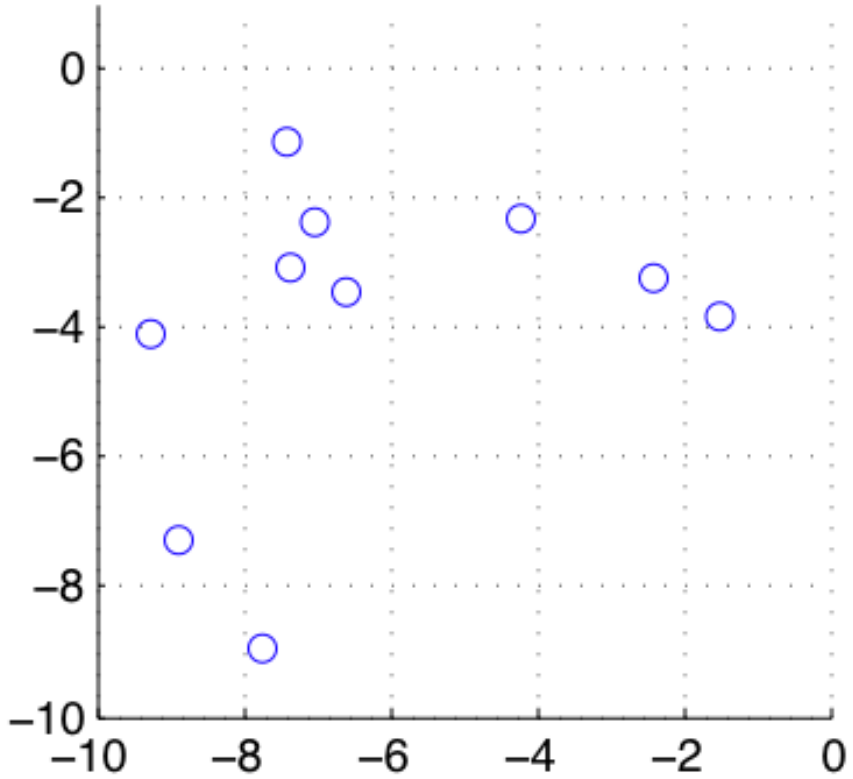
- While (search is not complete):
  - Fetch latest model results from the database
    - This includes the parameters and error score for each model that has been evaluated so far by other workers
  - Given the current stage of the swarm and the history of model results obtained so far, determine parameters for a new candidate model
  - Create entry for this new model in the database
  - Instantiate the model and run the dataset through it, updating results in the database periodically

Swarming algorithms

- Swarming determines:
  - Which input fields should be fed into the model
  - What components should be in the model (types of encoders, spatial and temporal pooler, classifier, etc.)
  - Scalar and enumerated parameters for each component
- Three different algorithms are leveraged
  - A Particle Swarm Optimization (PSO) algorithm is used to determine the values for scalar parameters
  - A custom algorithm is used to determine the values for enumerated parameters and which optional components should be in the model
  - A custom field search algorithm is used to determine which input fields should be used

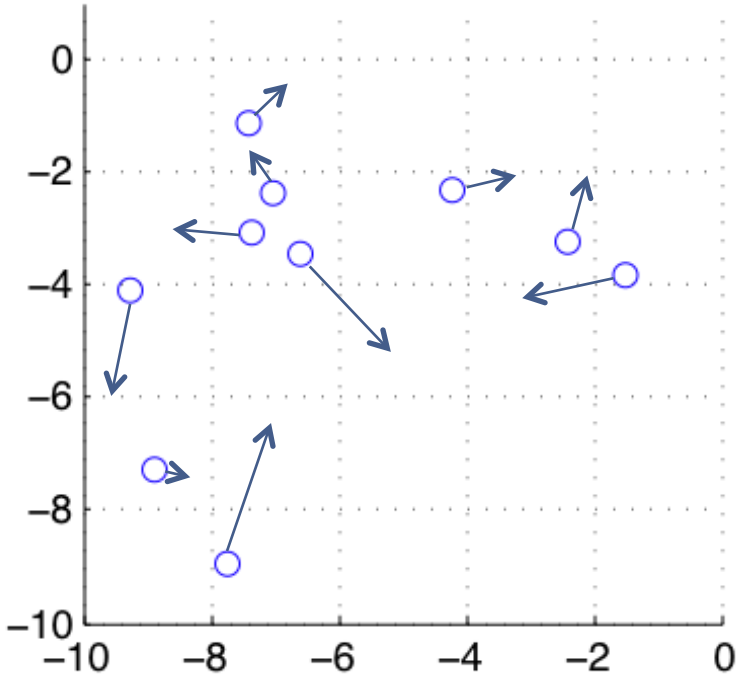
Particle Swarm Optimization (PSO)

- PSO used to determine values for scalar parameters
- A bunch of particles (models) are placed spread out within the parameter space



Particle Swarm Optimization (PSO)

- After each particle completes and gets an error score, it moves to a new position within the parameter space
- The new position is chosen such that the particle tends to move towards the global min (among all other particles) as well as towards the local min (that this particle has seen so far).



PSO Details

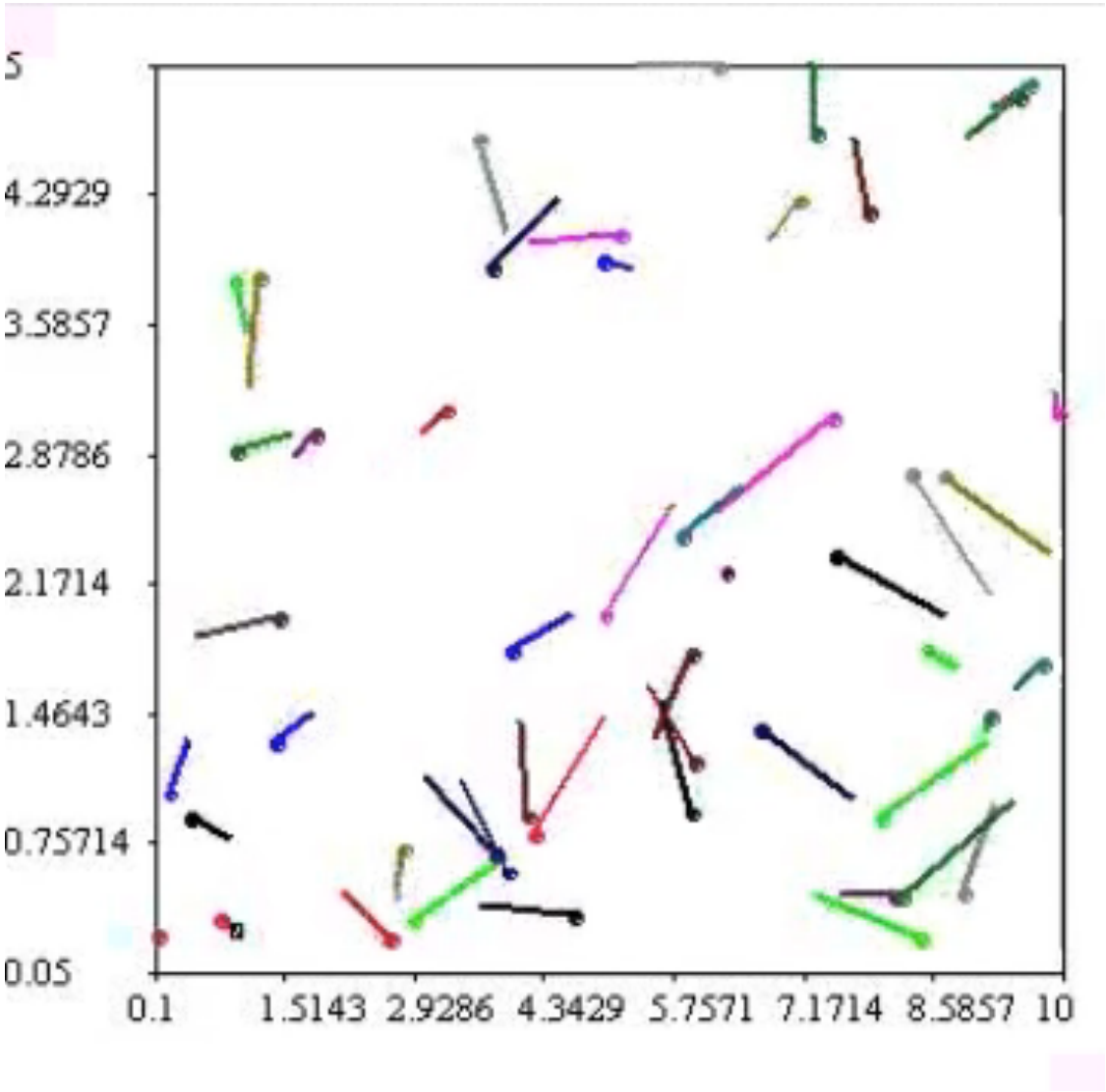
- Particles
  - Each particle is an instance of a model
  - Particle i initialized with random position x_i and velocity v_i
  - Each particle keeps track of its best position to date p_i
- Measure fitness of each particle at every iteration
  - The global best position is denoted g_{best}
- Update particles according to the equation:

$$v_i = v_i + \varphi 1 * rnd() * (p_i - x_i) + \varphi 2 * rnd() * (g_{best} - x_i)$$

$$x_i = x_i + v_i$$

- Intuition:
  - Particles move in a direction that blends local and global best
  - Random component helps explore space more thoroughly

PSO In Action



Optimizing enumerated parameters

- Examples of enumerated parameters:
  - encoderType: [“scalar”, “delta”, “adaptive”]
  - spLearning: [True, False]
- As each model completes, we build up an average error score for each choice. Ex:
  - “scalar”: 0.1, “delta”: 0.2, “adaptive”: 0.5
- When a new model is created, we choose a new enumerated value using a weighted probability:
  - Enumerate value with lowest average error score so far has highest probability of being chosen
- Enumerated parameters are also used to determine if optional components should be included:
  - includeTP: [True, False]

Field selection logic

- The swarm also figures out which fields are useful to the model
- Each field combination represents a different parameter space
  - Dozens of models are run for each field combination
- First, single-field models are evaluated (this is called “sprint 0”)
  - Ex. fieldA, fieldB, fieldC, fieldD
- Then, build up 2-field models using the best single-field from “sprint 0” (this is called “sprint 1”):
  - Ex. fieldA + fieldB, fieldA + fieldC, fieldA + fieldD
- Then, build up 3 field models using best 2-field model from “sprint 1”
  - Ex. fieldA + fieldC + fieldD
- When results stop improving, the swarm ends

Advanced: Customizing the swarm

- When you run a swarm using a JSON swarm description file as input: ("`run_swarm.py search_def.json`")
  1. description.py and permutations.py files are generated
  2. Swarm logic runs off of these files
- The permutations.py file determines which parameters of the model are optimized by the swarm
  - Controls the type, range, and granularity of each parameter
- For a custom swarm, you can edit the permutations.py file to optimize different parameters of the model
- To run a custom swarm, simply feed in the permutations file:
  - `run_swarm.py my_exp/permutations.py`