Automated Machine Learning
(AutoML)

Noor Awad M. Lindauer F. Hutter

University of Freiburg

Lecture 5:
Hyperparameter Optimization using Evolutionary Algorithms

Where are we? The big picture

- Introduction
- Background
    - Design spaces in ML
    - Evaluation and visualization
- $\rightarrow$ Hyperparameter optimization (HPO)
    - Bayesian optimization
    - $\rightarrow$ Evolutionary Algorithms (EAs)
    - Speeding up HPO with multi-fidelity optimization
- Pentecost (Holiday) – no lecture
- Architecture search I + II
- Meta-Learning
- Learning to learn & optimize
- Beyond AutoML: algorithm configuration and control
- Project announcement and closing

Evolutionary Algorithms

Task: [1min]

Who knows evolutionary algorithms or nature inspired algorithms? If so, name some examples?

Learning Goals

After this lecture, you will be able to . . .

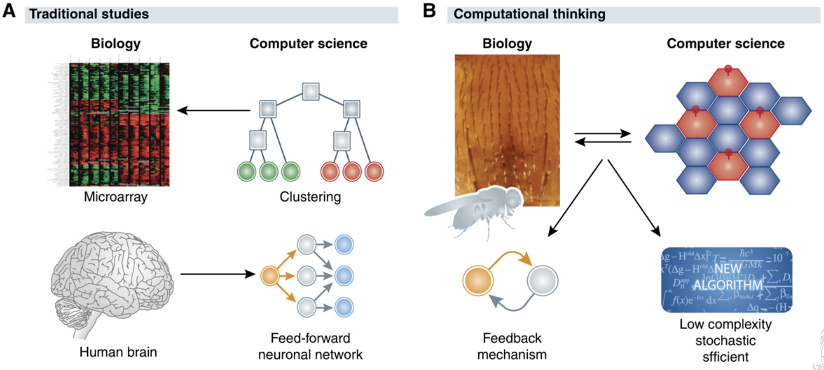
- explain the basics of evolutionary algorithms
- discuss the different types of evolutionary algorithms
- efficiently tune HPO using evolutionary algorithms
- discuss importance of evolutionary algorithms to solve many optimization problems

Lecture Overview

1. Basics of Evolutionary Algorithms
2. How Evolutionary Algorithms Work
3. Different Types of Evolutionary Algorithms
4. EAs for Hyperparameter Optimization
5. Practical Considerations

Nature Inspired Algorithms

- Nowadays, most new algorithms are nature-inspired, because they have been developed by drawing inspiration from nature
    - majority of these algorithms are based on some successful characteristics of biological system



Nature Inspired Algorithms for Optimization [Fister et al. 2013]

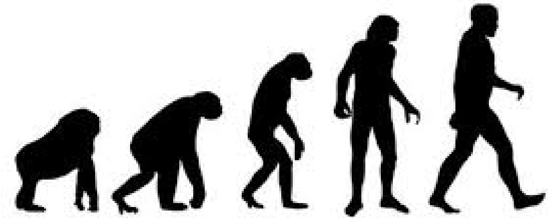
Swarm intelligence based algorithms			Bio-inspired (not SI-based) algorithms		
Algorithm	Author	Reference	Algorithm	Author	Reference
Accelerated PSO	Yang et al.	[69], [71]	Atmosphere clouds model	Yan and Hao	[67]
Ant colony optimization	Dorigo	[15]	Biogeography-based optimization	Simon	[56]
Artificial bee colony	Karaboga and Basturk	[31]	Brain Storm Optimization	Shi	[55]
Bacterial foraging	Passino	[46]	Differential evolution	Storn and Price	[57]
Bacterial-GA Foraging	Chen et al.	[6]	Dolphin echolocation	Kaveh and Farhoudi	[33]
Bat algorithm	Yang	[78]	Japanese tree frogs calling	Hernández and Blum	[28]
Bee colony optimization	Teodorović and Dell'Orco	[62]	Eco-inspired evolutionary algorithm	Parpinelli and Lopes	[45]
Bee system	Lucic and Teodorovic	[40]	Egyptian Vulture	Sur et al.	[59]
BeeHive	Wedde et al.	[65]	Fish-school Search	Lima et al.	[14], [3]
Wolf search	Tang et al.	[61]	Flower pollination algorithm	Yang	[72], [76]
Bees algorithms	Pham et al.	[47]	Gene expression	Ferreira	[19]
Bees swarm optimization	Drias et al.	[16]	Great salmon run	Mozaffari	[43]
Bumblebees	Comellas and Martinez	[12]	Group search optimizer	He et al.	[26]
Cat swarm	Chu et al.	[7]	Human-Inspired Algorithm	Zhang et al.	[80]
Consultant-guided search	Iordache	[29]	Invasive weed optimization	Mehrabian and Lucas	[42]
Cuckoo search	Yang and Deb	[74]	Marriage in honey bees	Abbass	[1]
Eagle strategy	Yang and Deb	[75]	OptBees	Maia et al.	[41]
Fast bacterial swarming algorithm	Chu et al.	[8]	Paddy Field Algorithm	Premaratne et al.	[48]
Firefly algorithm	Yang	[70]	Roach infestation algorithm	Havens	[25]
Fish swarm/school	Li et al.	[39]	Queen-bee evolution	Jung	[30]
Good lattice swarm optimization	Su et al.	[58]	Shuffled frog leaping algorithm	Eusuff and Lansey	[18]
Glowworm swarm optimization	Krishnanand and Ghose	[37], [38]	Termite colony optimization	Hedayatzadeh et al.	[27]
Hierarchical swarm model	Chen et al.	[5]	Physics and Chemistry based algorithms		
Krill Herd	Gandomi and Alavi	[22]	Big bang-big Crunch	Zandi et al.	[79]
Monkey search	Mucherino and Seref	[44]	Black hole	Hatamlou	[24]
Particle swarm algorithm	Kennedy and Eberhart	[35]	Central force optimization	Formato	[21]
Virtual ant algorithm	Yang	[77]	Charged system search	Kaveh and Talatahari	[34]
Virtual bees	Yang	[68]	Electro-magnetism optimization	Cuevas et al.	[13]
Weightless Swarm Algorithm	Ting et al.	[63]	Galaxy-based search algorithm	Shah-Hosseini	[53]
Other algorithms			Gravitational search	Rashedi et al.	[50]
Anarchic society optimization	Shayeghi and Dadashpour	[54]	Harmony search	Geem et al.	[23]
Artificial cooperative search	Civicioglu	[9]	Intelligent water drop	Shah-Hosseini	[52]
Backtracking optimization search	Civicioglu	[11]	River formation dynamics	Rabanal et al.	[49]
Differential search algorithm	Civicioglu	[10]	Self-propelled particles	Vicsek	[64]
Grammatical evolution	Ryan et al.	[51]	Simulated annealing	Kirkpatrick et al.	[36]
Imperialist competitive algorithm	Atashpaz-Gargari and Lucas	[2]	Stochastic difusion search	Bishop	[4]
League championship algorithm	Kashan	[32]	Spiral optimization	Tamura and Yasuda	[60]
Social emotional optimization	Xu et al.	[66]	Water cycle algorithm	Eskandar et al.	[17]

Nature Inspired Algorithms for Optimization

Evolutionary Computation (EC)

is a family of algorithms for global optimization which are inspired by biological evolution

- Evolutionary Algorithms (EAs) are subset of EC, and generic population-based optimization algorithms.

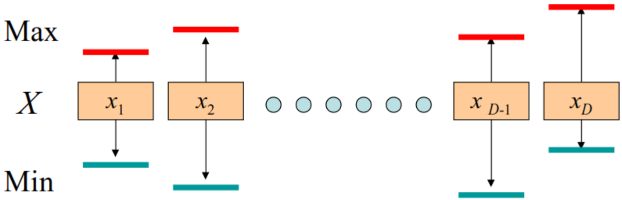


Lecture Overview

Representation

Each solution of the problem being solved is namely an individual

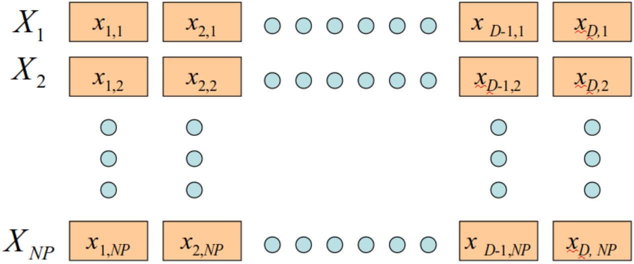
- solutions are represented as vectors of size D with each value taken from some domain
- solutions can also be matrices, network or probably any data structure
- each parameter x_i in X should be within the search range of the problem being solved $[Min, Max]$



How Evolutionary Algorithms Work

Maintain Population

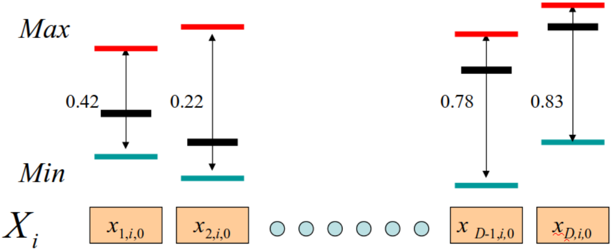
Maintain a population of size NP



Maintain Population

Maintain a population of size NP

- initialize NP D- dimensional individuals uniformly distributed within the search space
- different $rand$ values are instantiated for each i and j



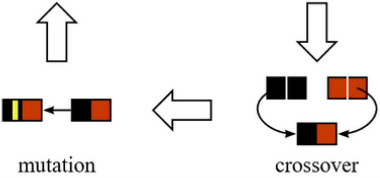
$$x_{j,i,0} = x_{j,\min} + rand_{i,j}[0,1] \cdot (x_{j,\max} - x_{j,\min})$$

How Evolutionary Algorithms Work

Evolve Population

Generate a new child (offspring) through mutation and crossover operations

- mutation operation is to generate new mutant vector from the the current population/parents based on some criterion(strategy)
- crossover operation is to combine the parent and mutant vector into one final offspring (i.e. inherit some features from parent and some from mutant vector)

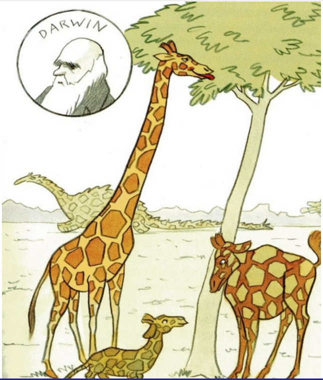


How Evolutionary Algorithms Work

Parent Selection Mechanism

"Survival of the fitter principle": each offspring is compared with its parent(s) and the one with a better fitness is forwarded to the next generation population

- high quality solutions more likely to become parents than low quality

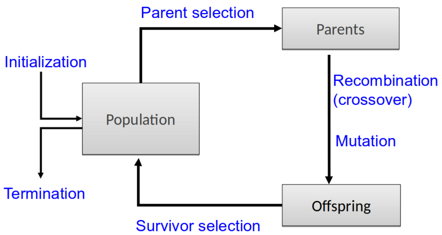


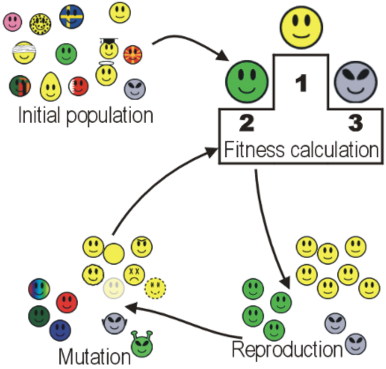
How Evolutionary Algorithms Work

- The basic steps of an evolutionary algorithm are:
  - Initialization, Mutation, Recombination, Selection

How Evolutionary Algorithms Work

- The basic steps of an evolutionary algorithm are:
  - Initialization, Mutation, Recombination, Selection





- Mutation and recombination can be performed in any order

[source]

How Evolutionary Algorithms Work

Algorithm 1: Optimization with EA

Input : black-box function f, dimension D, maximal number of function evaluations FE_{max}, population size NP, mutation rate and crossover rate

$g = 0$, $FE = 0$;

How Evolutionary Algorithms Work

Algorithm 2: Optimization with EA

Input : black-box function f, dimension D, maximal number of function evaluations FE_{max}, population size NP, mutation rate and crossover rate

$g = 0$, $FE = 0$;
$pop_g \leftarrow$ initial_population(NP, D);
$fitness_g \leftarrow$ evaluate_population(pop_g);
$FE = NP$;

How Evolutionary Algorithms Work

Algorithm 3: Optimization with EA

Input : black-box function f, dimension D, maximal number of function evaluations FE_{max}, population size NP, mutation rate and crossover rate

$g = 0$, $FE = 0$;
$pop_g \leftarrow$ initial_population(NP, D);
$fitness_g \leftarrow$ evaluate_population(pop_g);
$FE = NP$;
while $(FE < FE_{max})$ **do**
    mutate(pop_g);
    $offspring_g \leftarrow$ crossover(pop_g);
    $fitness_g \leftarrow$ evaluate_population($offspring_g$);

Algorithm 4: Optimization with EA

Input : black-box function f, dimension D, maximal number of function evaluations FE_{max}, population size NP, mutation rate and crossover rate

$g = 0$, $FE = 0$;
$pop_g \leftarrow$ initial_population(NP, D);
$fitness_g \leftarrow$ evaluate_population(pop_g);
$FE = NP$;
while $(FE < FE_{max})$ **do**
    mutate(pop_g);
    $offspring_g \leftarrow$ crossover(pop_g);
    $fitness_g \leftarrow$ evaluate_population($offspring_g$);
    pop_{g+1},$fitness_{g+1} \leftarrow$ select(pop_g,$offspring_g$);
    $FE = FE + NP$;
    $g = g$+1;
return *best individual* X *in* pop *according to* $fitness$

EAs Applied to HPO

Task: [2min]

Why might EAs be an interesting approach for HPO and many other real-life applications?

EAs Applied to HPO

Task: [2min]
Why might EAs be an interesting approach for HPO and many other real-life applications?

- very intuitive to use EAs for hyper-parameter tuning/optimization
- can handle different data types
- driven by surprisingly simple operations, nevertheless produced astonishing results
- time complexity is low compared to other model-based methods
  - no need to fit a model and evaluate it on validation data, hence the process is not expensive like in BO
  - parallelizable as a population of NP individuals can be evaluated in parallel on NP machines

EAs Applied to HPO

General Concept

We will generally define HPO problem as we wish to find the best combination of parameters that maximizes/minimizes some objective function (or fitness) by

- define a population of NP random individuals (or solutions) $\rightarrow pop_{g0}$
- start the evolutionary search by evolving pop_g using mutation (and/or) crossover $\rightarrow popnew_g$
- survival of the fittest $\rightarrow$ best individuals are now pop_{g+1}
- we will accept a final solution once we have either run the algorithm for some maximum number of iterations, or we have reached some fitness threshold

Evolutionary Algorithms

Pros and Cons of classical EAs

Pros:

- derivative-free methods
- can solve variety of optimization problems and real-world applications

Evolutionary Algorithms

Pros and Cons of classical EAs

Pros:

- derivative-free methods
- can solve variety of optimization problems and real-world applications
- highly parallelizable
- conceptually simple, yet powerful enough to solve complex problems
- time complexity is low compared to other algorithms

Evolutionary Algorithms

Pros and Cons of classical EAs

Pros:

- derivative-free methods
- can solve variety of optimization problems and real-world applications
- highly parallelizable
- conceptually simple, yet powerful enough to solve complex problems
- time complexity is low compared to other algorithms

Cons:

- stagnation: optimization process does not progress anymore
- premature convergence: algorithm converges to a single solution, but that solution is not as high quality as expected

Evolutionary Algorithms

Pros and Cons of classical EAs

Pros:

- derivative-free methods
- can solve variety of optimization problems and real-world applications
- highly parallelizable
- conceptually simple, yet powerful enough to solve complex problems
- time complexity is low compared to other algorithms

Cons:

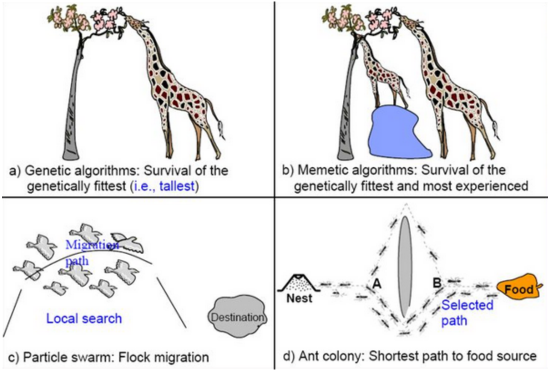
- stagnation: optimization process does not progress anymore
- premature convergence: algorithm converges to a single solution, but that solution is not as high quality as expected
- diversity of population structures: loss of population diversity for solving complex optimization problems
- lacks a good balance between exploration and exploitation

Lecture Overview

Different Types of Evolutionary Algorithms

Task: [2min]

Although different EAs are similar at highest level, each of these varieties implements an EA in a different manner. What are the possible differences?



Different Types of Evolutionary Algorithms

Task: [2min]

Although different EAs are similar at highest level, each of these varieties implements an EA in a different manner. What are the possible differences?

The differences include almost all aspect of evolutionary search including:

- choices of representation for individual structures
- **forms of mutation and crossover operations**
- types of selection mechanism used
  - there is more than one kind of selection, e.g. best of offspring, or best of offspring and parents, i.e. selection can be more complex than comparing an offspring to a parent
- measures of performance

Evolution Strategy (ES) [Beyer and Schwefel. 2002]

An optimization technique based on ideas of evolution.

- introduced in the early 1960s and then it developed further
- search operators are mutation and selection

Evolution Strategy (ES) [Beyer and Schwefel. 2002]

An optimization technique based on ideas of evolution.

- introduced in the early 1960s and then it developed further
- search operators are mutation and selection
- mutation operator adds a random number to each vector component, where the magnitude of the added random number is called step-size, and usually governed by
  - self-adaptation mechanism uses multivariate normal distribution
  - covariance matrix adaptation (CMA-ES)

Evolution Strategy (ES) [Beyer and Schwefel. 2002]

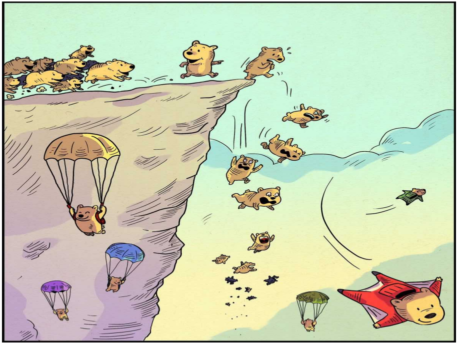
An optimization technique based on ideas of evolution.

- introduced in the early 1960s and then it developed further
- search operators are mutation and selection
- mutation operator adds a random number to each vector component, where the magnitude of the added random number is called step-size, and usually governed by
  - self-adaptation mechanism uses multivariate normal distribution
  - covariance matrix adaptation (CMA-ES)
    - represents the pairwise dependencies between the variables
    - learns a second order model of the objective function using the ranking of candidate solutions (NOT the derivative nor function values are required)

Evolution Strategy (ES)

Selection operator is based on fitness rankings, not actual fitness values

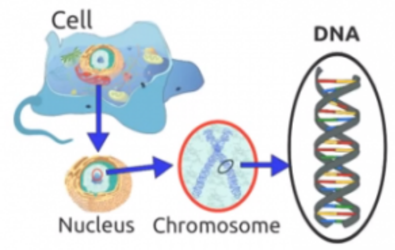
- $(1+\lambda)$−ES $\rightarrow$ general ES rule in which λ mutant vectors are generated and compete with the parent, and the best mutant becomes the parent for next generation
- $(1+1)$−ES $\rightarrow$ simplest ES



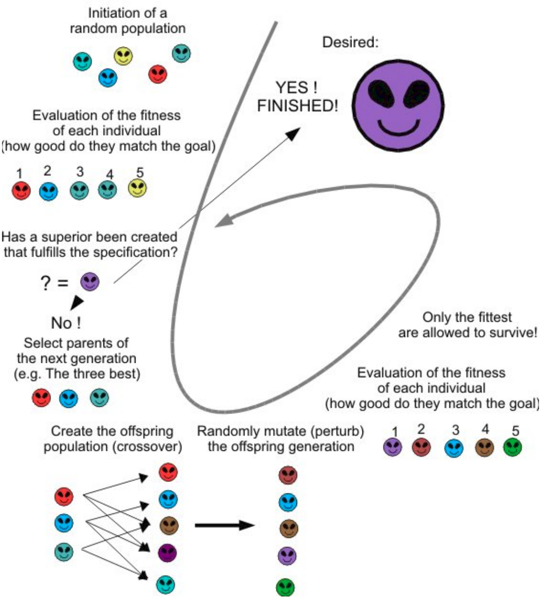
Genetic Algorithm (GA) [Mitchell. 1998]

The most popular type of EA which is inspired from the process of natural selection

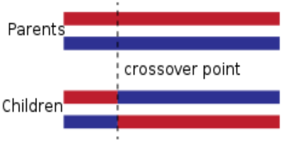
- introduced by Holland in 1960 based on the concept of Darwin's theory of evolution
- requires a genetic representation of the solution domain and a fitness function to evaluate the solution domain → encoding
- applies search operators which are crossover and mutation (sometimes both, sometimes one)



How GA works



Single-point crossover



Bit Flip mutation

0	0	1	1	0	1	0	0	1	0
0	0	1	0	0	1	0	0	1	0

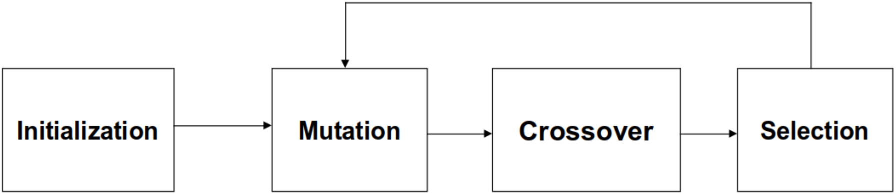
[source]

Differential Evolution (DE) [Storn and Price. 1995]

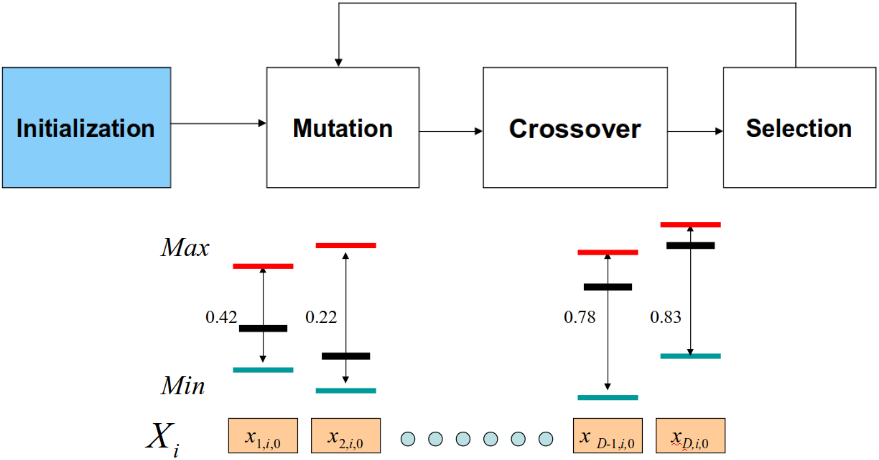
One of the simplest, yet effective, stochastic direct search EA

- introduced by Storn and Price in 1995
- outperformed several variants of GA and other EAs over a wide variety of optimization problems
- very easy to implement in any standard programming language
- very few control parameters (typically three for a standard DE) and their effects on performance have been well studied
- complexity is very low as compared to some of the most competitive continuous optimizers like CMA-ES

How DE Works

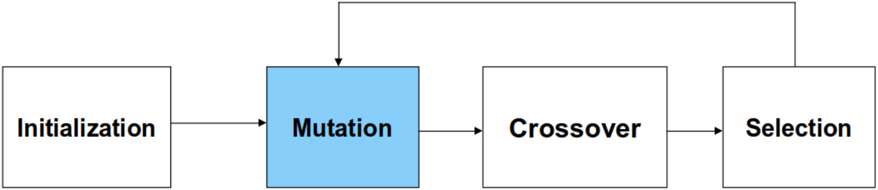


Differential Evolution (DE)



$$x_{j,i,0} = x_{j,\min} + rand_{i,j}[0,1] \cdot (x_{j,\max} - x_{j,\min})$$

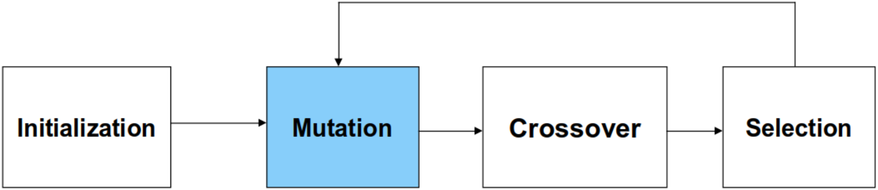
Differential Evolution (DE)

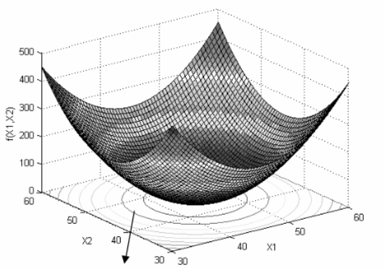


- There are many mutation strategies, classical one is DE\rand\1
- For each individual, randomly select three different individuals
- Add the weighted difference of two of the parameter vectors to the third to form a donor vector

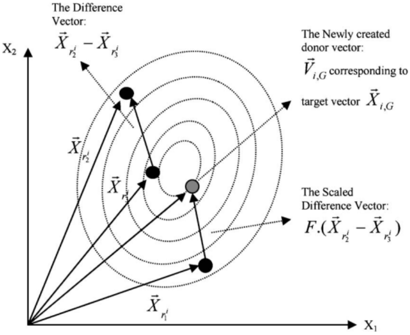
$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G})$$

- The scaling factor F is a random number from (0, 2)

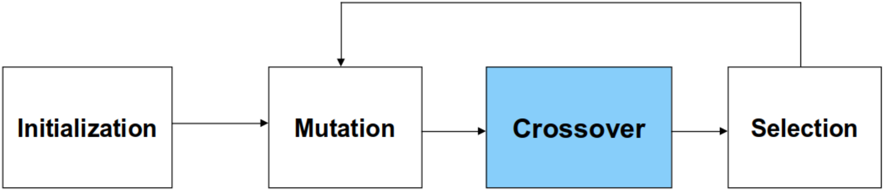




Constant cost contours of Sphere function



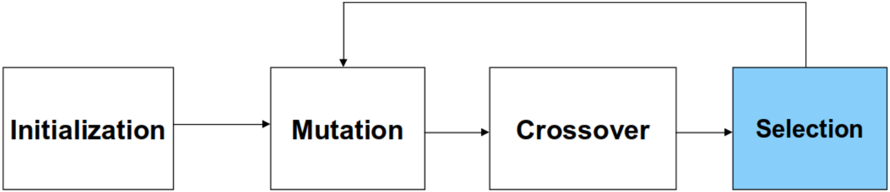
Differential Evolution (DE)



- Binomial (Uniform) Crossover:
- Components of the donor vector enter into the trial offspring vector in the following way
  Let j_{rand} be a randomly chosen integer between 1,...,D, and CR is the crossover rate

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } (rand_{i,j}[0,1) \leq Cr \text{ or } j = j_{rand}) \\ x_{j,i,G}, & \text{otherwise,} \end{cases}$$

Differential Evolution (DE)



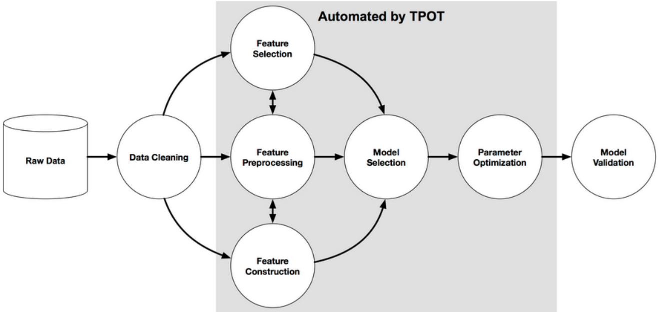
- "Survival of the fitter" principle (minimization problem)

$$X_{i,G+1} = U_{i,G}, \text{ if } f(U_{i,G}) \leq f(X_{i,G})$$
$$= X_{i,G}, \text{ if } f(U_{i,G}) > f(X_{i,G})$$

Lecture Overview

Tree-based Pipeline Optimization tool [Olson et al. 2016]

- the very first AutoML method for hyperparameter tuning using genetic programming (GP)
- [GitHubTPOT]: is built on top of scikit-learn
- automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for tested data

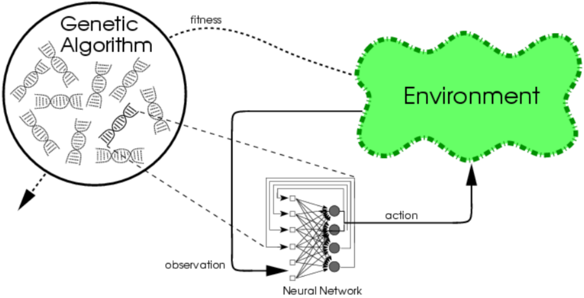


Tree-based Pipeline Optimization tool [Olson et al. 2016]

- data set flows through the pipeline operators, which add, remove, and modify the features in a successive manner
- population is a randomly generated fixed number of tree-based pipelines
- GP builds trees of mathematical functions to maximize the final classification accuracy of the pipeline
- GP is used to evolve the sequence of pipeline operators (using crossover and mutation) that acted on the data set as well as the parameters of these operators, e.g. , the number of trees in a random forest or the number of feature pairs to select during feature selection
- once evolutionary search is done, TPOT generates the best found pipeline as a Python code so you start from there

Neuroevolution [Stanley 2017]

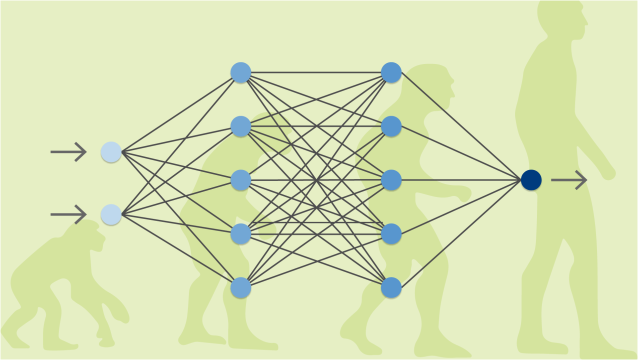
- is a ML technique that applies EAs to construct artificial neural networks (ANN), typologies and rules. i.e. tuning NNs using EAs
- is highly general; it allows learning without explicit targets, with only sparse feedback, and with arbitrary neural models and network structures
- is an effective approach to solve reinforcement learning problems, and applied in evolutionary robotics and artificial life



Neuroevolution

Different features for neuroevolution algorithms:

- conventional neuroevolution: evolve the weights for a fixed network topology
- topology and Weight Evolving Artificial Neural Network algorithms (TWEANNs): evolve both the topology of the network and its weights
- evolve the structure of ANNs in parallel to its parameters or separately



Neuroevolution vs. Gradient descent

Despite the fact that most NNs use gradient descent, neuroevolution are competitive with sophisticated modern gradient descent DL algorithms. Why?

Neuroevolution

Neuroevolution vs. Gradient descent

Despite the fact that most NNs use gradient descent, neuroevolution are competitive with sophisticated modern gradient descent DL algorithms. Why?

- because neuroevolution was found to be less likely to get stuck in local minimal. Many researchers at OpenAI and Uber proved that a simple neuroevolution algorithm is comparable/better than other methods
- neuroevolution is succeeding where it had failed before due to the increased computational power available in the 2010s
- neuroevolution methods are powerful especially in continuous domains compared to reinforcement learning

[Welcoming the Era of Deep Neuroevolution]

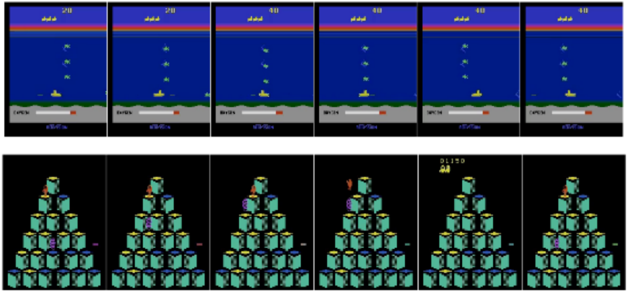
Neuroevolution

Method	Encoding	EAs	Aspects evolved
Neuro-genetic evolution 94	Direct	GA	Network weights
Cellular Encoding (CE) 94	Indirect	GP	Structure & parameters
EPNet 97	Direct	EP	Structure & parameters
NEAT 2002	Direct	GA	Structure & parameters
HyperNEAT 05/07	Direct	ES	Structure & parameters
HyperNEAT 2008	Indirect	GA	Structure & parameters
ES-HyperNEAT 2012	Indirect	GA	Structure & parameters
ICONE 2012	Direct	EA	Structure & parameters
.	.	.	.
.	.	.	.
CMA-HAGA 2017	Direct	ES	Structure & weights
Deep-NE 2019	Direct	GA	Structure & weights

[Designing neural networks through neuroevolution, Nature Machine Intelligence, 2019]

What's Else using EAs?

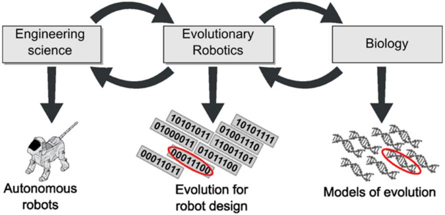
- ES to Play Atari Games/MuJoCo
  - a simple algorithm, natural ES (NES) is introduced in 2017 as a scalable alternative to RL [Salimans et al. 2017]
    - performs competitively with the best deep reinforcement learning algorithms, including deep Q-networks (DQN) and policy gradient methods (A3C)
  - canonical ES, introduced in 2018, outperformed NES to play Atari games [Chrabaszcz et al. 2018]
  - evolves networks with 1.7 million parameters

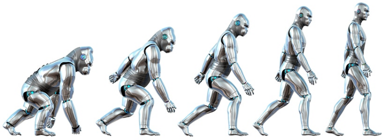


What's Else using EAs?

- Large-scale evolution of image classifiers [Real et al. 2017]
  - the use of a simple EA to discover good models for CIFAR-10 and CIFAR-100 datasets, and reach high accuracies of around 95.6% and 77.0% respectively.
- HPO of DNNs using CMA-ES [Loshchilov and Hutter 2016]
  - tuned the hyperparameters of a convolutional neural network for the MNIST dataset of 19 hyperparameters, and outperformed state-of-the-art BO methods
- Deep Neuroevolution [Such et al. 2018]
  - GAs are competitive alternative for training DNNs for RL
  - evolves DNNs with a simple GA that performs well on hard deep RL problems (Atari game)
  - GA performs as well as ES and deep RL algorithms based on DQN and A3C
  - evolves networks with over four million parameters, the largest NNs ever evolved with a traditional EA

- Evolutionary Robotics (ER)
    - research field which uses EAs to develop hardware, controllers and strategies for autonomous robots
    - population consists of candidate controllers which may be drawn from ANNs
    - goal is to use the evolutionary search to evolve population and generate the best so-found controller using mutation and crossover operations



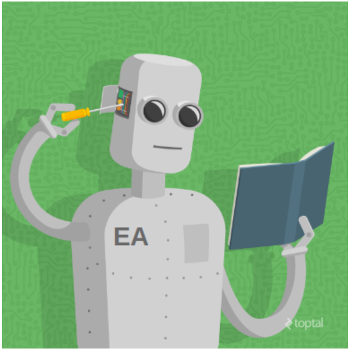


- A very useful [link] about ER

Lecture Overview

Challenges in EAs

- Classical EAs are sensitive to control hyperparameters setting
  - for some EAs, the performance is highly dependent on the settings of mutation rate, crossover rate and population size
- Many researchers tackle this problem by developing successful self-adaptive mechanisms for control parameter settings
  - adapt the settings of mutation and crossover rates using mathematical distributions such as: Cauchy, Normal, ... etc

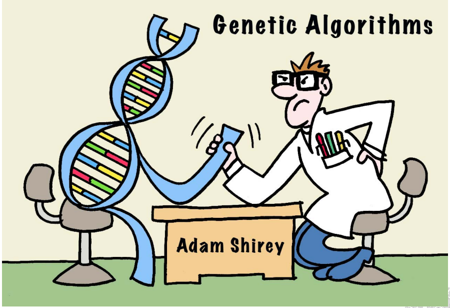


General Remark

No Free Lunch Theorem- NFL

Over a large set of problems, it is impossible to find a single best algorithm!

- for each specific problem there are suitable candidate algorithms
- you have to study and test them to know which one is the best fit for your problem
- you have to fine tune the algorithm's hyperparameters before you judge which one is the best



General Remark

Do you like evolutionary algorithms as much as I do?

Learning Goals

Now, you are able to . . .

- explain the basics of evolutionary algorithms
- discuss the different types of evolutionary algorithms
- efficiently tune HPO using evolutionary algorithms
- discuss importance of evolutionary algorithms to solve many optimization problems

Literature [These are links]

- [A Brief Review of Nature-Inspired Algorithms for Optimization. Fister et al. 2013]
- [Beyer and Schwefel. 2002. Evolution strategies – A comprehensive introduction]
- [Evolution Strategy]
- [Mitchell. 1998. An Introduction to Genetic Algorithms]
- [Storn and Price. 1995. Differential Evolution]
- [Das and Suganthan. 2011. Differential Evolution]
- [Storn and Price. 1995. Differential Evolution]
- [Olson et al. 2016. TPOT]
- [GitHubTPOT]

Literature [These are links]

- [Stanley 2017. Neuroevolution: A different kind of deep learning]
- [Salimans et al. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning]
- [Chrabaszcz et al. 2018. Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari]
- [Real et al. 2017. Large-Scale Evolution of Image Classifiers]
- [Loshchilov and Hutter 2016. CMA-ES for Hyperparameter Optimization of Deep Neural Networks]
- [Such et al. 2018. CDeep Neuroevolution]
- [Evolutionary Robotics]