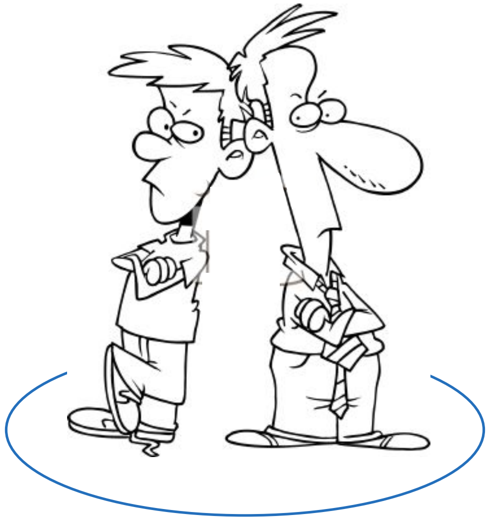
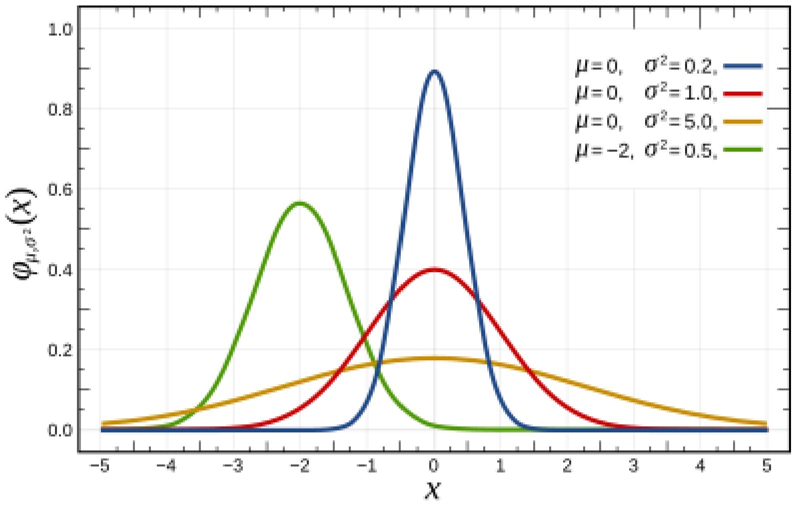
ES(1+1) Optimization Algorithm with 1/5 Rule





The red curve is the *standard normal distribution*

LTU CS

CJ Chung

Optimization/Search Algorithms

- Calculus-Based
  - Direct
    - Steepest Gradient Descent (**GD**) Algorithm …
  - Indirect
- Enumerative
- Guided random
  - Simulated Annealing
  - …
  - Evolutionary
    - GA, EP, **ES**, CA, …

GA: Genetic Algorithms
EP: Evolutionary Programming
ES: Evolution Strategies
CA: Cultural Algorithms

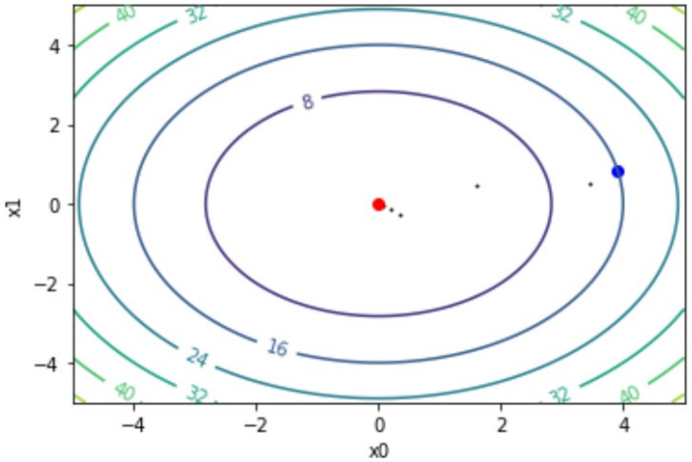
Function Optimization

- Function optimization is to find a structure having a max (or min) value of a function.
- If we think of the data structures as “points” in a space, this function can be thought of as a landscape over the space.
- When $f(x) = x$, what is the value of x which produces the minimum value of $f(x)$, where domain of x is [0, 10]? Yes. When x is 0, the value of $f(x)$ is the smallest, which is 0.

Function Optimization

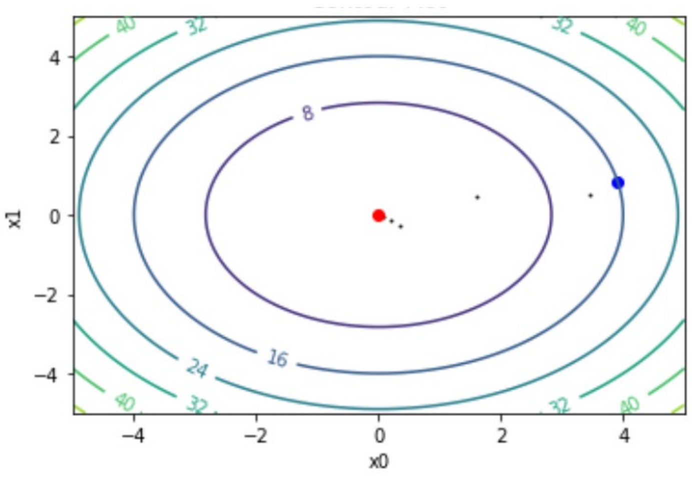
- Here is another simple problem. Now y = x^2. What makes the smallest y, when the domain range is between -2 and 2, [-2, 2]? Of course x=0 makes the smallest y, which is 0.
- For instance, let's solve the following optimization problem with 2 variables using a simple technique called Evolution Strategies (ES) inspired by the way of nature.

$$y = x_0^2 + x_1^2$$



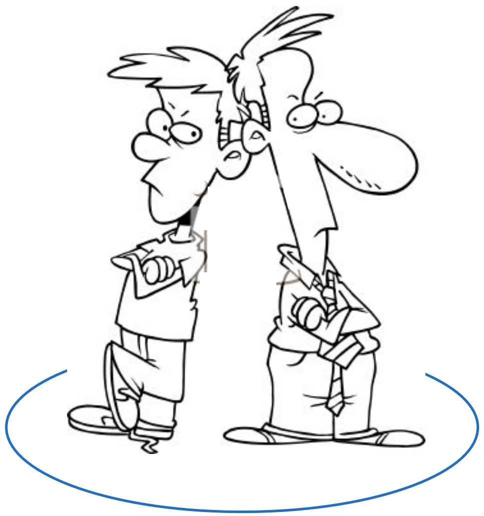
Function Optimization

- The problem is to find the value of x_0 and x_1, which produce the minimum value of $f(x_0, x_1)$
- Many mathematical methods such as Steepest Gradient Decent method have invented using calculus-based ideas. But here we just use our common sense without using calculus.
- Graphically, optimization (minimization) problem can be viewed as trying to find the lowest point in a landscape.



ES(1+1) – a parent and a child (offspring) (1/2)

- You can imagine an explorer wandering through valleys across through plains in search of topological extremes.
- We use that idea. We populate an agent, the first problem solver in the search space. This guy is evaluated in the beginning.
- This initial parent produces its child, here, ***at random***, in hoping that the child will find a better way of life.



ES(1+1) – one parent and one child (2/2)

- But, alas!, only one agent can survive in the space since the resource (food) is limited.
- Right after the birth, only one agent which is better has to be selected to continue the agent-race.
- That means either the parent or the child only can survive and become a next parent for the next generation.
- We repeat this process until we get an acceptable minimum f($\boldsymbol{x}$) value.
- This is the simplest form of Evolutionary Optimization Algorithm

Rechenberg originated the ES(1+1) idea using 2 agents in (1964) 1973

- **Question 1. How do we generate the initial agent?** Use a uniform random number generator with domain ranges
- **Question 2. How do we reproduce a child?**

Method 1 (the simplest way)

The function *N*(0, 1) will give you a Gaussian normal random number (https://en.wikipedia.org/wiki/Normal_distribution) to make a variation for each *x1* or *x2*. For example:

child's *x1* = parent's *x1* + *N*(0, 1) # variance, sigma_squared is 1

child's *x2* = parent's *x2* + *N*(0, 1)

Question 2. How do we reproduce a child? (continued)

Method 2: 1/5 success rule by Rechenberg

- What is the optimal value for the variance (sigma^2, stepsize)? Is it static or dynamic?
- Rechenberg postulated this 1/5 success rule for his Evolutionary Strategy as follows:

From time to time during the evolution process check the ratio of the number of successes to the total number of trials (variations). If the ratio is greater than 1/5, increase the variance; if it is less than 1/5, decrease the variance.

The (1+1)-ES with one-fifth success rule implements the idea that

- the step-size should <u>increase</u> if “multiple”steps are successful, indicating that the search is too local and “too small”.
- It should <u>decrease</u> if “too few” steps are successful, indicating that the step-length used for sampling solutions is “too large”.

- Balancing btw
  - Exploration vs Exploitation
  - Divergence vs Convergence

https://inria.hal.science/inria-00430515/document

Exploration vs. Exploitation

- Exploration:
  - Searching new, unvisited areas of the solution space.
  - Avoids premature convergence to local optima.
  - In ES: large step-sizes, high variance mutations.
- Exploitation:
  - Refining known good solutions.
  - Focuses search around promising regions.
  - In ES: smaller step-sizes, fine-tuning.

Balance is key:

- too much exploration = inefficient wandering
- too much exploitation = stuck in local minima

Divergence vs. Convergence (in population based search)

- **Divergence**. Multiple candidate solutions are not similar. The population members are spreading out in the search space rather than clustering around the same region.
- **Convergence**, where individuals become increasingly similar (loss of diversity). Premature convergence leads to local optima

Why the balancing is important?

- The idea of maintaining tension between two opposing forces is a unifying principle in metaheuristics.
- In practice, good optimizers oscillate between divergence/exploration phases and convergence/exploitation phases.

Implementation of the 1/5 rule

- A variable called “Success_Counter” to count number of mutation(variation) successes.
- Introduce the current “stepsize” variable
- Initialize the “Success_Counter” as 0 and the “stepsize” as 0.82
- Increase the Success_Counter, when the child is better than parent and the child become a parent for the next generation.

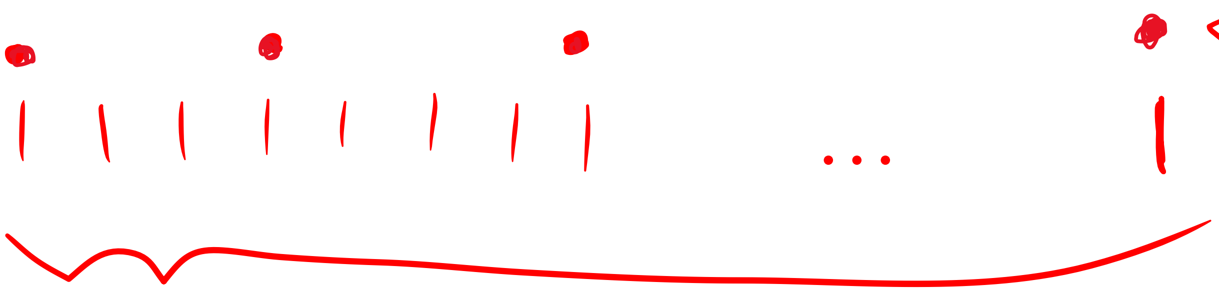
Why 0.82? Hans-Paul Schwefel, a PhD student of Rechenberg used the number.

Implementation of 1/5 (0.2) rule

- stepSize update rule for every WindowSize generations

```
IF success counter > WindowSize*0.2          // Why 1/5 (20%)?
    stepsize = stepsize / 0.82               // increase. Why 0.82?
Else If success counter < WindowSize*0.2
    stepsize = stepsize * 0.82               // decrease. Why 0.82?
Success counter is reset to zero.
```

- For example, when the WindowSize is 50, the above routine is called once every 50 generations to adjust the stepsize



success counter

50 generations

$4/50 = 8\%$

If success ratio > 20%

  increase stepsize

Else if success ratio < 20%

  decrease stepsize

Else

  keep the current stepsize

Reset the success counter

Implementation of 1/5 rule

- **Question 3. How do we select the next parent?**

  Just compare the evaluated value between parent and child. If the child is better than the parent, the child becomes the parent for the next generation.

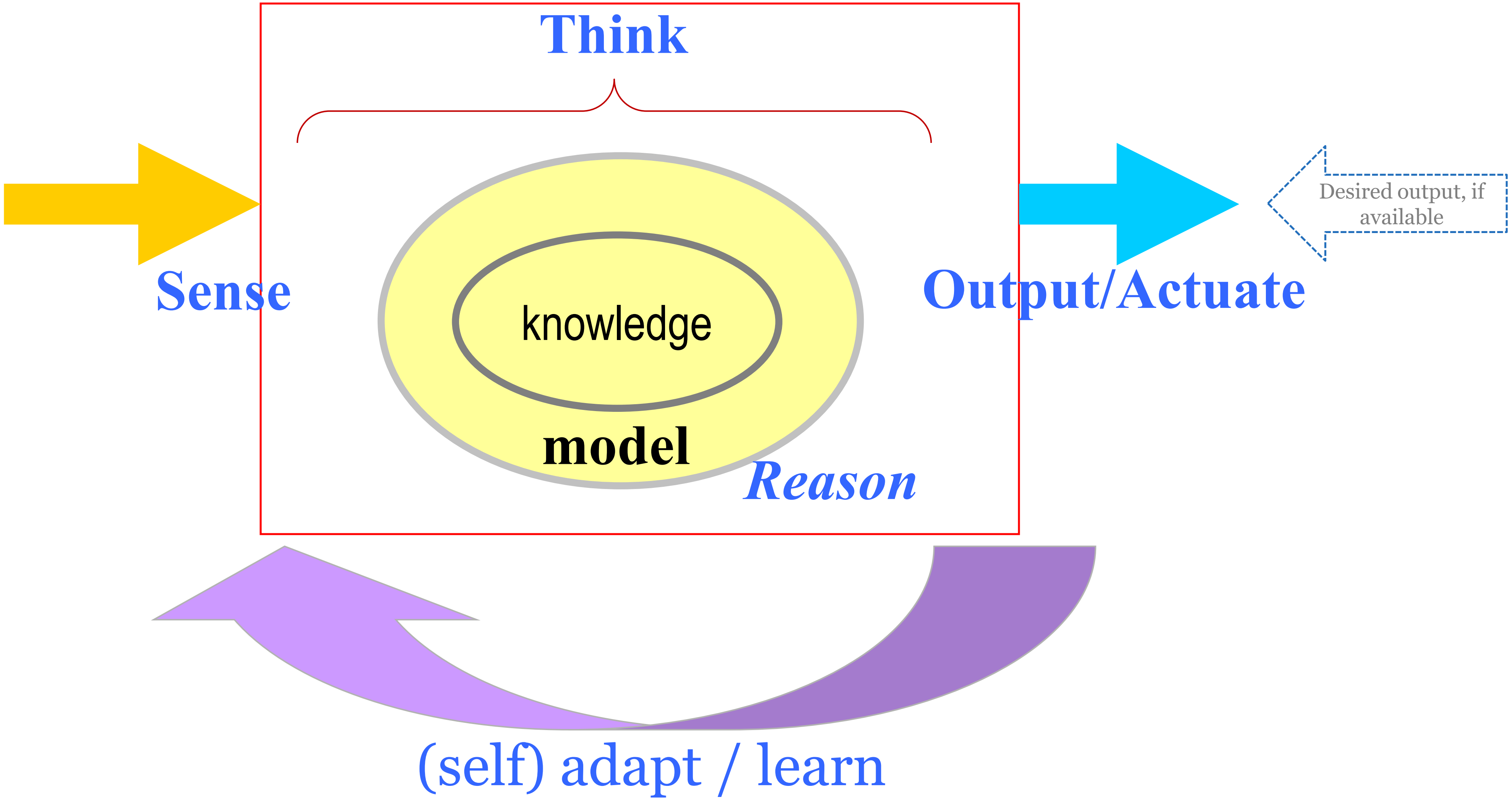
- **Question 4. What's next?**

  Keep on repeating the above processes until an acceptable child is produced.

ES(1+1) with 1/5 success rule

```
Initialize xp
stepSize = 0.82; successCnt = 0; WindowSize = 30
p_val = objfunc(xp)
for g in range(1, MaxGen+1):
    if (g % WindowSize) == 0: # update stepsize
       if successCnt > (WindowSize * 0.2):
         stepSize = stepSize / 0.82         #increase
       elif successCnt < (WindowSize * 0.2):
         stepSize = stepSize * 0.82         #decrease
       successCnt = 0
    for j in range(0, numVar): # mutate parent to generate an offspring
       xo[j] = xp[j] + np.random.normal(0.0, stepSize)  # mu and variance
    o_val = objfunc(xo)  # evaluate offspring
    if o_val < p_val: # if offspring is better, it becomes a parent
       xp = xo.copy()
       p_val = o_val
       successCnt += 1;
    if p_val < minima+EPSILON:
       return xp, p_val
```

ES(1+1) with 1/5 rule: An example of knowledge based self-adaptation



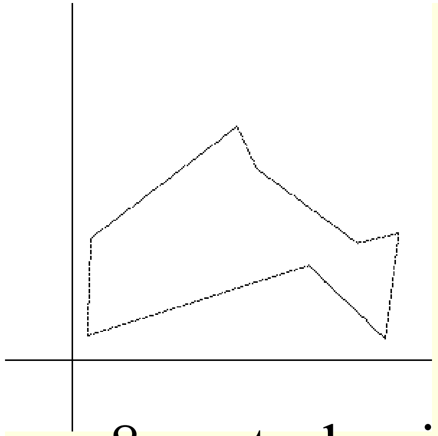
ES(1+1) with 1/5 rule is simple but powerful. Research topics for improvements

- Plus Strategy: ES(*n*+*n*)
- Comma Strategy: ES(*n*,m) – *n* parents must die
- Why 1/5 (20%)? Adapt the rule dynamically as time goes - ES(*n*+*m*) with 1/z rule, n, m, z are adaptive (dynamic). Find the best n, m, and z for a specific group of problems.
- Why step size change ratio is 0.82? Make it also adaptive
- Why just one step size for all variables? Maintain step size for each variable
- …

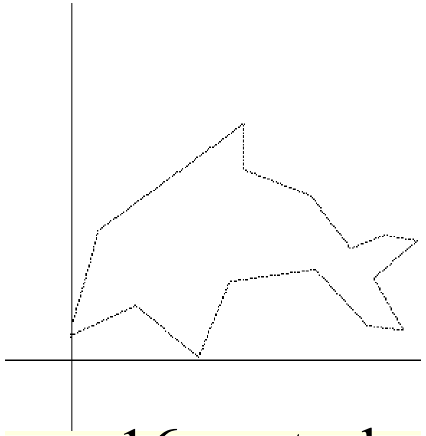
ES(1+1) with 1/5 rule is simple but powerful. Research Papers Published directly

- Evolutionary Hyperparameter Optimization to Find Lightweight CNN Models for Autonomous Steering https://ieeexplore.ieee.org/document/11103679
- Optimizing Hyperparameters for Deep Learning Models Using Evolutionary Algorithms: Solving the Four-Class Intertwined Spiral Classification Problem. https://ieeexplore.ieee.org/document/11103665

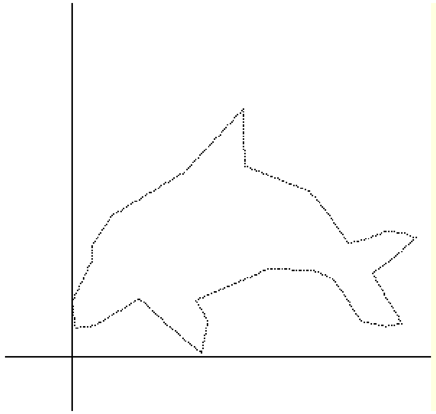
Application: IEEE CEC 2D Target Shape Optimization Competition 1st Place – to guess a secret 2D shape in a black box



8 control points

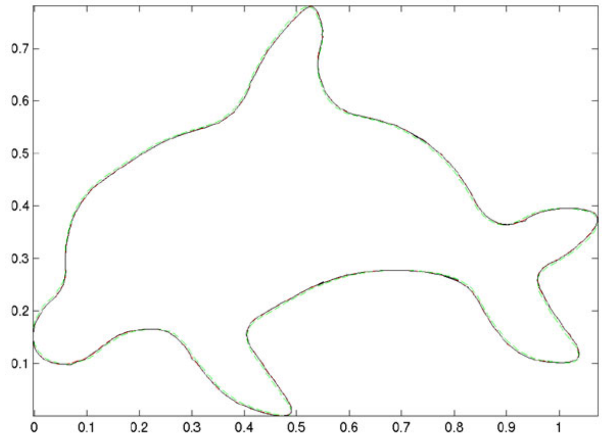


16 control points

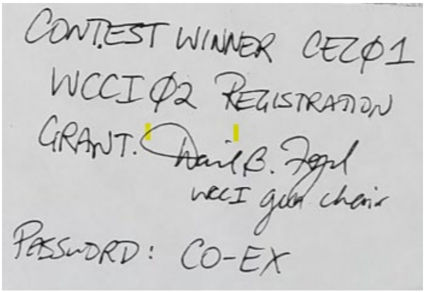


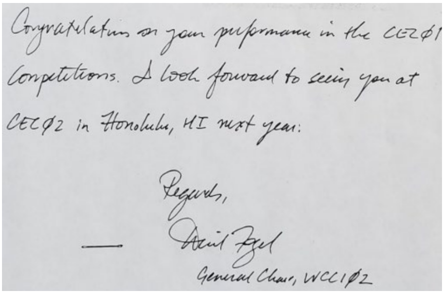
32 control points

…



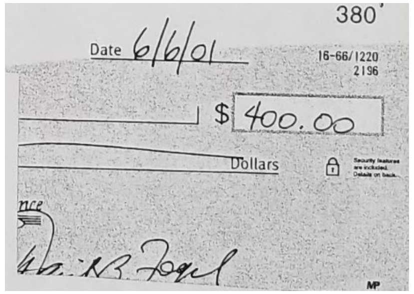
IEEE 2001 Congress on Evolutionary Computation


CONTEST WINNER CECØ1
WCCIØ2 REGISTRATION
GRANT: David B. Fogel
WCCI gen chair
PASSWORD: CO-EX

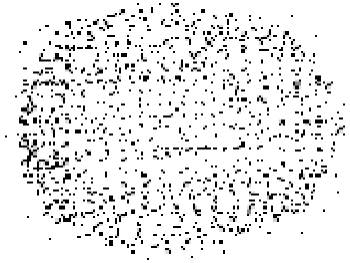

Congratulations on your performance in the CECØ1 Competitions. I look forward to seeing you at CECØ2 in Honolulu, HI next year.

Regards,

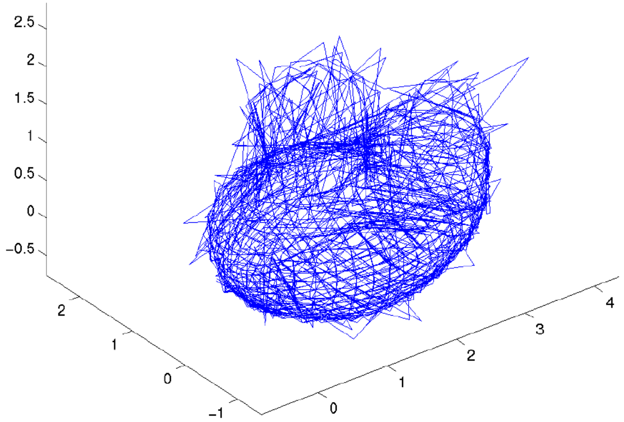
David Fogel
General Chair, WCCIØ2

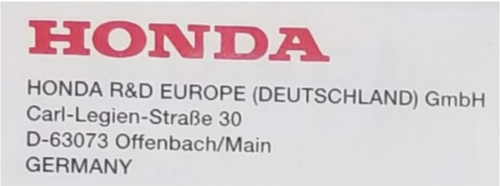


Application: IEEE 2002 CEC (Congress on Evolutionary Computation), 3D Shape Optimization, 1st Place

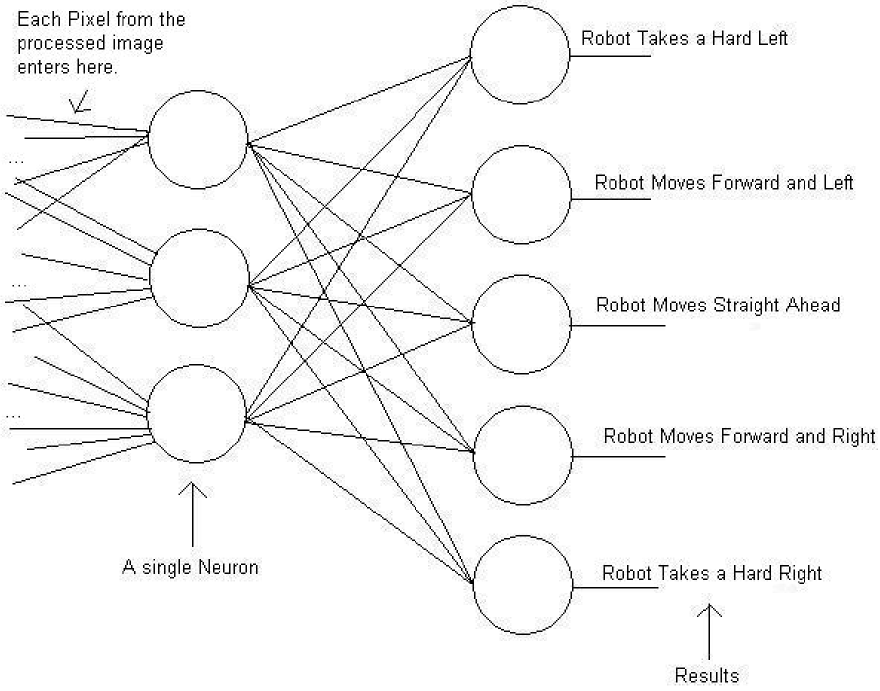


…





LTU IGVC team 2003, training weights of a NN



No hope, restart the evolution!

Stock Investment Strategy using 1/5 rule idea

- You are investing 3% of your salary in the stock market
- During the past 4 months (Window size)

  If you made money on average

  - It is time to increase 3% to 3.65% (3/0.82)

  Else # lost money

  - It is time to decrease 3% to 2.46% (3*0.82)

To scale the step-size relative to the range of a parameter

1. **Define the range of the parameter**: Let's say the parameter x you are optimizing has a range $[x_{\text{min}}, x_{\text{max}}]$.
2. **Relative step size**: You want the step size σ to be meaningful in the context of the parameter's range. A simple way to do this is by expressing σ as a fraction of the parameter range:

   $$\sigma_{\text{scaled}} = \sigma \times (x_{\text{max}} - x_{\text{min}})$$

   This ensures that the step size remains proportional to the parameter's allowed values.
3. **Update rule (1/5th success rule)**: After generating a new candidate solution, observe whether the mutation was successful (i.e., if it leads to a better solution). If the success rate is higher than 1/5, increase the step size; otherwise, decrease it:
   - If the success rate $s > \frac{1}{5}$, increase σ_{scaled} by a constant factor (e.g., 1.5).
   - If $s < \frac{1}{5}$, decrease σ_{scaled} by a constant factor (e.g., 0.85).

   Mathematically:

   $$\sigma_{\text{scaled}} = \begin{cases} \sigma_{\text{scaled}} \times c_{\text{up}} & \text{if success rate} > \frac{1}{5} \\ \sigma_{\text{scaled}} \times c_{\text{down}} & \text{if success rate} < \frac{1}{5} \end{cases}$$

   where c_{up} and c_{down} are typically around 1.5 and 0.85, respectively.

Optimizing NN Hyper Params Using ES(1+1) with 1/5 rule

Search & Optimization Strategy - Balancing between

? vs Exploitation

? vs Convergence

ES(1+1) with 1/5 success rule

```
Initialize xp
stepSize = 0.82; successCnt = 0; WindowSize = 30
p_val = objfunc(xp)
for g in range(1, MaxGen+1):
    if (g % WindowSize) == 0: # update stepsize
       if successCnt > (WindowSize * 0.2):
         stepSize = stepSize / 0.82         #increase
       elif successCnt < (WindowSize * 0.2):
         stepSize = stepSize * 0.82         #decrease
       successCnt = 0
    for j in range(0, numVar): # mutate parent to generate an offspring
       xo[j] = xp[j] + np.random.normal(0.0,     ?    )  # mu and variance
    o_val = objfunc(xo)  # evaluate offspring
    if o_val < p_val: # if offspring is better, it becomes a parent
       xp = xo.copy()
       p_val = o_val
       successCnt += 1;
    if p_val < minima+EPSILON:
       return xp, p_val
```

ES(1+1) with 1/5 success rule

What's wrong?

```
Initialize xp
stepSize = 0.82; successCnt = 0; WindowSize = 30
p_val = objfunc(xp)
for g in range(1, MaxGen+1):
    if (g % WindowSize) == 0: # update stepsize
        if successCnt > (WindowSize * 0.2):
          stepSize = stepSize * 0.82         #increase
        elif successCnt < (WindowSize * 0.2):
          stepSize = stepSize / 0.82         #decrease
        successCnt = 0
    for j in range(0, numVar): # mutate parent to generate an offspring
        xo[j] = xp[j] + np.random.normal(0.0, stepSize)  # mu and variance
    o_val = objfunc(xo)  # evaluate offspring
    if o_val < p_val: # if offspring is better, it becomes a parent
        xp = xo.copy()
        p_val = o_val
        successCnt += 1;
    if p_val < minima+EPSILON:
        return xp, p_val
```