Evolutionary Algorithms

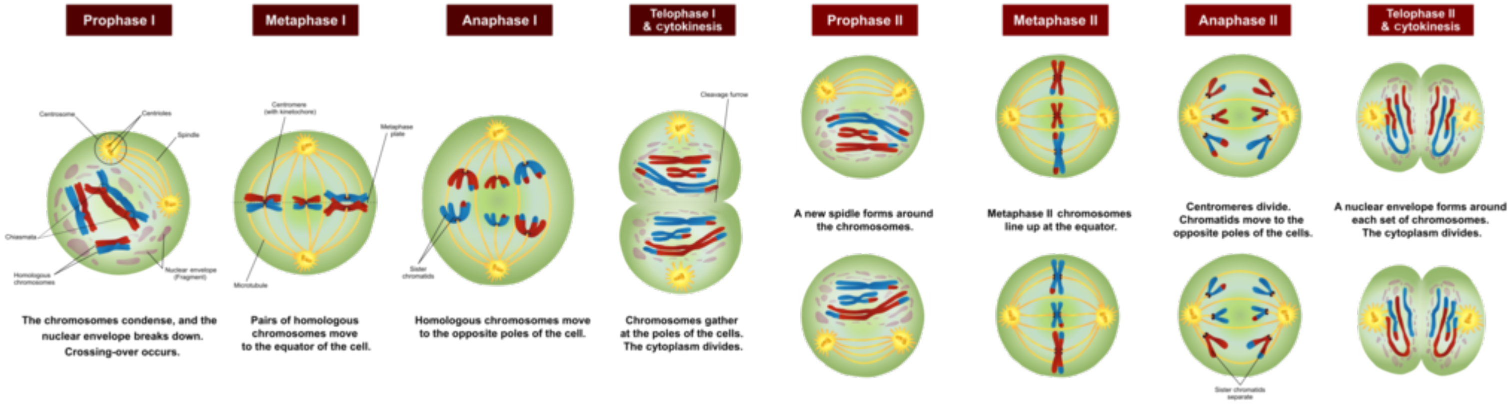
COSC 410: Applied Machine Learning

Spring 2022

Prof. Apthorpe

Outline

- Definition
- Candidate Representation
- Fitness
- Selection
- Genetic Operators
- Examples
- Important Limitations



Definition

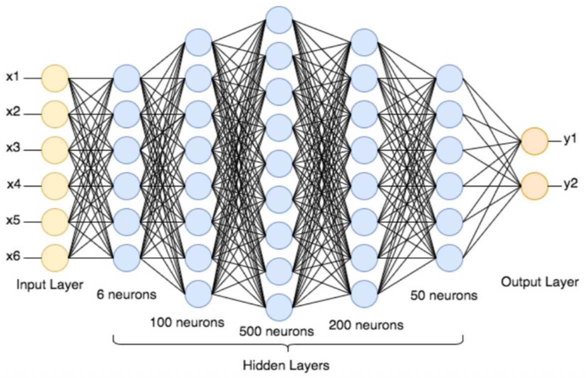
- **Heuristic optimization** algorithm inspired by **biological evolution**
- **Candidate solutions** act as **individuals** in a **population**
- Better (more **“fit”**) solutions **recombine** into successive generations
- **Repeat** until “good enough” solution is found

Candidate Representation

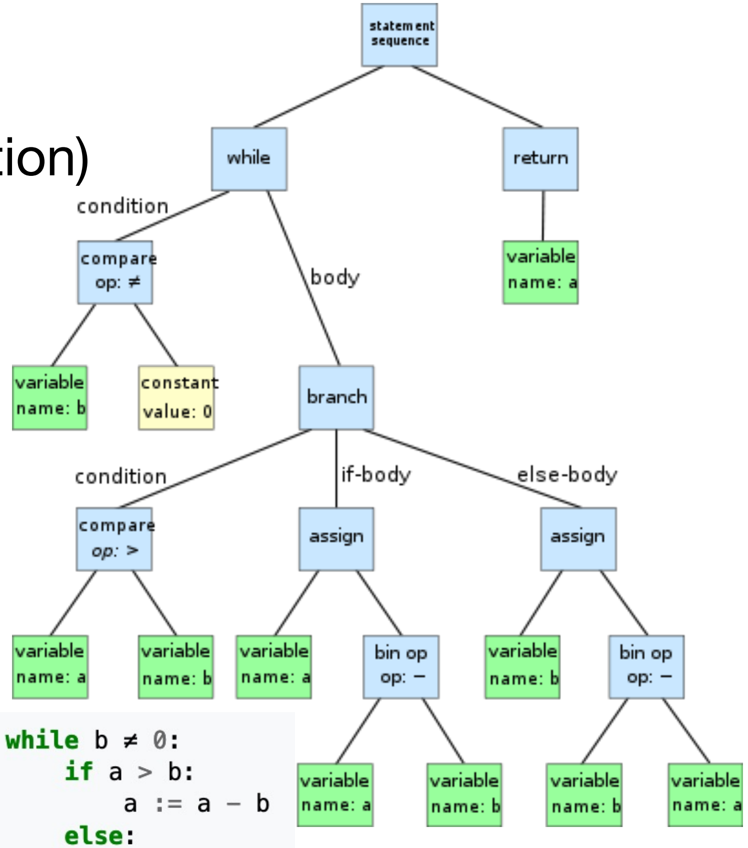
- **Direct encoding**
  - **Genotype** (representation) maps directly to **phenotype** (solution)
- **Indirect encoding**
  - Genotype specifies how phenotype should be generated

Traditional **Genetic Algorithms**

1	1	0	0	1	0	1	Binary
5	4	9	0	1	3	3	Numerical
R	D	e	f	A	A	z	String



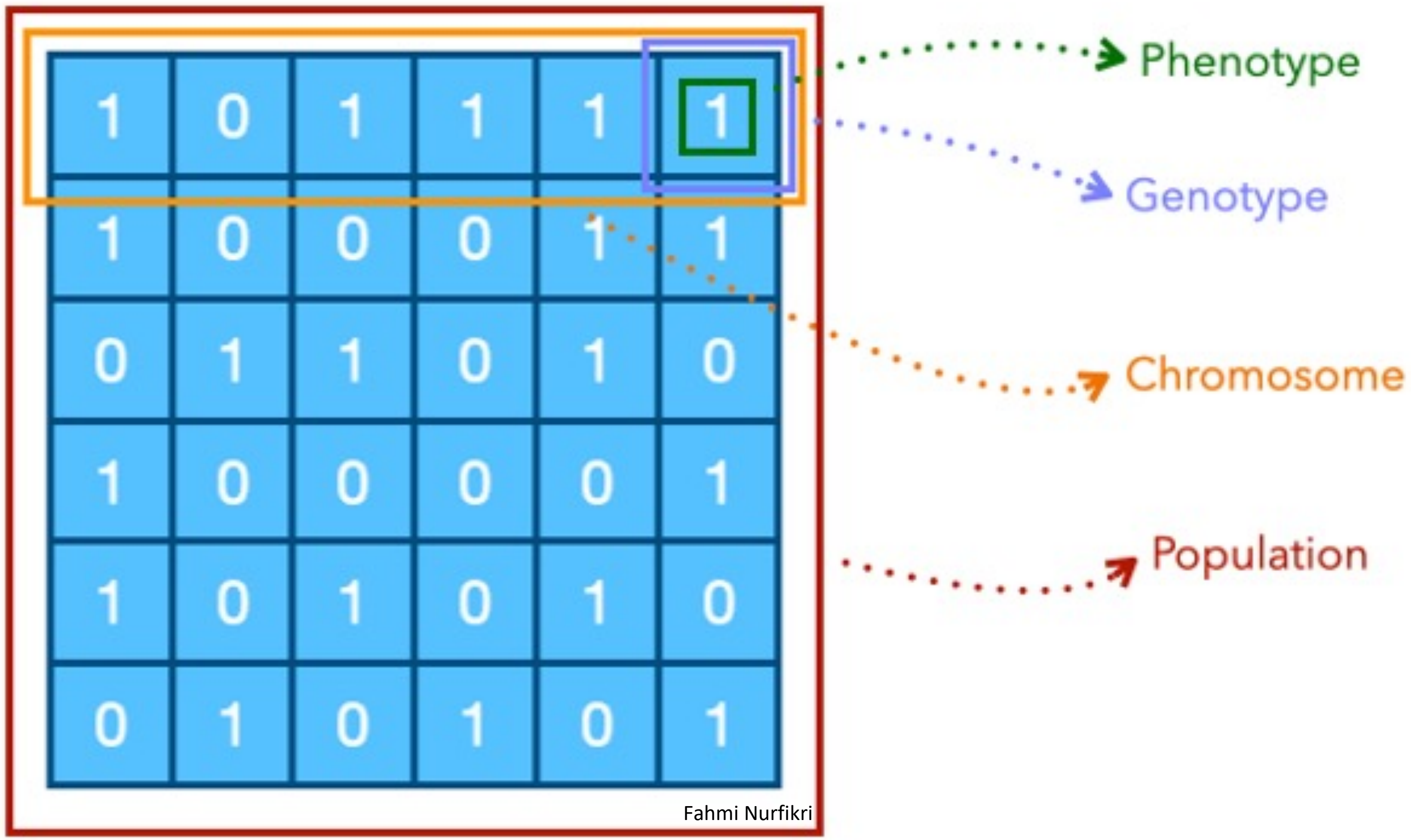
Neural Network Architecture




```
while b ≠ 0:
    if a > b:
        a := a − b
    else:
        b := b − a
return a
```

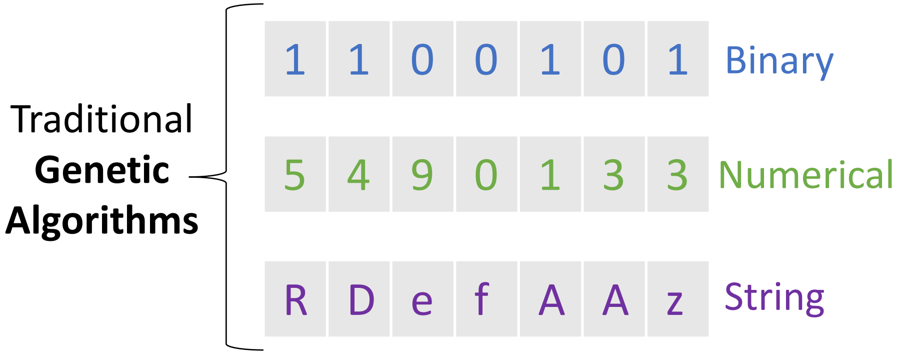
Abstract Syntax Tree

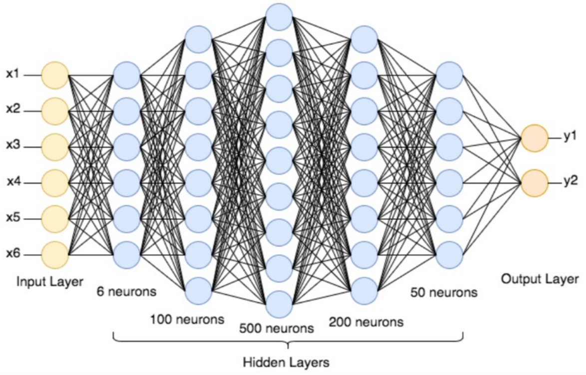
Candidate Representation



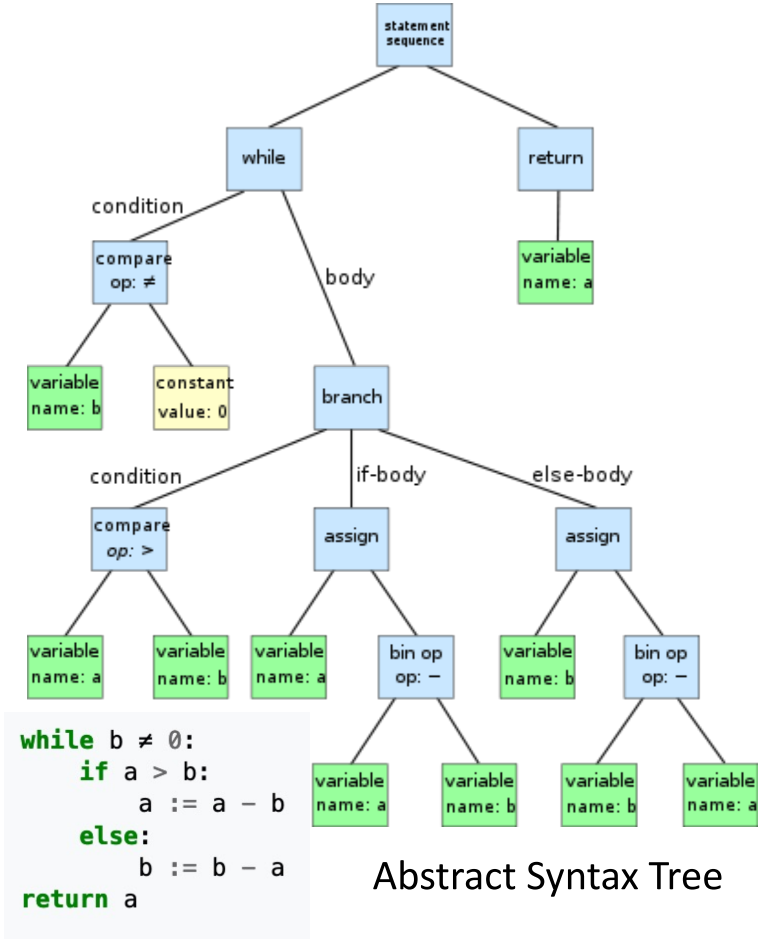
Candidate Representation

- Genotype structure matters!
  - Should reflect important elements of the phenotype
  - Simplifies genetic operators (esp. crossover)





Neural Network Architecture



Abstract Syntax Tree

Fitness

- **Fitness function** measures the overall success of candidates
  - "How well does the candidate solve the problem?"
  - What the evolutionary algorithm tries to maximize (or minimize)
  - Should be fast to compute (why?)

https://rednuht.org/genetic_cars_2/

Selection

- Choose which candidate solutions contribute to next generation
- **Fitness proportionate selection**
  - Random draw of candidates weighted by fitness
- **Tournament selection**
  - Draw N uniformly random candidates→ Keep candidate with highest fitness → repeat

Genetic Operators

How would you design a mutation operation for a neural network architecture or other graph?

- Introduce variations into successive generations

- **Mutation**
  - Randomized variation of single candidate

1 0 1 0 0 1 0

↓

1 0 1 0 1 1 0

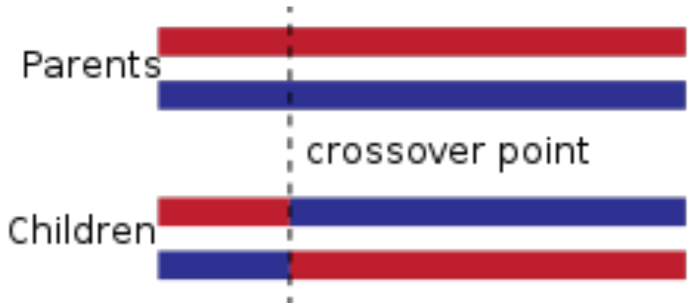
Genetic Operators

How would you design a crossover operation for a neural network architecture or other graph?

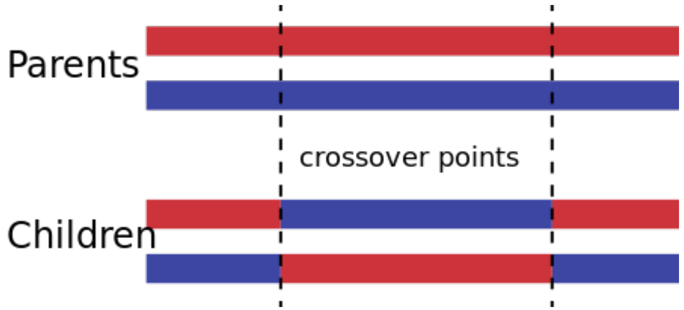
- Introduce variations into successive generations

- **Crossover**
  - Combination of two (or more) candidate into new candidate

One point crossover



Two point crossover



k-point crossover…

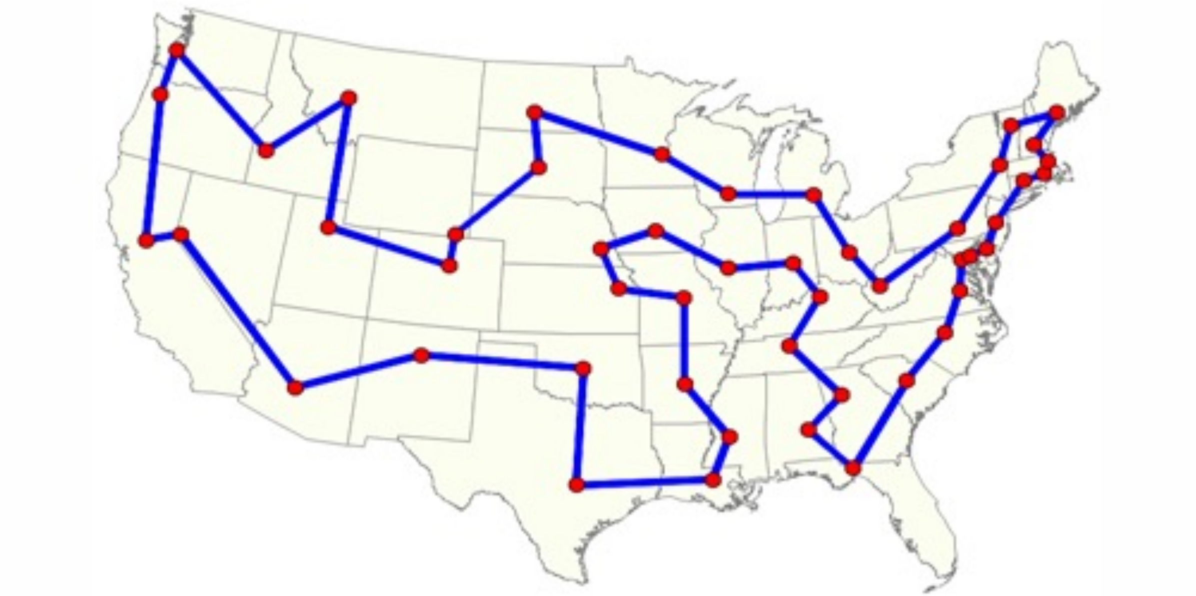
Genetic Operators

- Introduce variations into successive generations

- **Elitism**
    - Best N candidates continue to next generation **unchanged**
    - Prevents best solutions from getting lost in successive generations

Example: Traveling Salesperson Problem

- Given a list of **cities** and the **distances between each pair of cities**
- What is the **shortest possible route** that **visits each city exactly once** and **returns to the origin city**?
- Problem is **NP-hard**



Example: Traveling Salesperson Problem

1. Represent **candidate solutions** as **ordered lists** of cities
    - Initialize starting population of N candidates randomly (such that each candidate includes all cities)

[1,2,3,4,5,6,1]   [6,1,3,4,2,5,6]   [2,1,3,4,6,5,2]

[2,5,6,1,4,3,2]   [4,1,2,6,5,3,4]   etc.

2. Calculate **fitness** of each solution

↑ Shorter path   ↓ Longer path

Example: Traveling Salesperson Problem

3. Select candidates to contribute to next generation based on fitness

Fitness + [1,2,3,4,5,6,1]

[2,1,3,4,6,5,2] Fitness +++

Fitness +++ [2,1,3,4,6,5,2]

[4,1,2,6,5,3,4] Fitness +

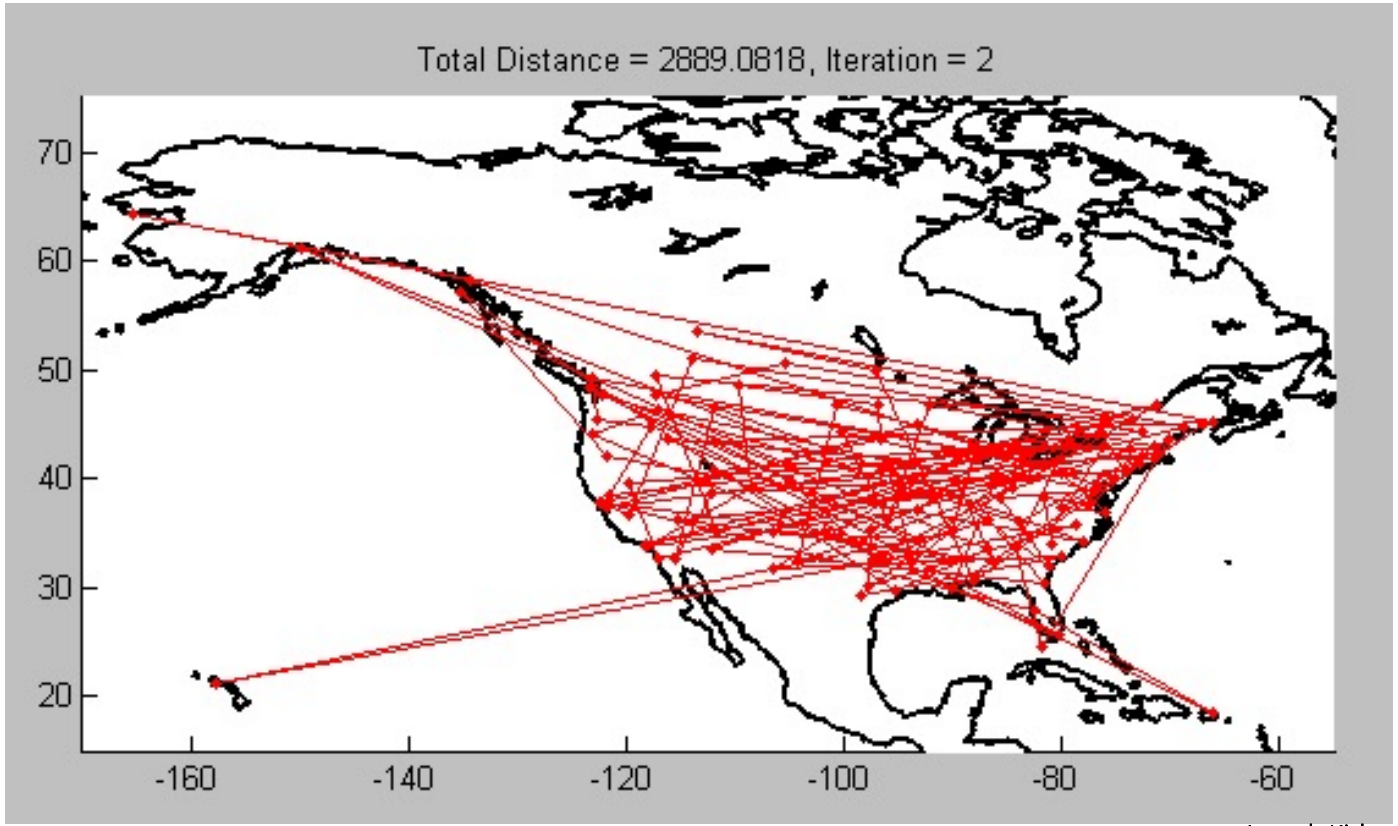
4. Produce successive generation via genetic operators (crossover & mutation)

[1,2,3,4,5,6,1]
[2,1,3,4,6,5,2]
→ [1,2,3,4,6,5,1]

[2,1,3,4,6,5,2]
[4,1,2,6,5,3,4]
→ [4,1,3,6,5,2,4]

Example: Traveling Salesperson Problem

5. Repeat until a good solution is found!



Joseph Kirk

https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html

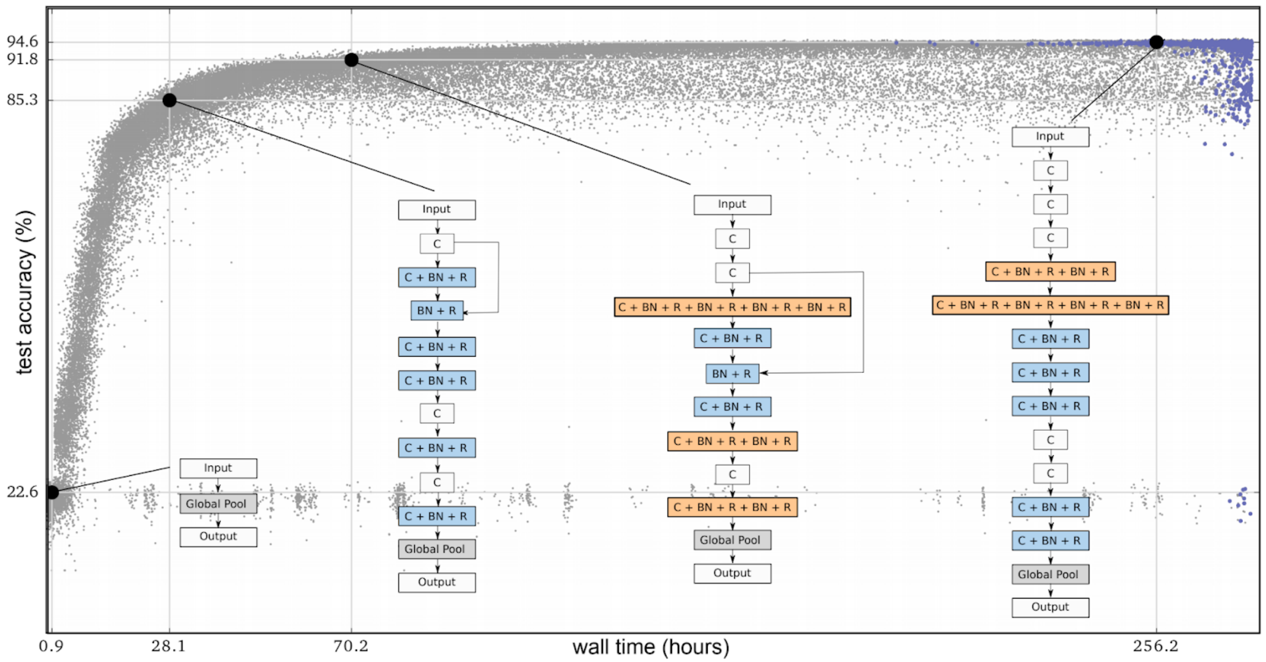
Example: Neural Network Architectures

A Simple Approach

The following is an example of an experiment from our first paper. In the figure below, each dot is a neural network trained on the CIFAR-10 dataset, which is commonly used to train image classifiers. Initially, the population consists of one thousand identical simple *seed* models (no hidden layers). Starting from simple seed models is important — if we had started from a high-quality model with initial conditions containing expert knowledge, it would have been easier to get a high-quality model in the end. Once seeded with the simple models, the process advances in steps. At each step, a pair of neural networks is chosen at random. The network with higher accuracy is selected as a *parent* and is copied and mutated to generate a *child* that is then added to the population, while the other neural network *dies* out. All other networks remain unchanged during the step. With the application of many such steps in succession, the population evolves.

The mutations in our first paper are purposefully simple: remove a convolution at random, add a skip connection between arbitrary layers, or change the learning rate, to name a few.

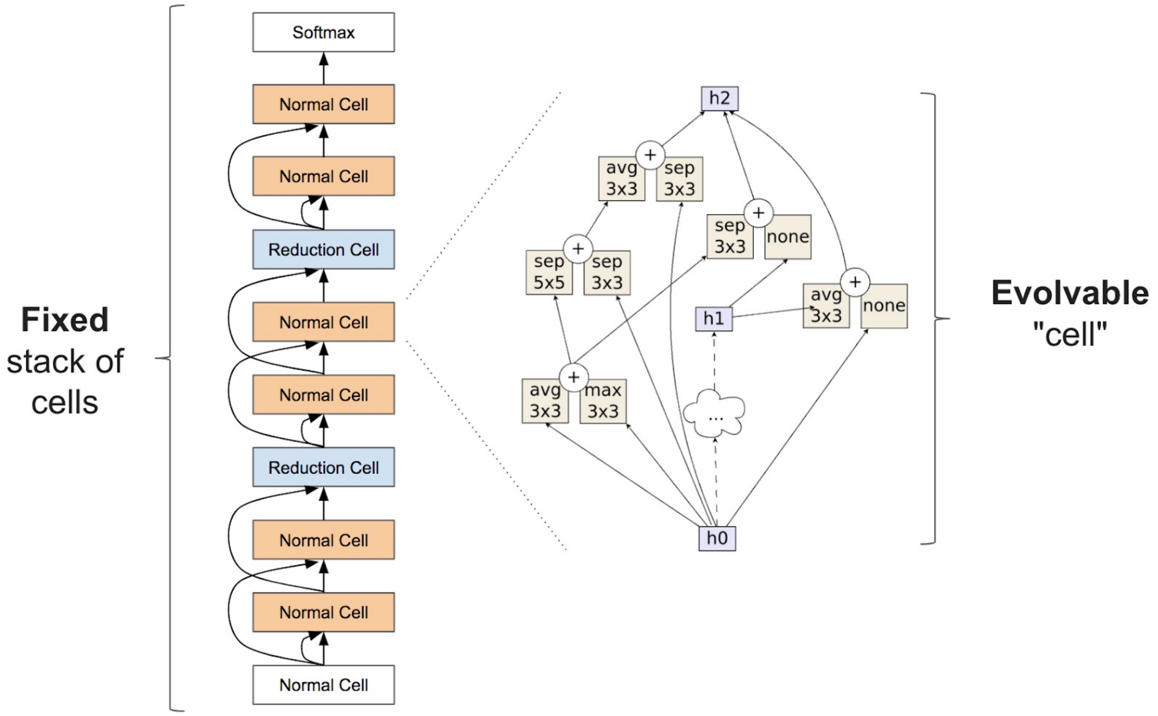
In this paper, the networks can also inherit their parent's weights. Thus, in addition to evolving the architecture, the population trains its networks while exploring the search space of initial conditions and learning-rate schedules. As a result, the process yields fully trained models with optimized hyperparameters. No expert input is needed after the experiment starts.



https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html

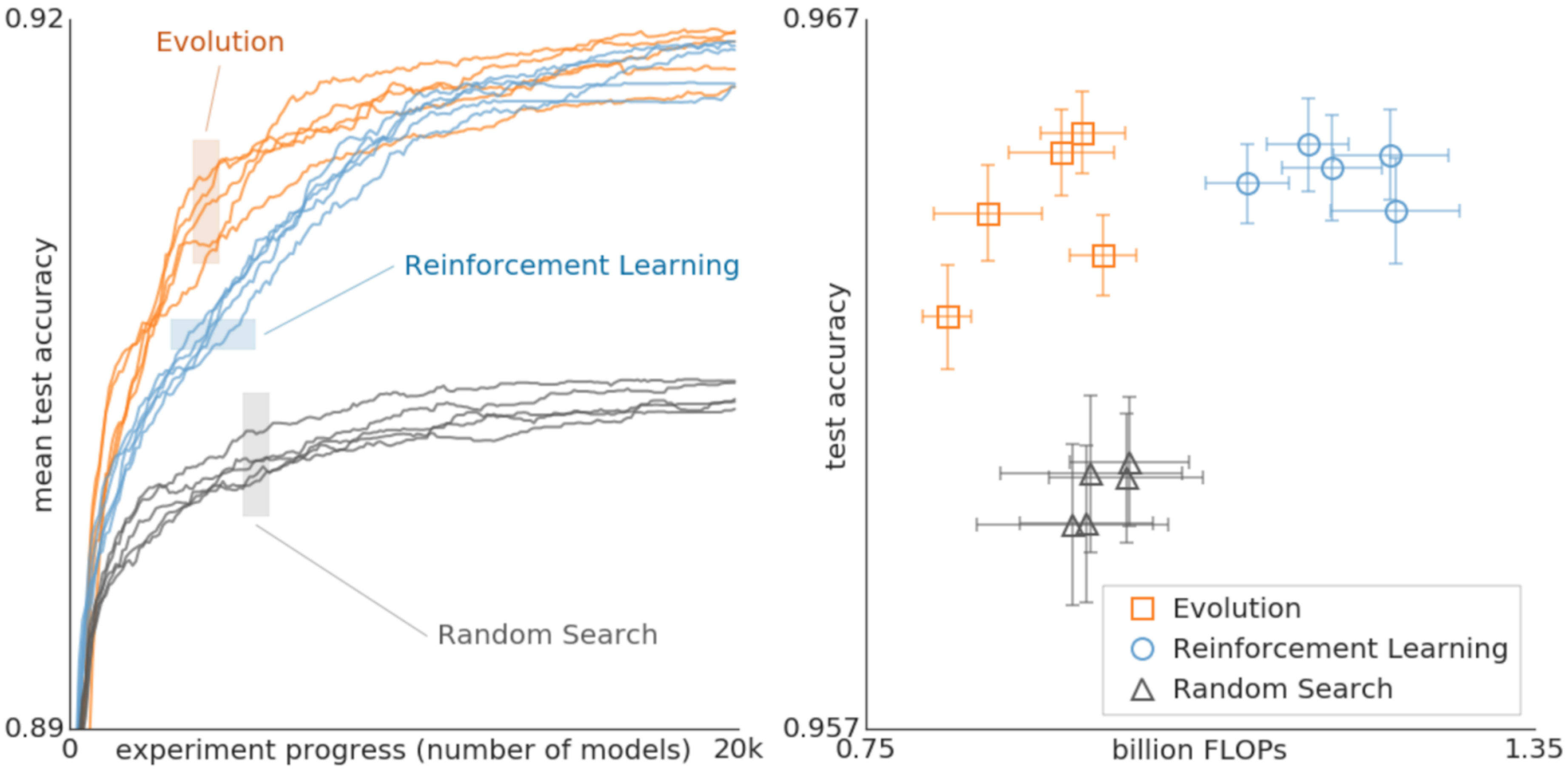
Example: Neural Network Architectures

In our second paper, "Regularized Evolution for Image Classifier Architecture Search" (2018), we presented the results of applying evolutionary algorithms to the search space described above. The mutations modify the cell by randomly reconnecting the inputs (the arrows on the right diagram in the figure) or randomly replacing the operations (for example, they can replace the "max 3x3" in the figure, a max-pool operation, with an arbitrary alternative). These mutations are still relatively simple, but the initial conditions are not: the population is now initialized with models that must conform to the outer stack of cells, which was designed by an expert. Even though the cells in these seed models are random, we are no longer starting from simple models, which makes it easier to get to high-quality models in the end. If the evolutionary algorithm is contributing meaningfully, the final networks should be significantly better than the networks we already know can be constructed within this search space. Our paper shows that evolution can indeed find state-of-the-art models that either match or outperform hand-designs.



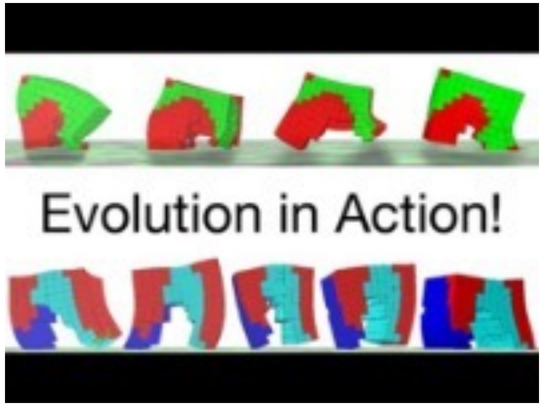
https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html

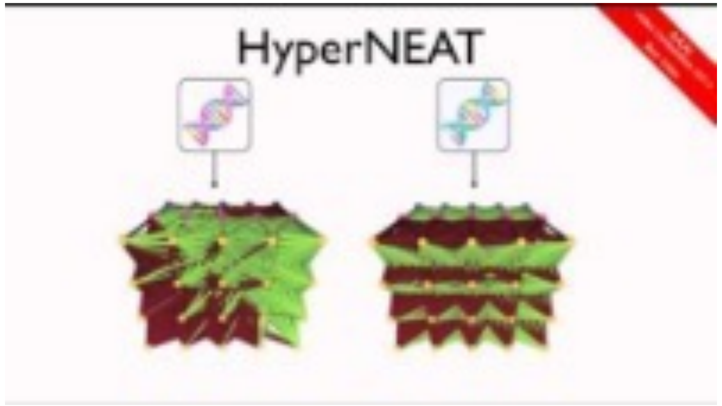
Example: Neural Network Architectures



Comparison between evolution, reinforcement learning, and random search for the purposes of architecture search. These experiments were done on the CIFAR-10 dataset, under the same conditions as Zoph et al. (2017), where the search space was originally used with reinforcement learning.

Video & Interactive Examples





https://math.hws.edu/eck/js/genetic-algorithm/GA.html

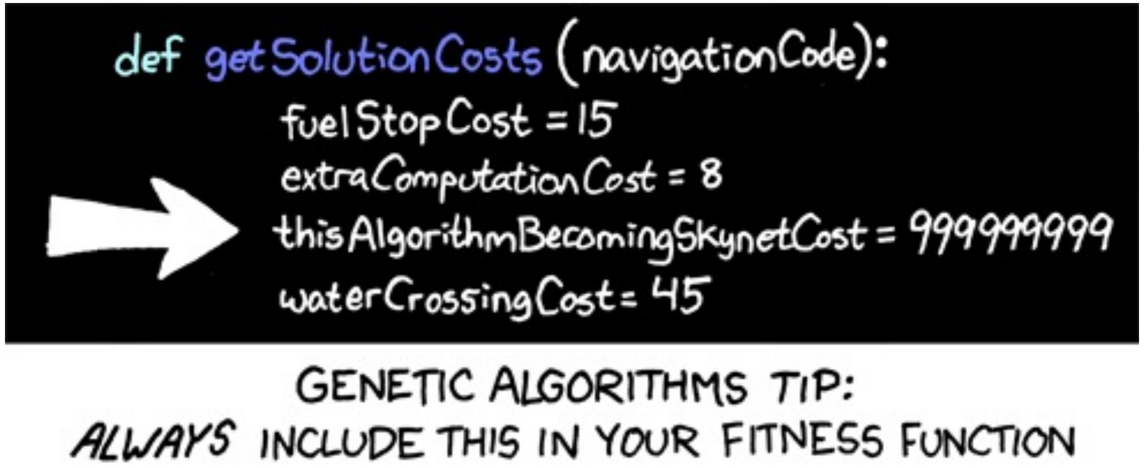
https://rednuht.org/genetic_cars_2/

https://rednuht.org/genetic_walkers/

Many other examples are available from a wide range of research in evolutionary computing

Important Limitations

- Computationally inefficient if fitness is slow to evaluate
  - When could this happen?
- Likelihood of success decreases with problem complexity
  - Why?
- Easily trapped in local optima
  - Why?
- Specific problems usually have better specific algorithms
  - Examples?
- More…



https://xkcd.com/534/

Questions?