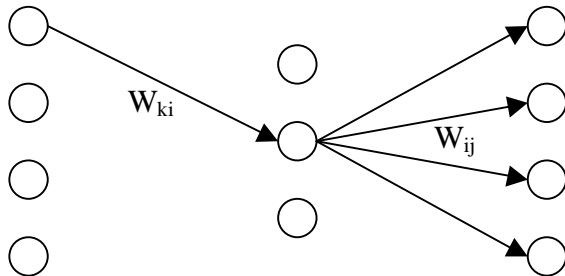


Multilayer Neural Networks



We calculate the change between the hidden layer and the output layer using the **Delta Rule**:

$$\Delta W_{ij} = \eta w_{ij} (t_j - O_j)$$

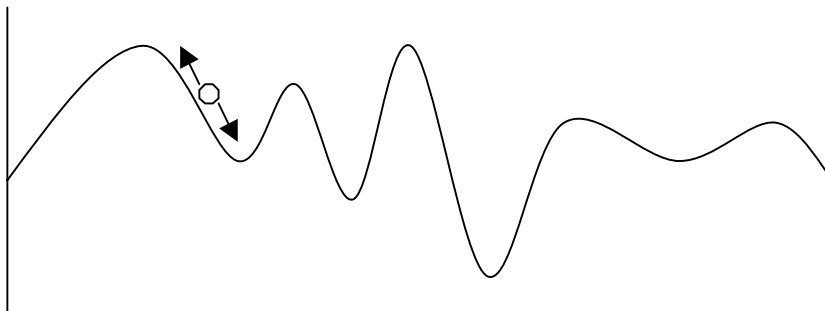
In order to calculate the change of weight between the input layer and the hidden layer, we use **Backpropagation**:

- Corrective feedback is propagated backwards layer by layer
- Also called **Generalized Delta Rule**
- Sum up the changes between the hidden layer and output layer

$$\Delta W_{ki} = \eta W_{ki} \sum_{j=1,m} \Delta W_{ij} \quad \text{where } m = \# \text{ of output nodes}$$

Hill Climbing

- Gradient descent method
- Greedy algorithm
- Not guaranteed to find a global minimum
- On an energy curve, take the step that brings you lower



Simulated Annealing

What is it?

- Algorithm to introduce noise to help get out (“jump out”) of **local minima**
- We calculate the probability of “jumping” as follows:

$$P = e^{-\Delta E/T} \quad \text{where } T = \text{temperature, } E = \text{energy}$$

- We want to lower the **temperature** until the network is not caught in a local minima
- First we use **hill climbing** to get to a local minima
- Once there, we pick a random point on the energy curve. The difference between that point and the point at which our network is at is the ΔE
- Initially we keep the temperature high to increase the size of jumps, because it is unlikely that we are close to the **global minimum**.
- We gradually decrease the temperature, and therefore the size of jumps, as time passes because we are likely getting closer to the global minimum and don't want to run the risk of jumping too far.
- Has been used for integrated chip layout
- Good for problems in which the **search space** is large
- Good for problems in which there is no need for knowledge and there is a well-known **evaluation function**.

Problems and limitations of Simulated Annealing

- The schedule of lowering the temperature is still a ‘black art’.
- There is **no memory** in simulated annealing. The network does not know what parts of the curve it has already visited.
- If the global minimum is, by chance, found early on, it's likely that we will “jump out” of it.
- **Ridge phenomenon** – in a multidimensional space, the individual may move along a “ridge” on which energy is neither increasing or decreasing.

Genetic Algorithms

- Start with a whole species of **individuals**, and allow them to **cross over** and **mutate**.
- **Genetic Programming** – a specialized instance of Genetic Algorithms (GA) applied to computer programs.
- Example:

Boundary Following Robot

Software engineering method: Analyze the problem and build a program.

Genetic programming method: start with 5000 “lousy” programs and use them to generate better programs. The optimal program may or may not be in the initial 5000.

Generation 1:

Population: 5000

Simple Evaluation Function:

- Let each program run for 60 steps
- Count the number of boundary cells covered
- Repeat 10 times
- A perfect program will cover a total of 320 boundary cells

Evaluating every program requires (60 * 10 * 5000) steps. Time intensive!

Generation 2:

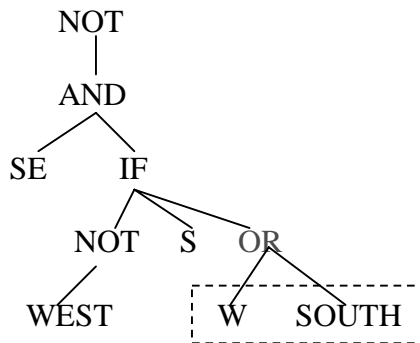
α = % of parent population carried over from Gen 1(eg, 10%)

β = % of children produced by **crossover** of random members of Gen. 1

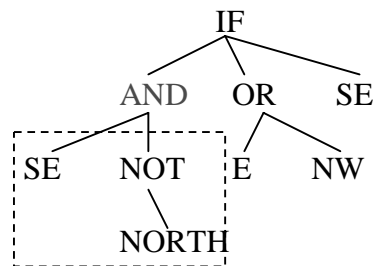
γ = % of children produced through **mutation** (eg, 1%)

Crossover

- Randomly select one node in mother program and one node in father program.

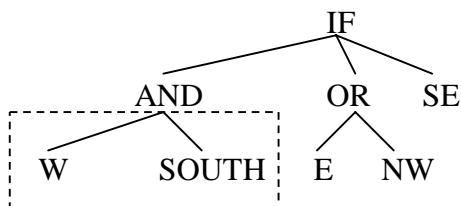


Mother Program



Father Program

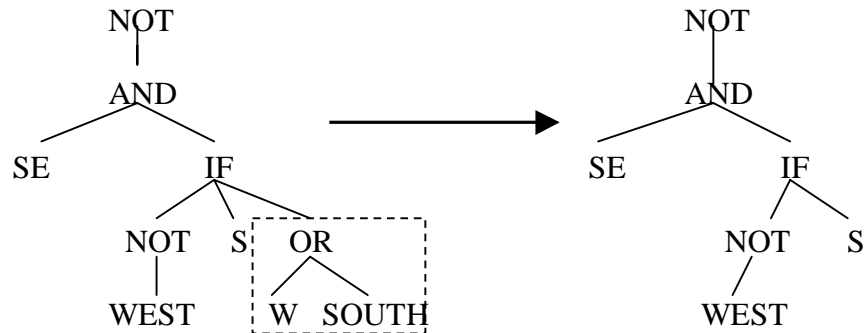
- Replace what was under the selected node in the father tree with what was under the selected node in the mother tree. In basic GA, we create only one child from crossover to ensure variety.



Child program produced by crossover

Mutation

- Pick a node at random and replace it and its subtree by an randomly generated (possibly null) subtree.



Problems and Limitations with Genetic Algorithms

- Highly **computationally intensive** (most need to be run overnight, even on fast machines)
- Requires a **well-defined fitness function** to evaluate
- Needs to have a large population, otherwise the state space is too big and the chance of finding an optimal program is small.
- No guarantee that the optimal program will be found.
- No general theory of how to build fitness functions
- There are many problems for which it is impossible to build a fitness function (eg, when fitness is user specific, as in a word processing program)
- **Search space** for many problems is too big; our example has a very limited world, with limited perception and possible actions.

Additional Thoughts on GA

- They may have a tendency to overfit
- As generations pass, it may be a good idea to **reduce noise** by choosing more parents for the next generation, and doing fewer crossovers and mutations
- Why is this machine learning? Because it improves through **experience (trials)** over **time**. Its learning does not require any **memory**.