Turtles! Hill climbing! Hammers! Paper bags!

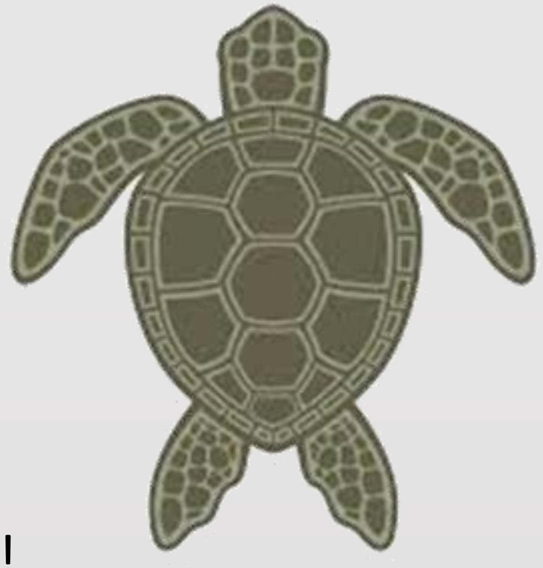
Frances Buontempo

ACCU 2018

https://github.com/doctorlove/turtles.git

Overview

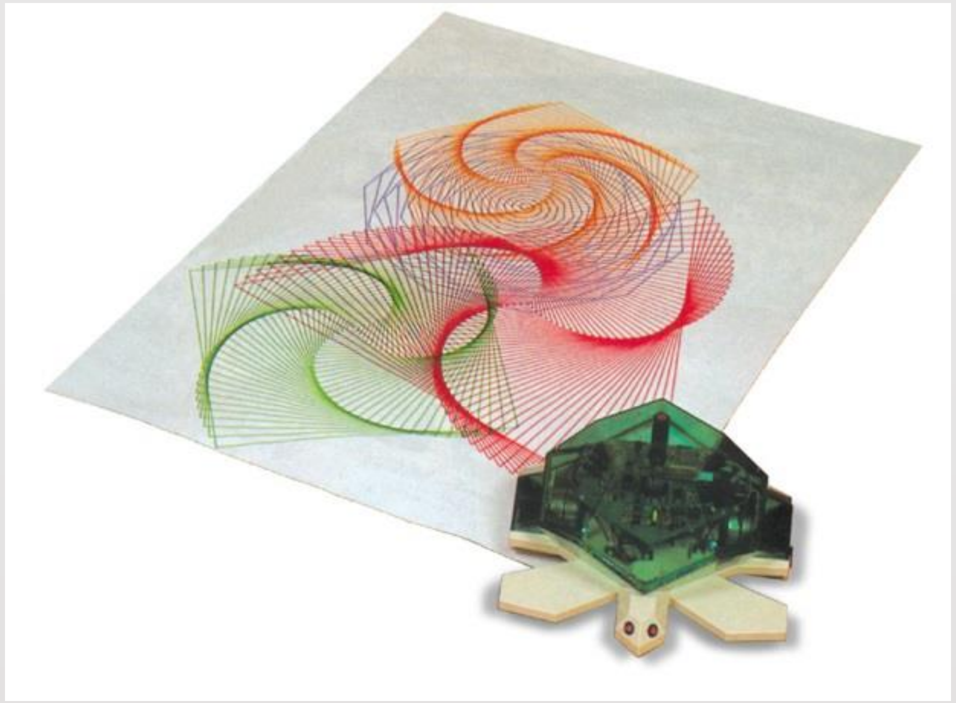
- Turtles
  - In Python
- Hill climbing
  - Or descending
- Hammers
  - Simulated annealing
- Particles Swarms – a bonus!
  - Local and global search
- Gradient descent
  - Quick reminder about gradients and comparison with hill climbing
- Neural networks
  - A quick overview
- Paper bags
  - Permeate everything
  - Some have straight edges, some don't
  - They could be shown 3D



Context: Optimisation

- Objective functions, e.g.
  - `f(x) = 0`, what's `x`?
  - `f(inputs) == expected_output` => `f(inputs) - expected_output = 0`
- Make it small
- Make it big
- Simply the best
  - Thursday's workshop.
    - https://conference.accu.org/2018/sessions.html#XSimplytheBestOptimisingwithanEvolutionaryComputingFramework
- Error measures
  - Different error measures e.g. https://en.wikipedia.org/wiki/Cross_entropy

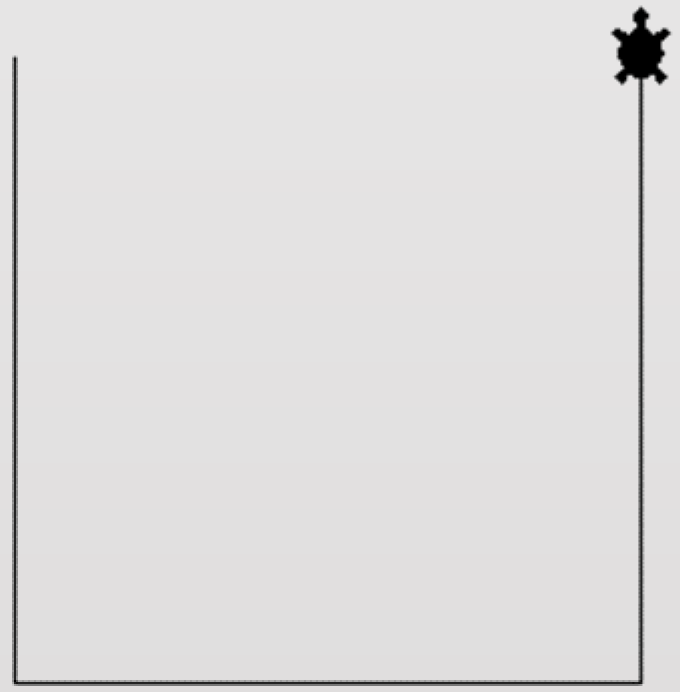
Logo



- Yay! Robot turtles
  - Sadly none today
  - Robot turtle draws lines
  - You program the moves
  - Like spirographs
- Seymour Papert, 1960s
  - "body syntonic" reasoning.
  - Yay! Neural networks!
  - "Perceptrons: An Introduction to Computational Geometry"
    - By Marvin Minsky and Seymour A. Papert
    - https://mitpress.mit.edu/books/perceptrons

Turtles!

```
import turtle


def draw_bag():
  turtle.shape('turtle')
  turtle.right(90)
  turtle.forward(70)
  turtle.left(90)
  turtle.forward(70)
  turtle.left(90)
  turtle.forward(70)


turtle.setworldcoordinates(-70., -70., 70., 70.)
draw_bag()
turtle.mainloop()
```



Code your way out of a paper bag

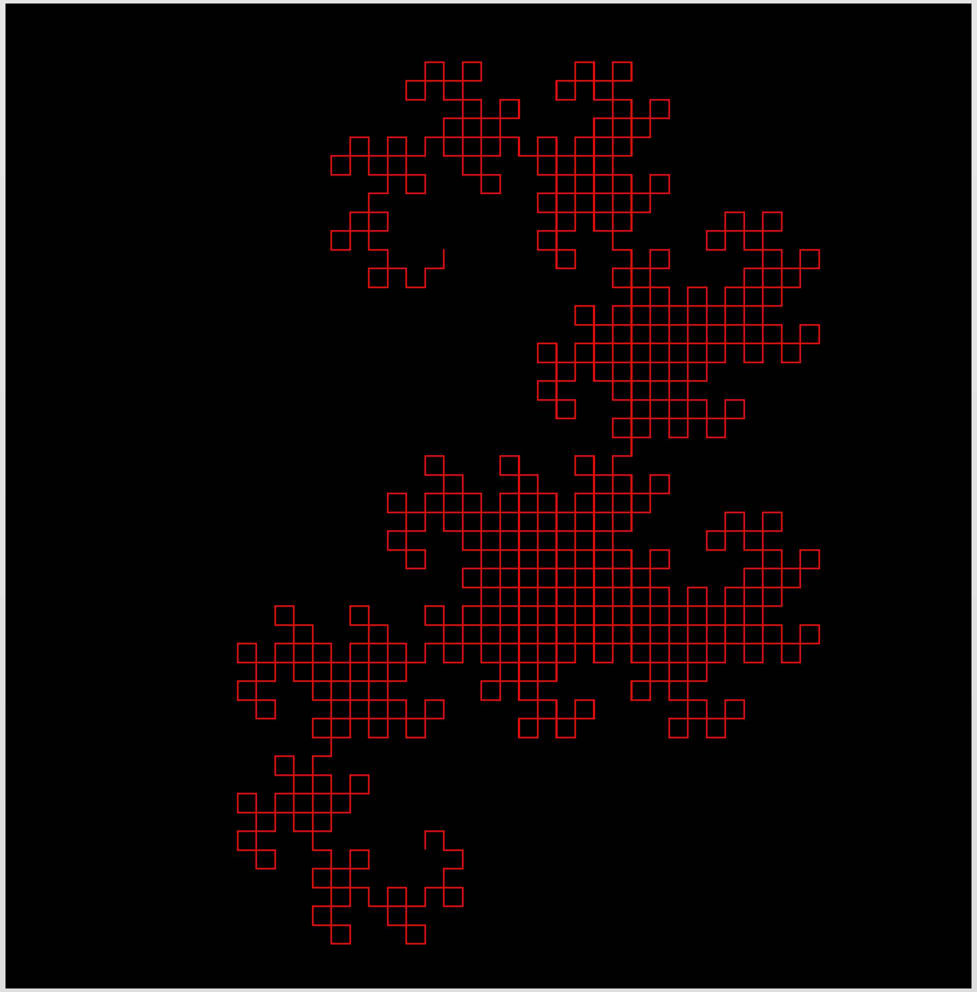
- Jeff Atwood, co-founder of Stack Overflow, considers many programmer's inability to program on his blog.
  - https:blog.codinghorror.com/why-cant-programmers-program
- He quotes Dan Kegel as saying "**We're tired of talking to candidates who can't program their way out of a paper bag.**"
- This is your chance to stand out from the pack!
  - `Imagine a paper bag, represented by three sides.`
  - `Pick a point inside, and move to new points until some are out of the paper bag.`
  - `Draw a line joining these points, so you can see the path taken.`

Code your way out of a paper bag

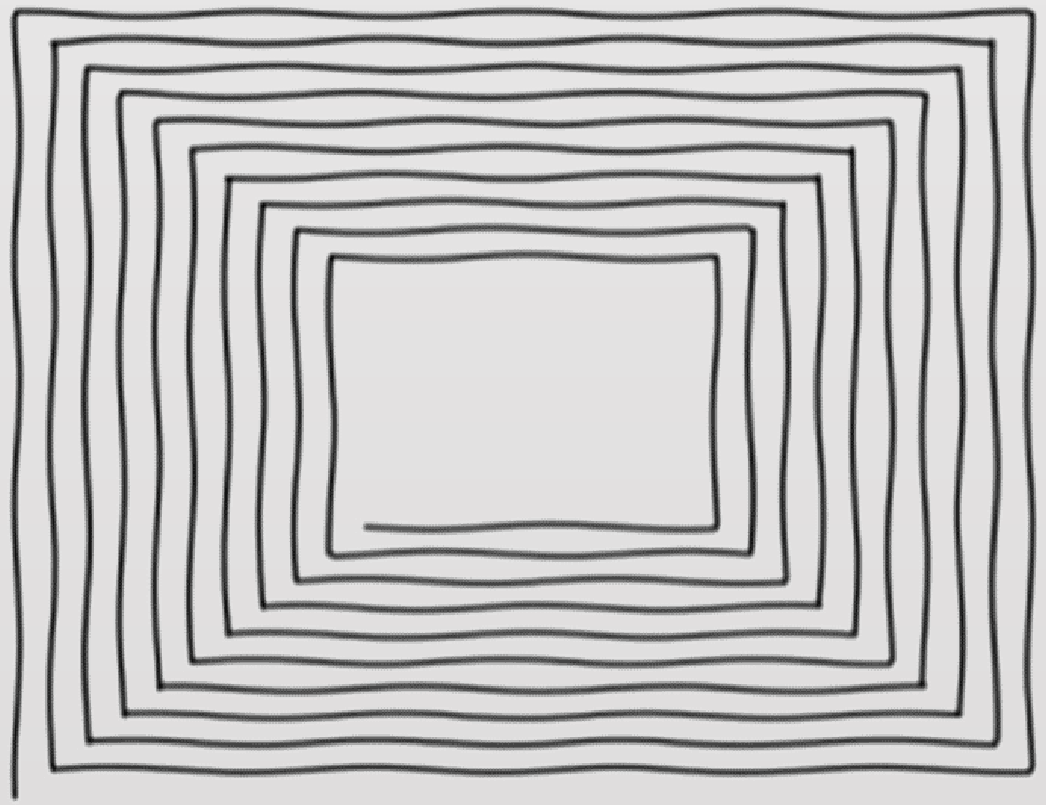
- Lindenmayer Systems (L-systems)
  - http://python3.codes/drawing-fractals-with-lindenmayer-systems/
- Recursion
- Grammars
- Trees, ferns…
- Self-similar
  - fractals

Dragons! T-shirt!

```
from turtle import*

def X(n):
    if n>0:    L("X+YF+",n)
def Y(n):
    if n>0:    L("-FX-Y",n)

def L(s,n):
    for c in s:
        if   c=='-': lt(90)
        elif c=='+': rt(90)
        elif c=='X': X(n-1)
        elif c=='Y': Y(n-1)
        elif c=='F': fd(12)

bgcolor('black')
pencolor('red')
up()
goto(-20, 120)
down()
X(10)
hideturtle()

mainloop()
```



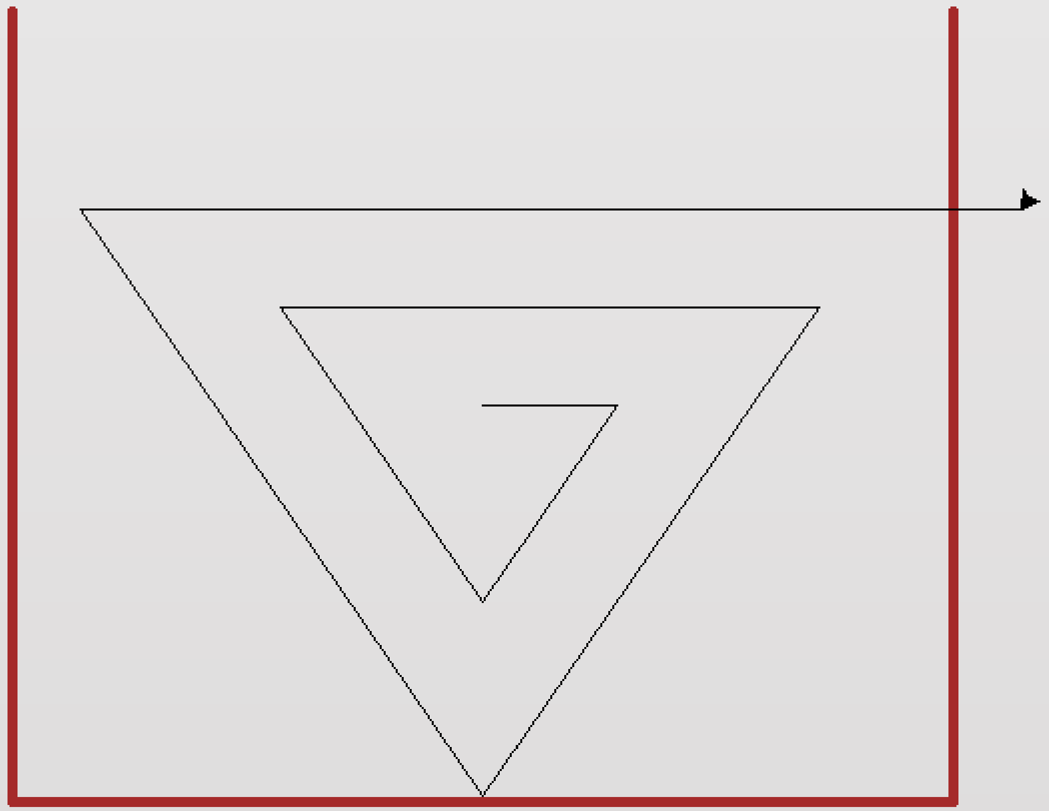
Spirangles



Spirangle code

```
def draw_triangles(number):
  t = turtle.Turtle()
  for i in range(1, number):
    t.forward(i*10)
    t.right(120)
```

- Note to self: do a demo.
  - `triangles.cmd`



How did the turtle get into the paper bag?

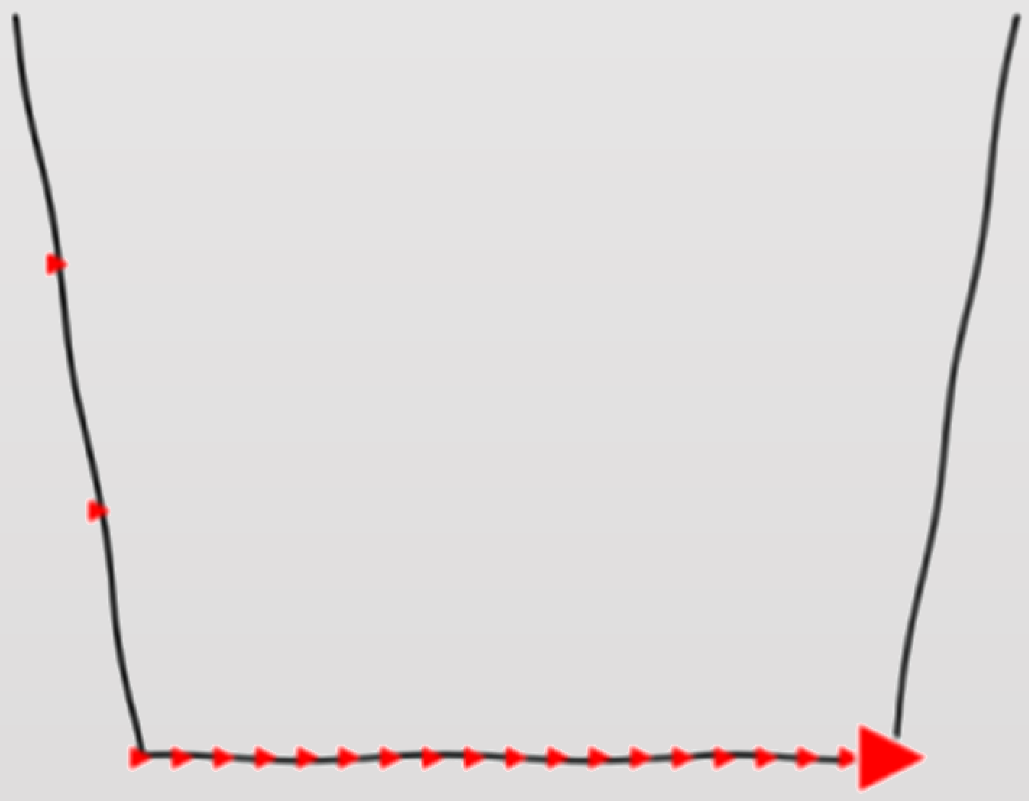
- Not a joke
  - Punchlines welcome
- I don’t know
- But let’s consider a few ways
  - Hill climbing
  - Simulated annealing
  - Particle swarms

What shaped paper bag?

- Aside – mathematical functions
  - One y value per x value…
  - So, no rectangular bags here
- Some near rectangular
  - Some crumpled up
  - Start line 1D (lines)
  - Then consider extensions

Hill walking

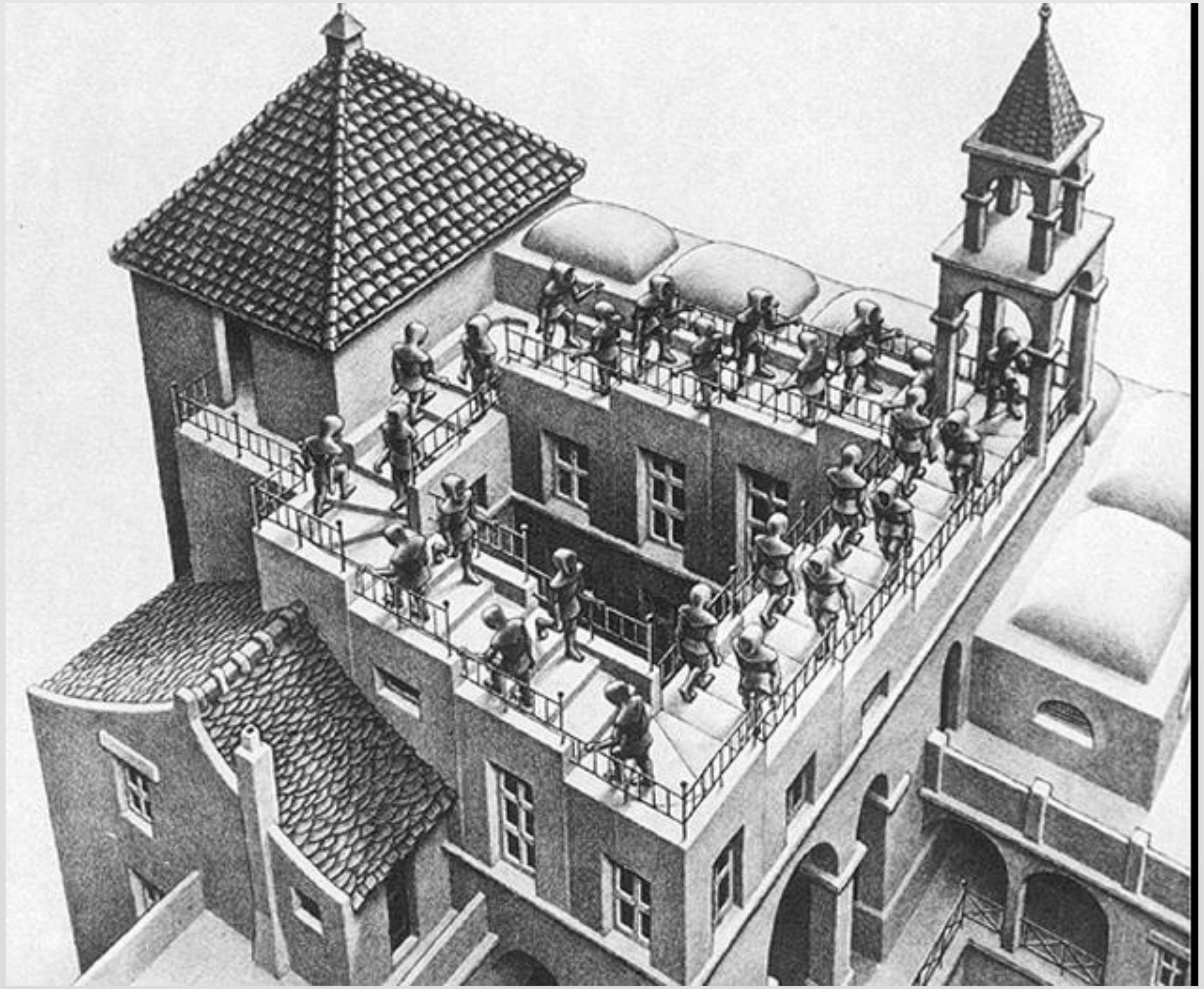
```
def seek(x, step, f):
  height = f(x)
  while True:
    if f(x-step) < height:
      x -= step
    elif f(x+step) <= height:
      x += step
    else:
      break
    height = f(x)
    yield x, height
```



What can possibly go wrong?

- What step size?
  - Called a **learning rate** in machine learning
  - Learning rate is a meta-parameter
  - Decay…
  - Beware oscillations:
    - https://www.oreilly.com/ideas/contouring-learning-rate-to-optimize-neural-nets?imm_mid=0f59d0&cmp=em-data-na-na-newsltr_20170823
- What shaped bag?
  - This affects what can go wrong…

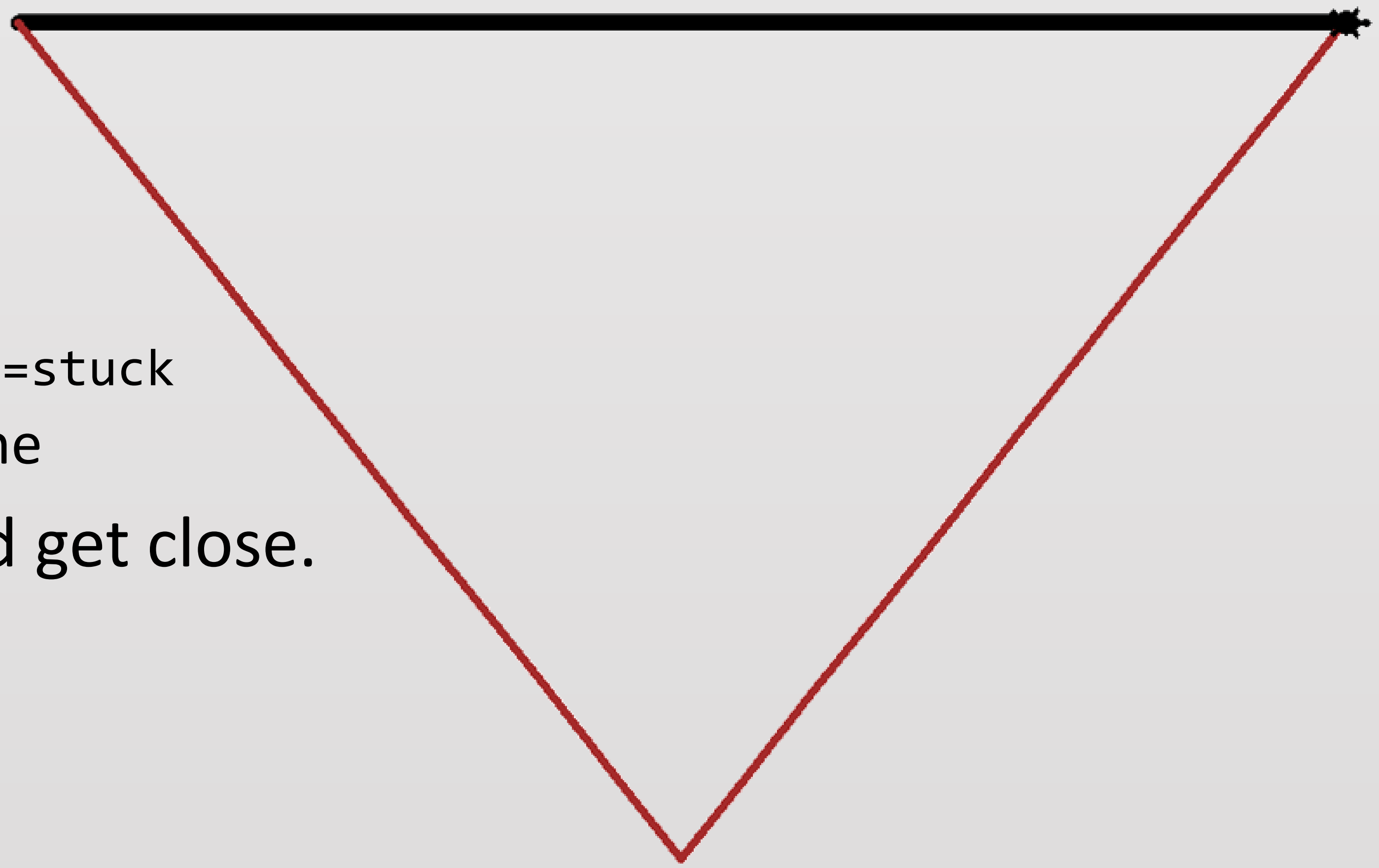
What else could go wrong?



An almost rectangular paper bag

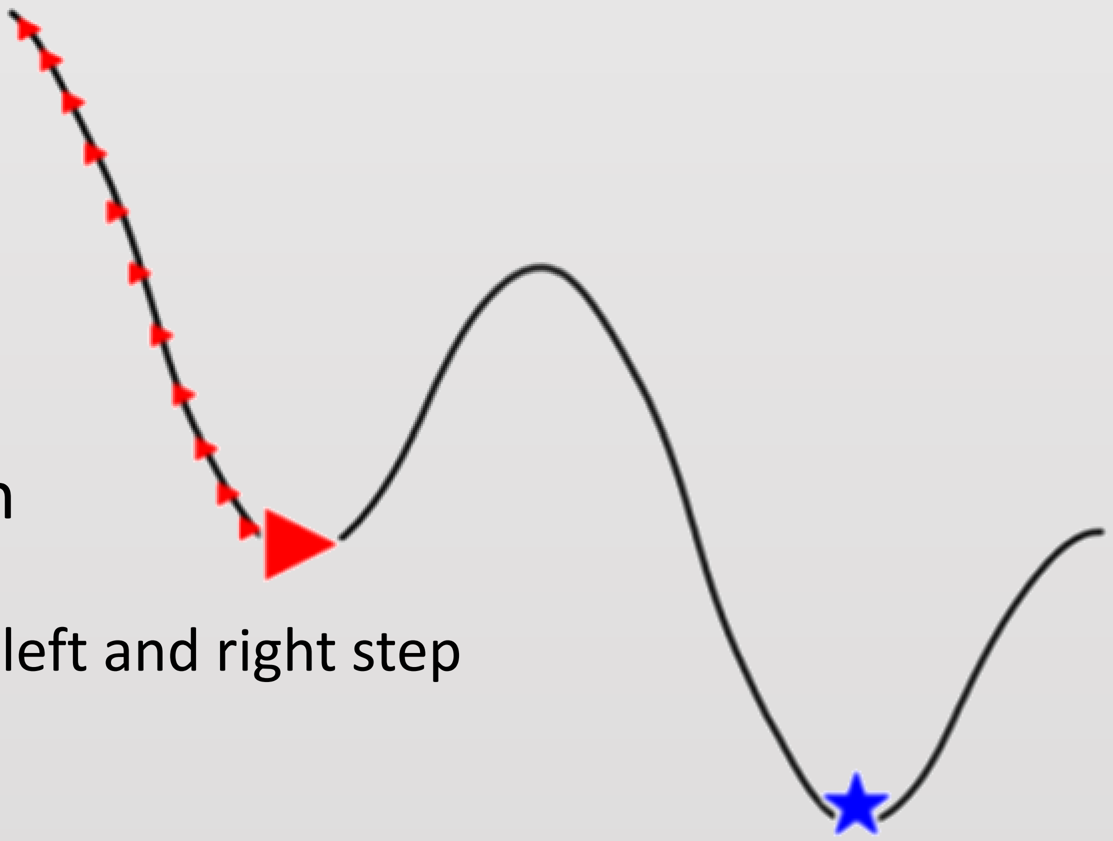
- Note to self to do a demo
  - `slanty.cmd`
- Show off the quadratic bag too
  - $y = x^2$
  - quad.cmd
- But…

Stuck!



- f(x) = | x | on [-10, 10]
  - fabs(x)
- Step size constant 20
  - `python into_bag.py -f=stuck`
  - Could alter this over time
- 10 would work, 3 would get close.
- Convergence
  - Or not

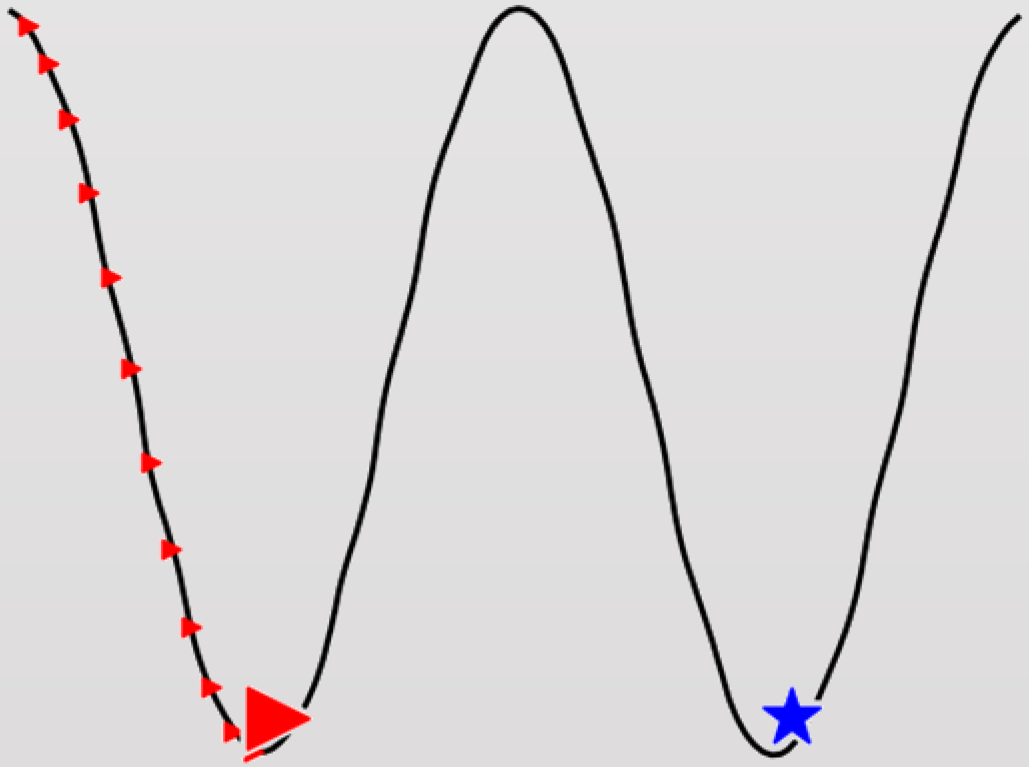
Crumpled bag



- f(x) = 5*cos(x) - x
- One minimum on screen
  - A **local minima**
  - Turtle considers a single left and right step
  - Guess what happens!
- Note to self
  - Do a demo
  - `crumple.cmd`

More than one minimum

- E.g. cosine
- Turtle considers a single left and right step
- Guess what happens!
- Note to self
  - Demo
  - `cos.cmd`



Global v. local minima

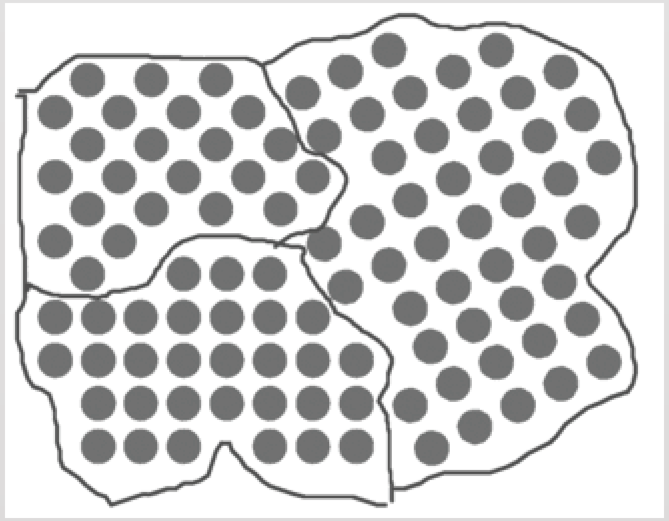
- Don’t be ***greedy***!
- Hill-climbing – local only
- How do you avoid this?
  - Make some suggestions…
- Do the maths
  - Not always possible
- Brute force
  - Not always time
- Do something random…

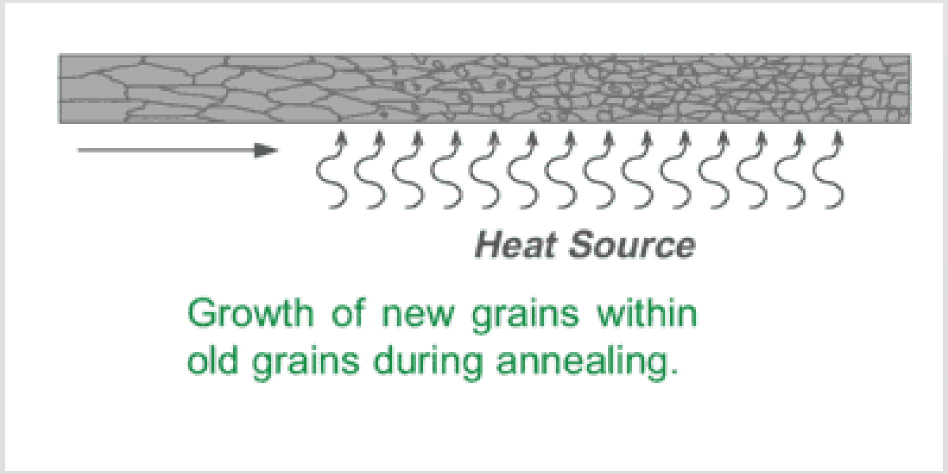
Try random stuff

- Metaheuristic: can find global minima
  - often population based e.g. ACO, PSO, GA, “evolutionary computing”
- Simulated annealing
  - Like hill climbing but tries random points too, once in a while
  - Probabilistic approximation technique
- Particle swarm optimisations
  - Particles remember a local best and a global best and compare notes
  - Other swarm optimisations too

Actual annealing

- "… a heat treatment that alters the physical and sometimes chemical properties of a material to increase its ductility and reduce its hardness, making it more workable." (Wikipedia)
- Statistical thermodynamics: Bolztmann distribution
  - `p(state)=exp(mumble/T),` for temperature T





Simulated annealing

```
For a while
    Choose potential new x
        (e.g. step left, right and a random jump)
    Find y = f(x)
    Better than before?
        Good.
        x = potential new x.
        Continue
    No? OK, **maybe** choose it anyway
```

Algorithm details: **maybe** requirements

- Should decrease as the temperature, T, decreases
  - i.e. make turtle less likely to jump over time
  - You need to pick a temperature out of the air and make in smaller over time
  - Convergence
- Want really bad new values to be unlikely
  - How much worse is it?
  - What does it **cost**?
    - A function of new and old values
- Gives a number from 0 to 1
  - Is a probability
- Pick a random number from 0 to 1 to compare
  - take the new (worse) value if the transition probability is higher

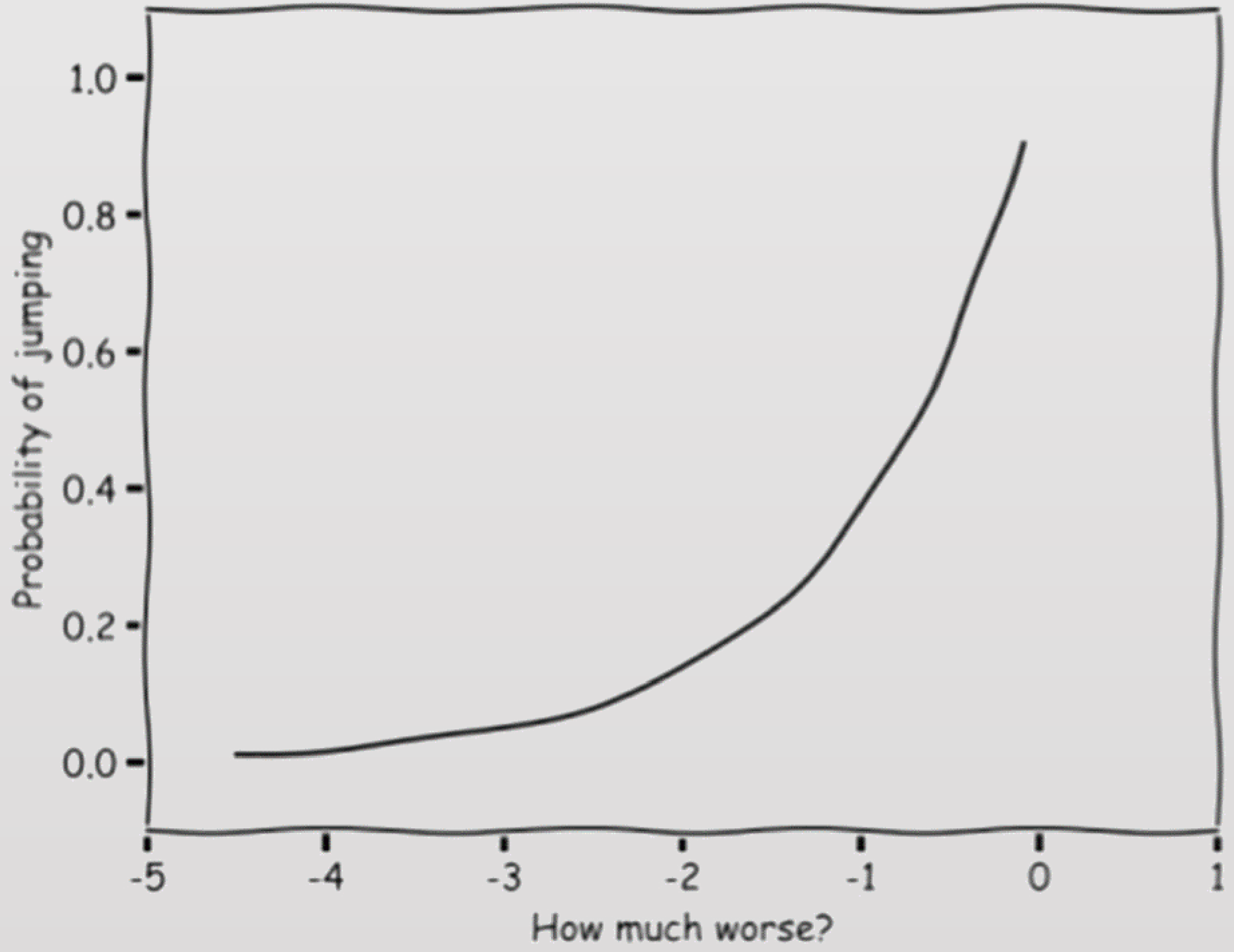
Maybe (jump) function choice

- Called “transition probability”
- Use

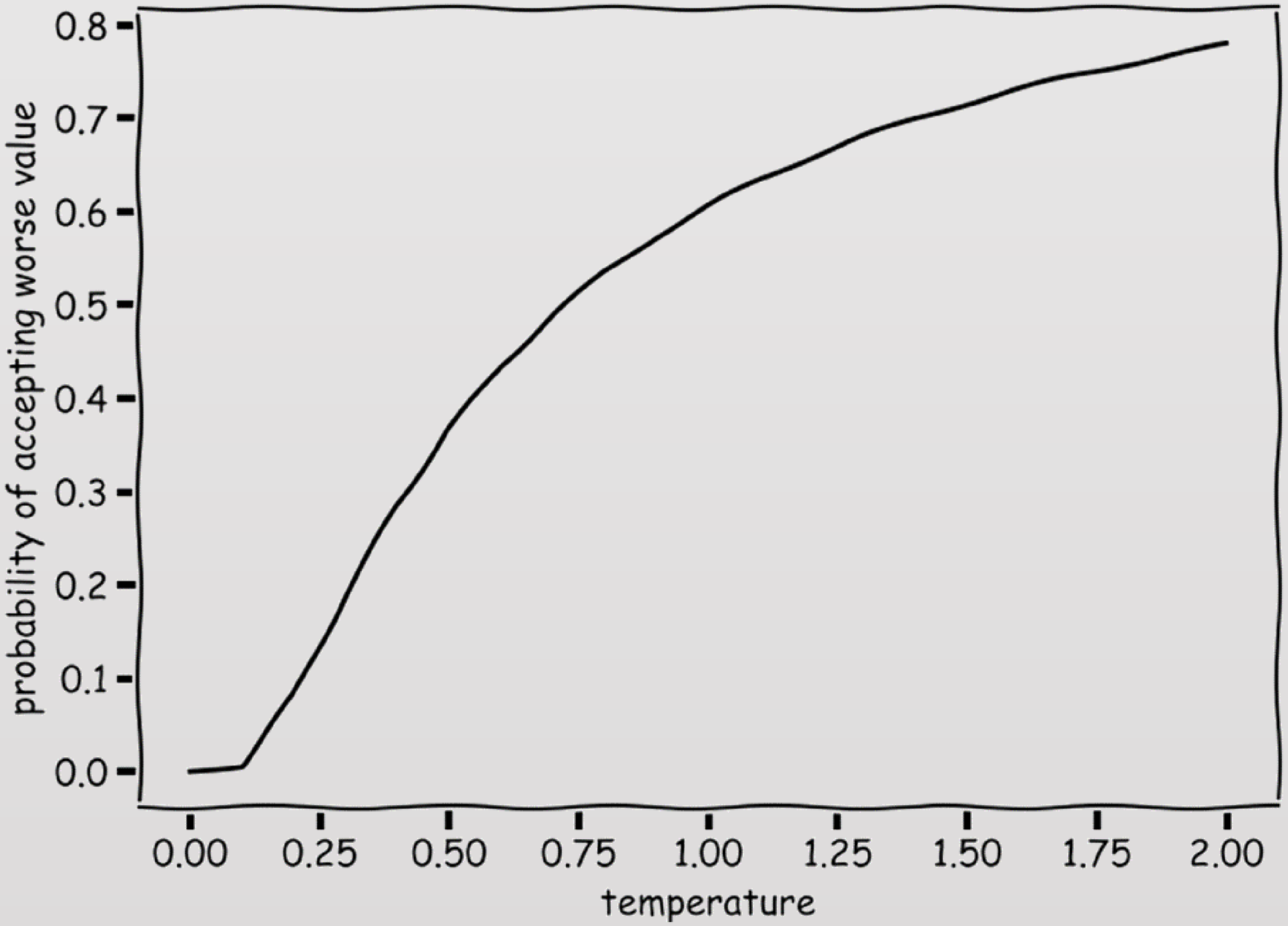
$$e^{\frac{cost(old)-cost(new)}{T}}$$

- Like “the embodied energy of metal particles as they are cooled slowly after being subjected to high heat.”
    - From Boltzmann distribution from statistical thermodynamics
- Based on science!

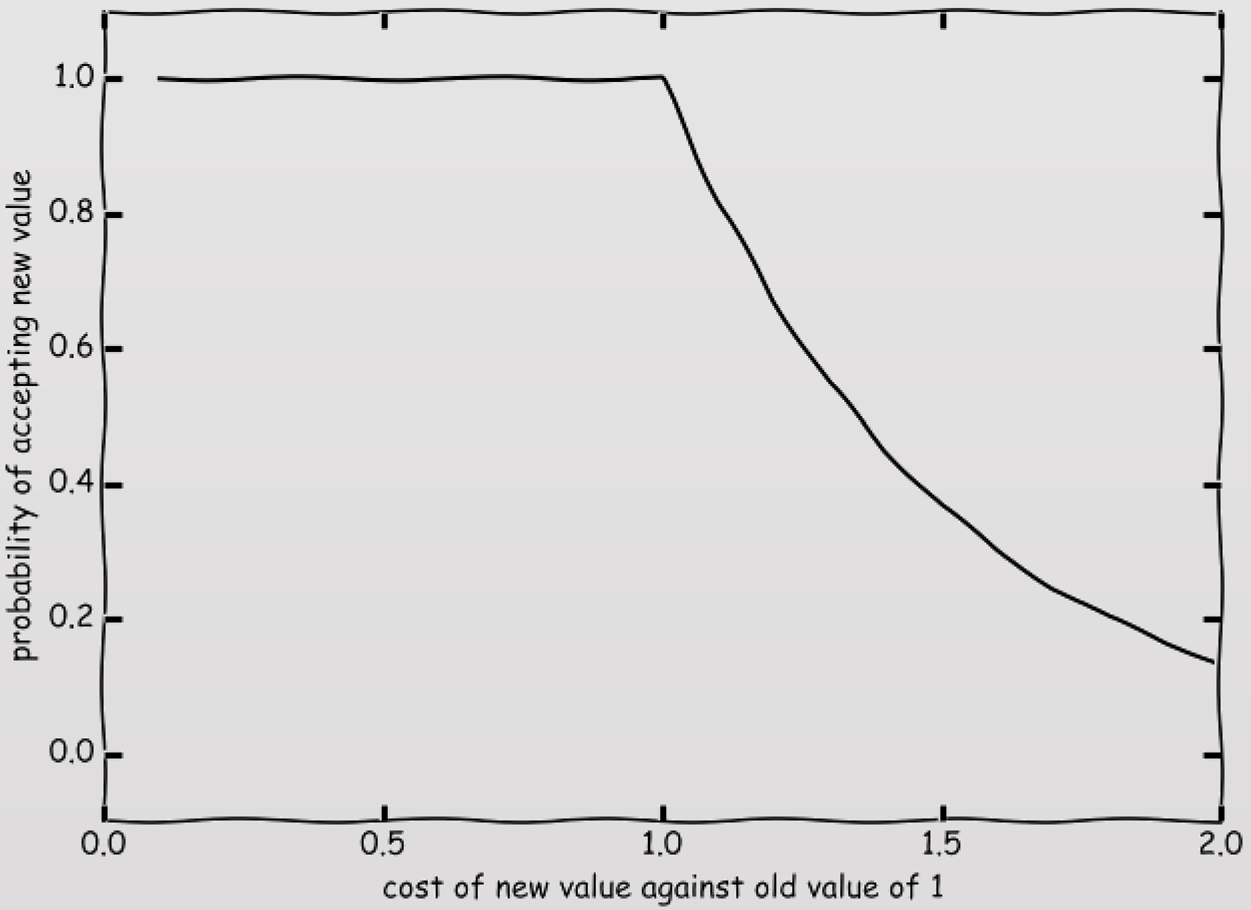
Exponential curve



Maybe: cooler => less likely



Maybe: new value gets worse



Algorithm details:

```
For a while
  Choose potential new x
        (e.g. last x +/- step or random jump)
  Find y = f(x)
  Better than before? Good. Continue
  No? OK, maybe choose it anyway
```

- For a while == until “cooled”
  - Start at temperature = 10.0?
    - Try something, see what happens
  - Make temperature -= 0.1 until we hit -5
    - Other numbers are available
  - Often temperature *= α
    - For some α (maybe between 0.8 and 0.99)

Algorithm details:

```
For a while
  Choose potential new x
        (e.g. last x +/- step or random jump)
  Find y = f(x)
  Better than before? Good. Continue
  No? OK, maybe choose it anyway
```

- Step size: new x value to try
    - Try +/-0.1, and something random
    - Decrease when temp <= 0
- Random?
    - Gaussian

Algorithm details:

```
For a while
  Choose potential new x
       (e.g. last x +/- step or random jump)
  Find y = f(x)
  Better than before? Good. Continue
  No? OK, maybe choose it anyway
```

- Better == cost function
- Use the y coordinate:
  - Lower is better

Transition probability

```
def transitionProbability(cost_old, cost_new, temperature):
  if temperature <= 0:
    return 0
  elif cost_new < cost_old:
    return 1
  else:
    return math.exp((cost_old - cost_new) / temperature)
```

Seek

```
def seek(x, step, f, temperature):
  best_x, best_y = x, f(x)
  while temperature > -5:
    if temperature < 0: step /= 2.0
    possible = [x-step, x+step, x+random.gauss(0, 1)]
    x, jump = find_new_x(possible, f, x, best_y, temperature)
    if not jump:
      best_x = x
      best_y = f(x)
    yield x, f(x), temperature, jump
    temperature -= 0.1
```

find

```
def find_new_x(possible, f, x, best_y, temperature):

  if len(possible) == 0:
    raise ValueError("Possible points empty")
  jump = False

  for new_x in possible:
    new_y = f(new_x)
    if new_y < best_y:
      x = new_x
    elif transitionProbability(best_y, new_y, temperature) > random.random():
      jump = True
      x = new_x

  return x, jump
```

Simulated annealing

- Note to self – do demo(s)
  - `sa_slanty_bag`
  - `sa_quad`
  - `sa_cosine_slope`
  - `sa_cosine`

Caveats

- Decisions
  - Parameters: where to start,…
  - Cooling scheme: Subtract, multiply,…
    - See http://www.fys.ku.dk/~andresen/BAhome/ownpapers/permanents/annealSched.pdf
- Might not work
  - Will it try more points than a brute false approach?
  - If you can do the maths, then do the maths
  - (Trying to code it for such problems as a learning exercise/for a talk is ok)
- Nice walkthrough:
  - http://katrinaeg.com/simulated-annealing.html

What about several minima?

- E.g. cosine
- Try several turtles…
- Questions?
  - How many?
  - Where do they start?
- Note to self
  - Demo: sa_cosine_turtles
- Keep variety
  - Niching methods etc

Swarm!

- Success!
- Let’s have more turtles!
- Particle swarm
  - **Paper Bag Escapology Using Particle Swarm Optimisation**
    - https://accu.org/index.php/journals/2023

Particle Swarm Optimisation, PSO

```
Choose n
Initialize n ~~particles~~turtles randomly
For a while:
  Update best global position (for synchronous)
  Draw turtles current positions
  Move turtles (for asynch: update global best here)
  Update personal best position for each turtles
```

PSO Move

- **Position, x, moves by a velocity, v over time, t**
- $x_{t+1} = x_t + v_{x,t+1}$
  - And y, z, w…
- $v_{x,t+1} = wv_{x,t} + c_1 r_1 (p_t - x_t) + c_2 r_2 (g_t - x_t)$
  - Constants w, c_1, c_2
  - Random numbers r_1, r_2
  - Personal and global best variables to track p_t, g_t
  - And you had to pick a temperature and cooling scheme out of the air for PSO
    - Now look! THREE parameters!!!

Particle

```
class Particle:
  def __init__(self, x, name):
    self.x = x
    self.best = x
    self.velocity = random.random()
    self.name = name
    self.history = []
```

PSO: start with something random

```
def initialise(count, min_x, max_x):
  particles = []
  for i in range(count):
    x = random.uniform(min_x, max_x)
    particles.append(Particle(x, str(i)))
  return particles
```

PSO Swarm

```
def swarm(count, min_x, max_x, epochs, f):
  particles = initialise(count, min_x, max_x)
  best = find_best(particles, particles[0].best, f)

  for _ in range(epochs):
    yield particles
    best = find_best(particles, best, f)
    update_velocity(particles, best)
    move(particles, min_x, max_x, f)
```

Find best

```
def find_best(particles, best, f):
  for particle in particles:
    if f(particle.x) < f(best):
      best = particle.x
  return best
```

Update velocity

```
def update_velocity(particles, best, w=0.1, c1=0.4, c2=0.2):
  for particle in particles:
    r1 = random.random();
    r2 = random.random();
    particle.velocity = w * particle.velocity + \
                   c1 * r1 * (particle.best - particle.x) + \
                   c2 * r2 * (best - particle.x)
```

PSO Move

```
def move(particles, min_x, max_x, f):
   for particle in particles:
     x = particle.x + particle.velocity
     if max_x < x:
       particle.x = max_x
     elif min_x > x:
       particle.x = min_x
     else:
       particle.x = x
     particle.history.append((particle.x, particle.velocity))
     if f(particle.x) < f(particle.best):
       particle.best = particle.x
```

PSO

- Note to self: demo
  - `pso_quad_1.cmd`
  - `pso_quad_2.cmd`
  - `pso_quad_10.cmd`
  - `pso_cosine_slope.cmd`
  - `pso_cosine.cmd`
- Parameters
  - Count? Epochs?
  - Using w = 0.5, c1 = 0.3, c2 = 0.3 fails badly
  - Using w = 0.5, c1 = 0.4, c2 = 0.2 looks better for >=2 turtles
- Question: does a swarm of 1 ever work?
  - Answer – probably not
- Does mean we have one example we can check by hand
  - Were it not for the pesky random numbers

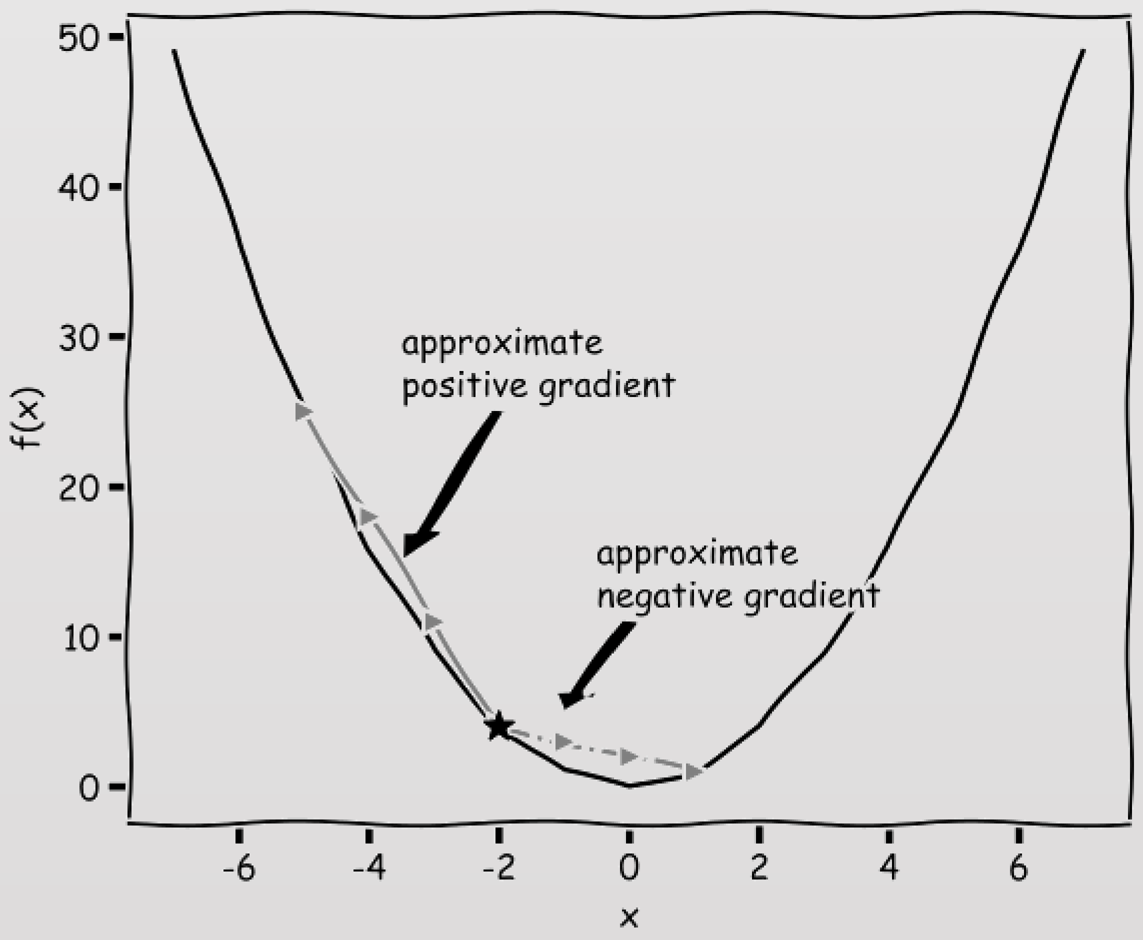
A mixed success/fail

- Equidistant carrots
    - two points are equally good
- Parameters might need more experimentation
    - Certainly shouldn’t be hard coded!
- How do you test code like this?
    - If it doesn’t work consistently is your implementation wrong?
    - Are your parameters wrong?
    - What does one turtle do?
        - What would you expect?
    - Discuss…

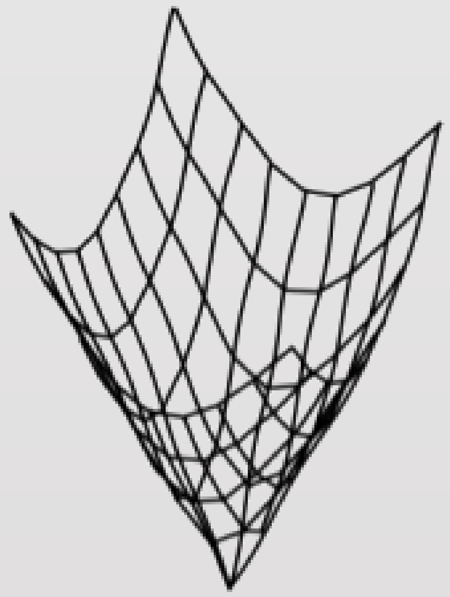
Gradient descent

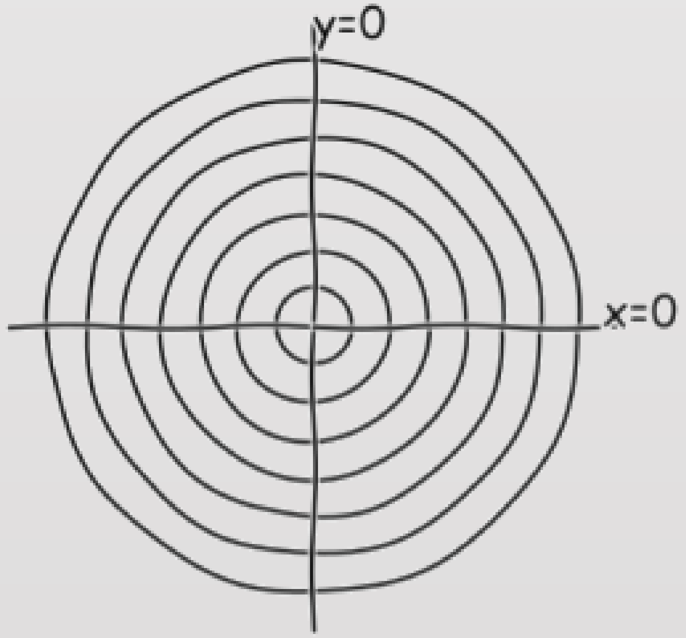
- Hill climbing is doing a gradient in one direction
- Could do along each axis for more dimensions
- Or linear combinations of axes…
- Gradient descent:
    - Vanilla version – **batch v.** mini-batch
        - When fitting a big data set
            - https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/
    - Stochastic gradient descent
    - Fixed learning rate v. decay
- Alternative: Simplexes
- Standard pathological surfaces

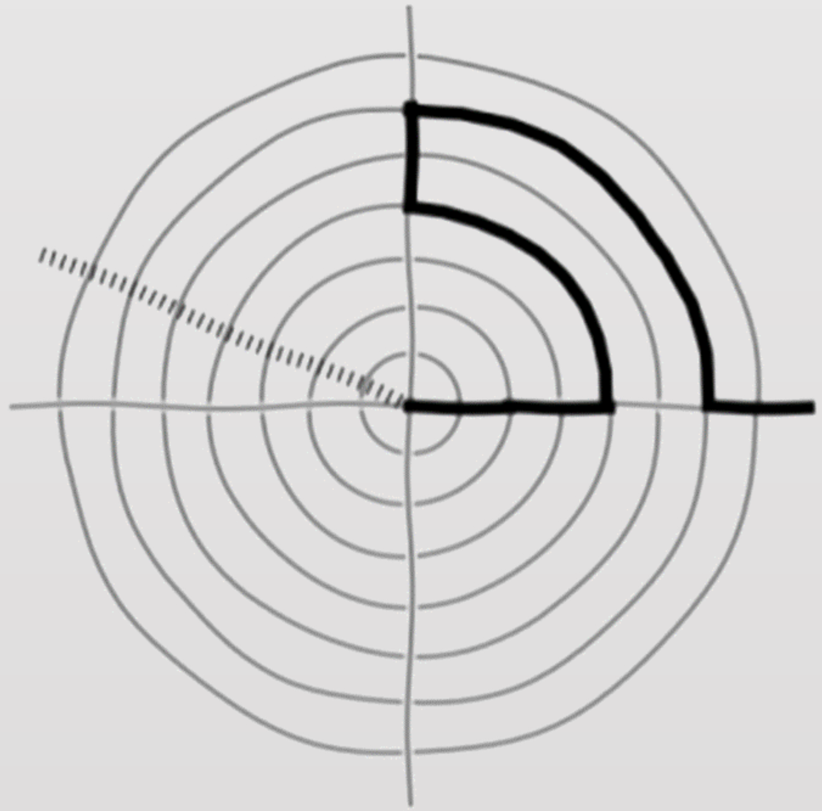
What's a gradient?



Gradients c.f. hill climbing







Gradients: problems

- Finite difference
  - Numerical approximation to gradient
  - For when we can’t do the maths
- Numbers!
- The “vanishing gradient problem”
  - https://en.wikipedia.org/wiki/Vanishing_gradient_problem
- Especially in neural networks:
  - “By the time the error signals traveled all the way back to the early layers, they were so small that the net couldn’t learn.” “deep (many layers) and “wide” (many parallel operations)”

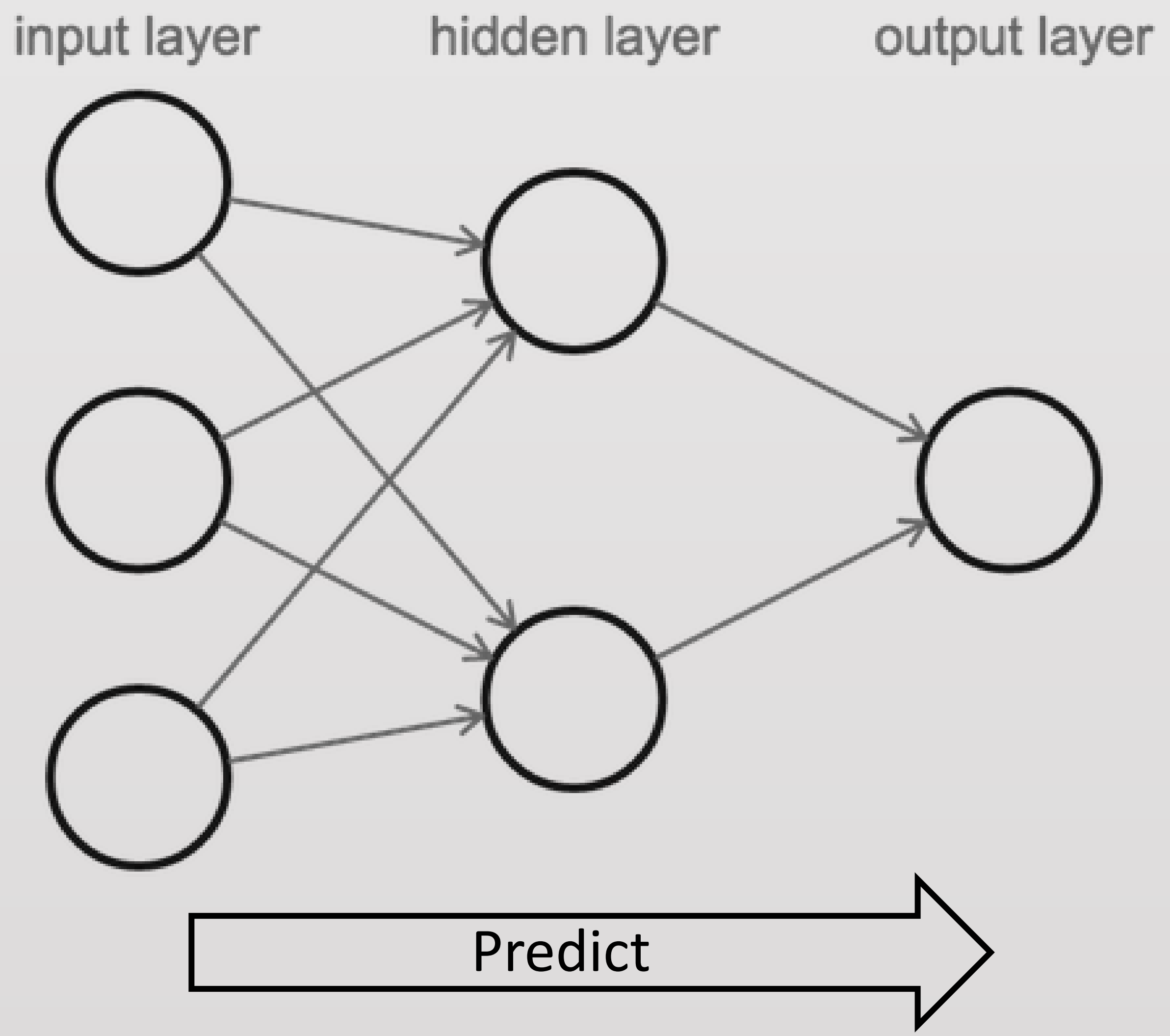
Gradient descent algo

- Minimise $f(\overline{x}) = f(x_1, x_2, \dots, x_n)$
- Where function f has gradient $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right)$
  - A vector – or linear combination of each axis direction
- Given a starting point $\overline{p_n} = (p_1, p_2, \dots, p_n)$
- Move to $\overline{p_{n+1}} = \overline{p_n} - \gamma \nabla f(\overline{p_n})$
  - γ is the learning rate we saw before
  - Might vary
- //TODO – maybe some code? Or a turtle?

Neural Networks

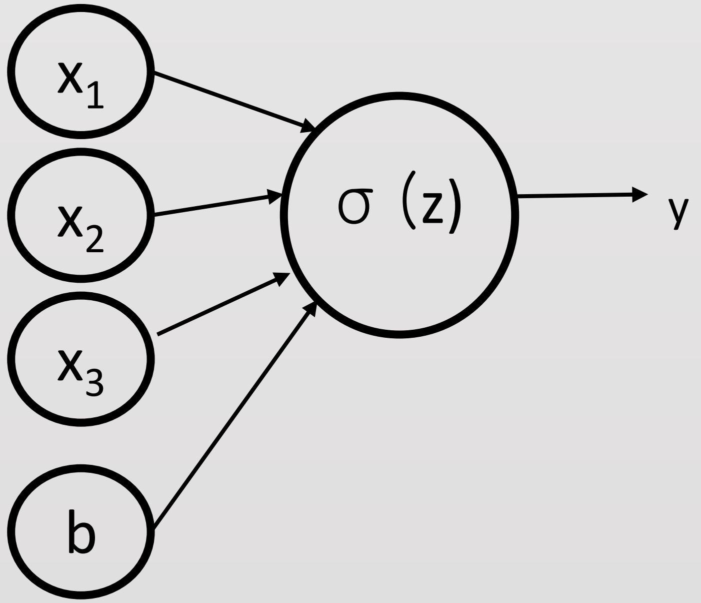
- “Neural networks, in the most general sense, are function approximators”
  - https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained
- History:
  - 1950s Rosenblatt’s “Perceptron” (a linear classifier)
  - 1960s Minsky & Papert
  - -> 1990s academic stuff then hits the world.
    - See http://ml4a.github.io/ml4a/neural_networks/
- They are usually trained on data (supervised learning)
  - `f(input)= output`
  - f worked out by ~~magic~~ mathematics

Typical feedforward NN

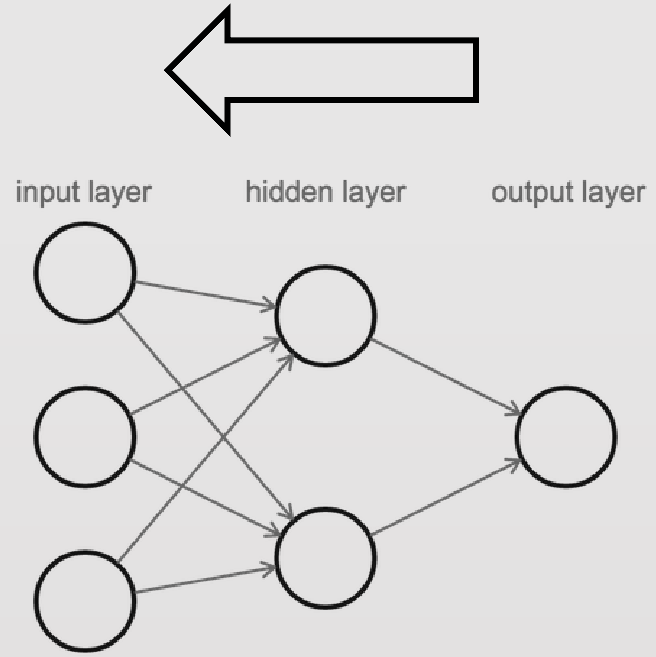


Overall idea

- Linear classifier/regression
  - $f(x, y) = b + w_1 x + w_2 y$
- More dimensions
  - $f(X) = b + \sum_i w_i x_i$
- Introduce an activation function, σ
  - Like a neuron firing in a brain
  - "Squishing" non-linear function
  - E.g. sigmoid $f(z) = {}^{1}/_{1+e^{-z}}$
- $z = b + \sum_i w_i x_i$
- $f(X) = \sigma(z) = \sigma(b + \sum_i w_i x_i)$



Backpropagation



- $f(X) = \sigma(z) = \sigma(b + \sum_i w_i x_i) = y$
  - Find bias and weights for training data
- How?
  - Do the maths
  - Finite difference
  - Automatic differentiation
  - Ian Sheret’s talk
    - https://skillsmatter.com/skillscasts/11337-automatic-differentiation-in-c-plus-plus-who-does-it-why-and-how
    - Sparse v. dense data => different techniques

~~Excuses~~Tech challenge

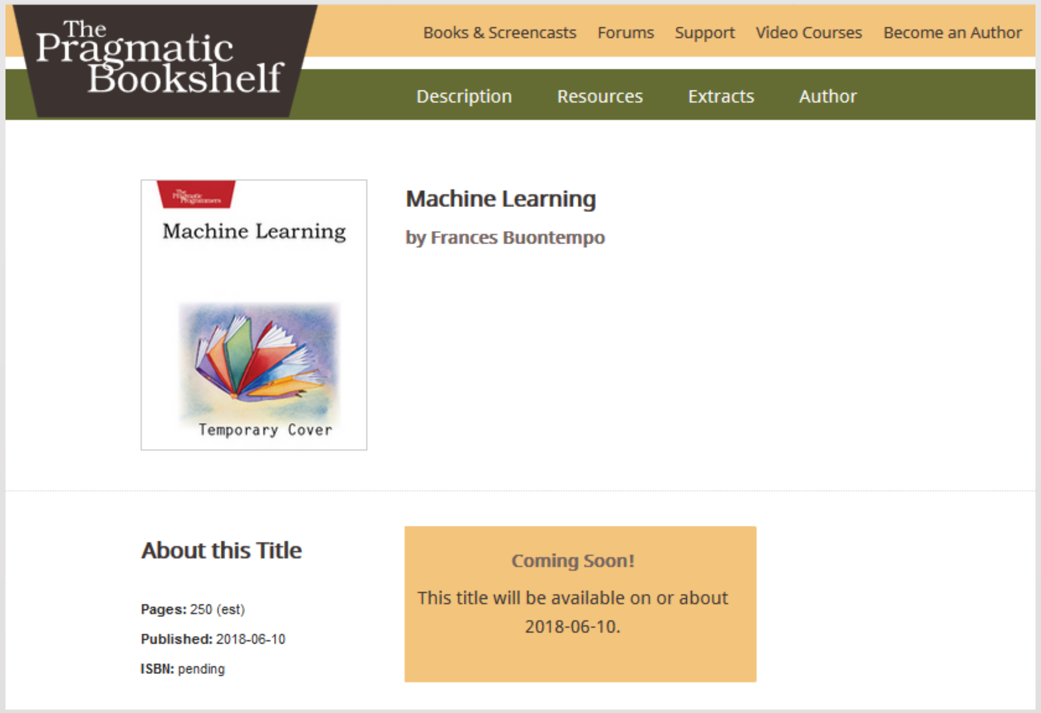
- Representing a >1D bag needs 3D turtle graphics
- I found this http://spencertipping.com/cheloniidae/
  - “Cheloniidae is an absurdly over-engineered turtle graphics library for Java.”

Wrap up

- Try it your self:
- Hill climbing
  - Solve a slider puzzle
    - https://towardsdatascience.com/solve-slide-puzzle-with-hill-climbing-search-algorithm-d7fb93321325
- Gradient descent
  - Make a NN from scratch
    - http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/

What did we learn?

- Turtle graphics are fun
  - Fran needs a 3D implementation in Python
- Problem solving
  - Try something and see if it works
  - Iteratively improve
  - Test your code as you go
- Paper bag escapology a parable for coding well?
  - Using machine learning is absurdly over-engineered!
- Upcoming book: http://pragprog.com/



Machine Learning

by Frances Buontempo

About this Title

Pages: 250 (est)

Published: 2018-06-10

ISBN: pending

Coming Soon!

This title will be available on or about 2018-06-10.