

The Many Types of AI

And the Obstacle of Generality

Thomas Nield

JavaMug

3/13/2019

Thomas Nield

Business Consultant at Southwest Airlines

Author

Getting Started with SQL by O'Reilly

Learning RxJava by Packt

Trainer and content developer at O'Reilly Media

OSS Maintainer/Collaborator

RxKotlin

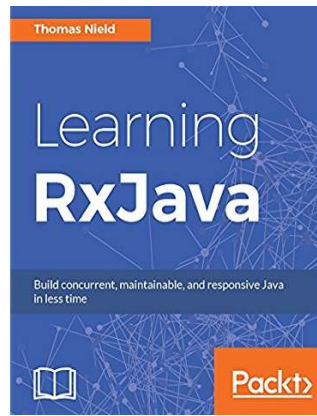
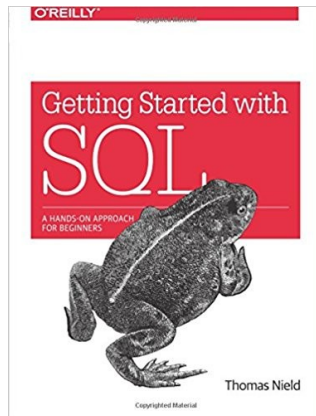
TornadoFX

RxJavaFX

Kotlin-Statistics

RxKotlinFX

RxPy



thomasnield9727



<https://github.com/thomasnield>



Thomas Nield

Agenda

Anecdote - Monty Hall Problem

Why Learn Mathematical Modeling

Live Examples

- Discrete Optimization
- Machine Learning



The Monty Hall Problem



The Monty Hall Problem

DOOR 1

DOOR 2

DOOR 3

Choose a door, one has a prize.
Two others have goats.

The Monty Hall Problem



DOOR 1

DOOR 2

DOOR 3

You choose **Door 1**.
What is the probability it has the prize?

The Monty Hall Problem

DOOR 1
33.33%

DOOR 2
33.33%

DOOR 3
33.33%

You choose **Door 1**.
What is the probability it has the prize? **33%**

The Monty Hall Problem

DOOR 1

DOOR 2

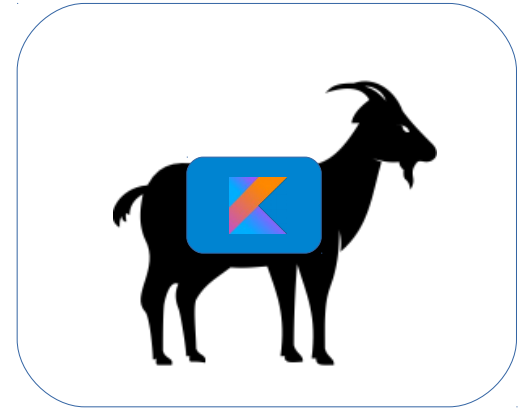


Twist! **Door 3** was just opened. It's a goat.
Did you want to switch from **Door 1** to **Door 2**?

The Monty Hall Problem

DOOR 1
?%

DOOR 2
?%



What is the prize probability of each door now?

The Monty Hall Problem

DOOR 1
?%

DOOR 2
?%



HINT: The prize probability of either door is not 50%

The Monty Hall Problem

DOOR 1
33.33%

DOOR 2
66.66%



The probability of **Door 1** is **33.33%** while **Door 2** is now **66.66%**.
You should switch! But why?

The Monty Hall Problem

33.33%



66.66%



33.33%



The Monty Hall Problem

According to Bayes Theorem...

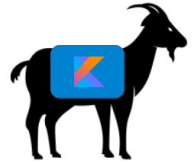
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(P_1|D_2) = \frac{P(D_2|P_1)P(P_1)}{P(D_2)} = \frac{(.5)(.33)}{(.5)} = .33$$

$$P(P_2|D_2) = \frac{P(D_2|P_2)P(P_2)}{P(D_2)} = \frac{(1)(.33)}{(.5)} = .66$$

DOOR 1
33.33%

DOOR 2
66.66%



The Monty Hall Problem

D_2 = Probability of door 2 being left = .5

P_1 = Probability door 1 contains prize = .33

P_2 = Probability door 2 contains prize = .5

$P(P_1|D_2)$ = Probability door 1 contains prize given door 2 is left = .33

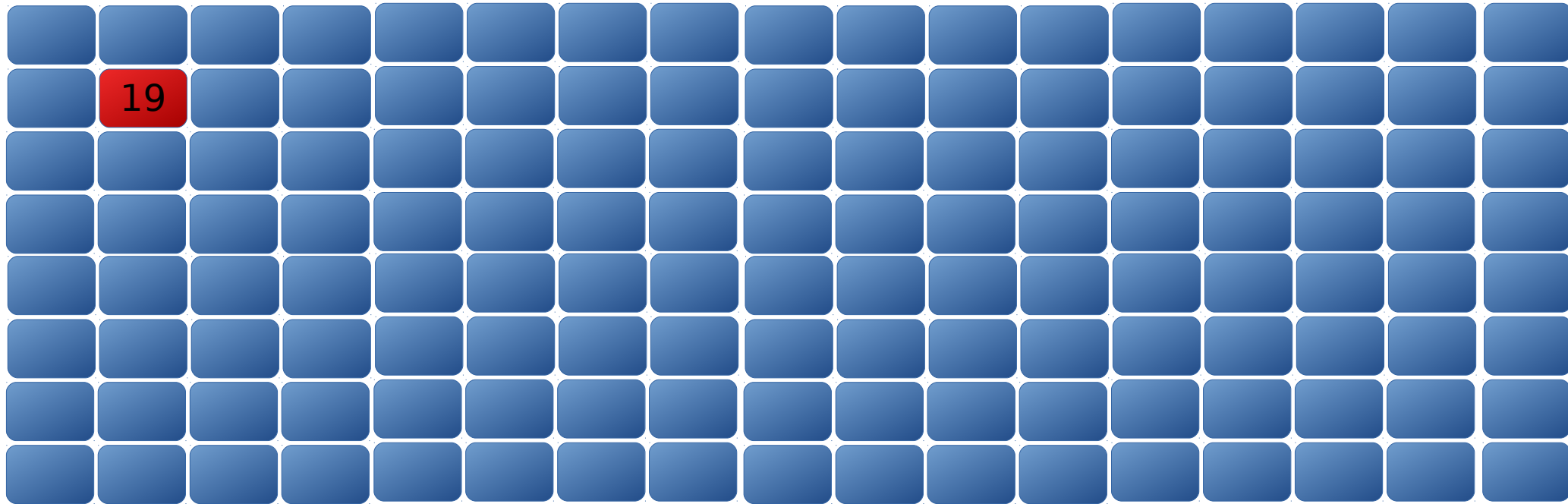
$P(P_2|D_2)$ = Probability door 2 contains prize given door 2 is left = .66

$P(D_2|P_1)$ = Probability door 2 is left given door 1 has prize = .5

$P(D_2|P_2)$ = Probability door 2 is left given door 2 has prize = 1.0

The Monty Hall Problem

Still confused? Hyperbolize! Imagine you had 1000 doors, and you chose **Door #19**.



The Monty Hall Problem

All other doors are opened but yours and **Door #77**. Inclined to switch now?



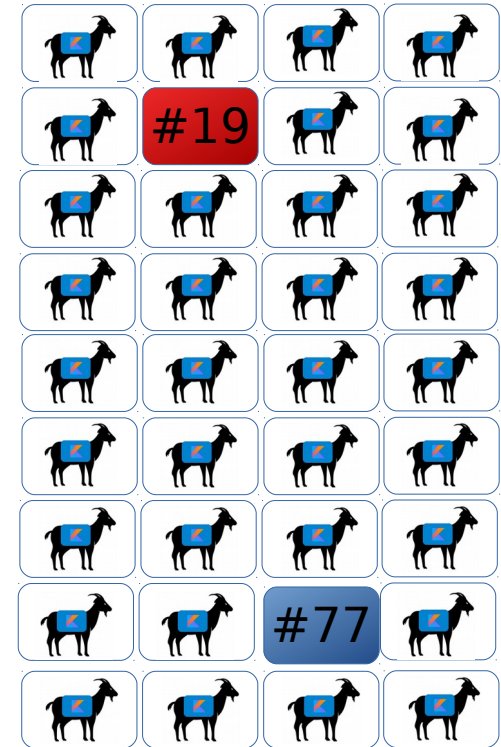
The Monty Hall Problem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(P_{19}|D_{77}) = \frac{P(D_{77}|P_{19})P(P_{19})}{P(D_{77})} = \frac{\frac{1}{999} * \frac{1}{1000}}{\frac{1}{1000}} = \frac{1}{999}$$

$$P(P_{77}|D_{77}) = \frac{P(D_{77}|P_{77})P(P_{77})}{P(D_{77})} = \frac{\frac{999}{1000} * \frac{1}{1000}}{\frac{1}{1000}} = \frac{999}{1000}$$

Yes, you should switch!



Monty Hall Simulation in Kotlin

Monte Carlo simulation of the Monty Hall Problem

<https://gist.github.com/thomasniel/7fe76d27a57afbea49939dc1879c9883>

Why Am I Showing This?

The Monty Hall problem encapsulates the critical (and misunderstood) nuances of probability.

Sometimes we only have incomplete data, but we need to make a decision anyway.

When new partial data is available, we must merge (not abandon) the data we previously had.

As programmers, we thrive in certainty and exactness.

But the valuable, high-profile problems today often tackle uncertainty and approximation.

Users and businesses expect “smarter” applications that will predict what they want.

Machine learning and optimization is nondeterministic and unavoidably has error.

Many search spaces are too large to exhaustively explore.

Part I: Why Learn Mathematical Modeling?



What is Mathematical Modeling?

Mathematical modeling is a broad discipline that attempts to solve real-world problems using mathematical concepts.

Applications range broadly, from *biology* and *medicine* to *engineering*, *business*, and *economics*.

Mathematical Modeling is used heavily in optimization, machine learning, and data science.

There is no “general artificial intelligence”, but rather the right technique for the right type of problem.

Real-World Examples

Product recommendations

Staff/resource scheduling

Airline network optimizer

Dynamic pricing

Inventory forecaster

DNA sequencing

Sport event planning

Kidney exchanges

Disaster Management

Why Learn Mathematical Modeling?

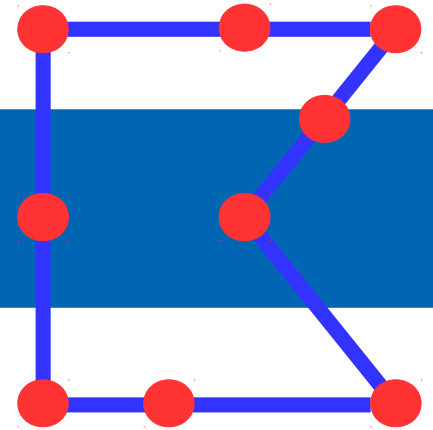
Technologies, frameworks, and languages come and go... but math never changes.

Classic mathematical techniques have survived decades (even centuries) and are not prone to obsolescence and deprecation.

Knowing mathematical modeling will greatly improve your professional marketability for decades to come, without constant investment in learning new technologies.

Math does not get deprecated.

Mathematical Modeling Part I: Discrete Optimization



What Is Discrete Optimization?

Discrete optimization is a space of algorithms that tries to find a feasible or optimal solution to a constrained problem.

Scheduling classrooms, staff, transportation, sports teams, and manufacturing

Finding an optimal route for vehicles to visit multiple destinations

Optimizing manufacturing operations

Solving a Sudoku

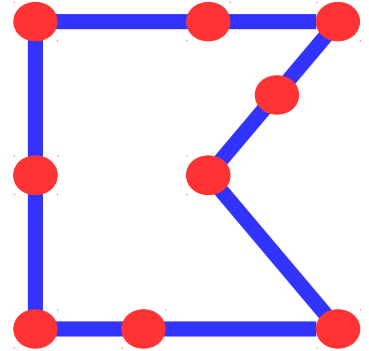
Discrete optimization is a mixed bag of algorithms and techniques, which can be built from scratch or with the assistance of a library.

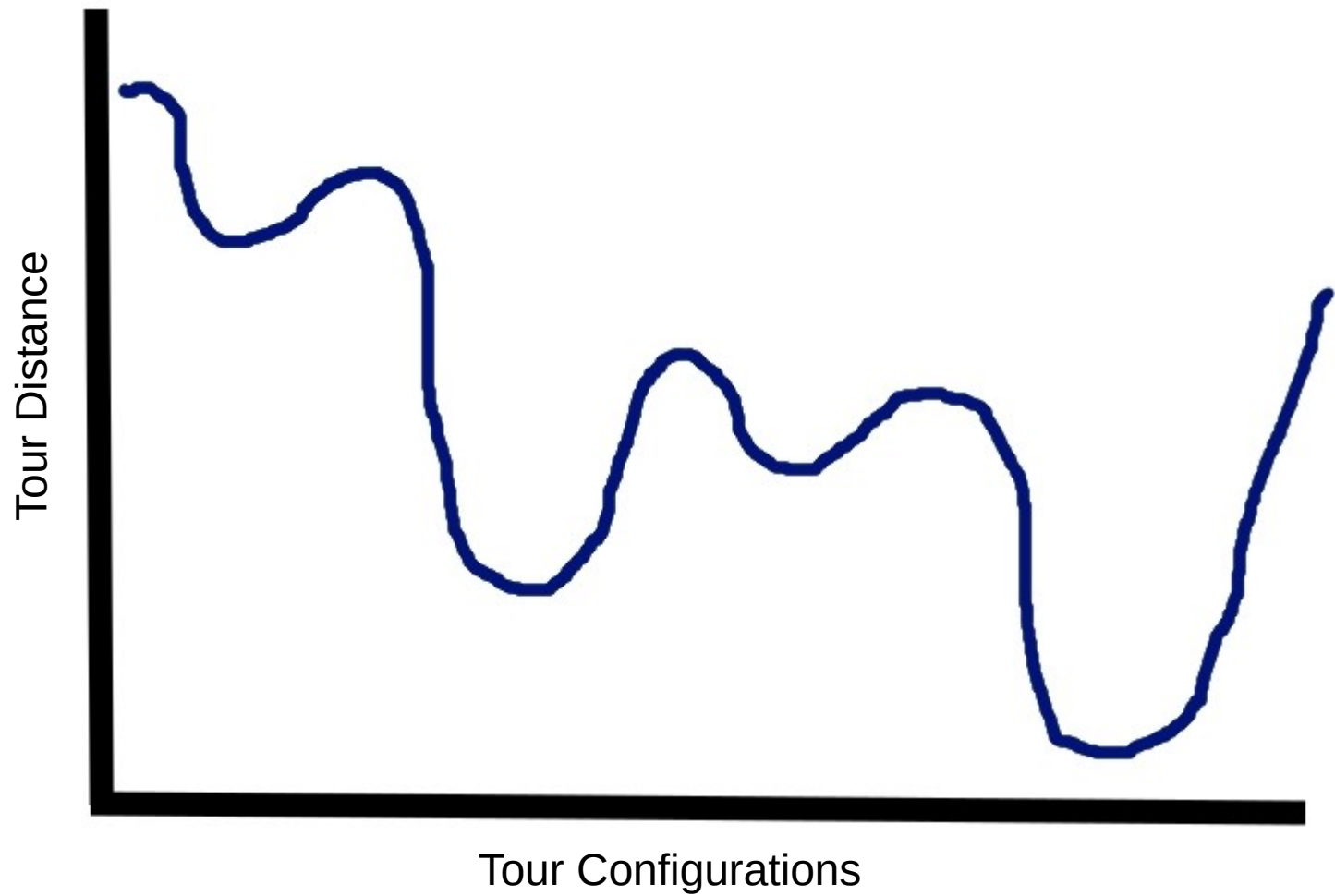
Traveling Salesman Problem

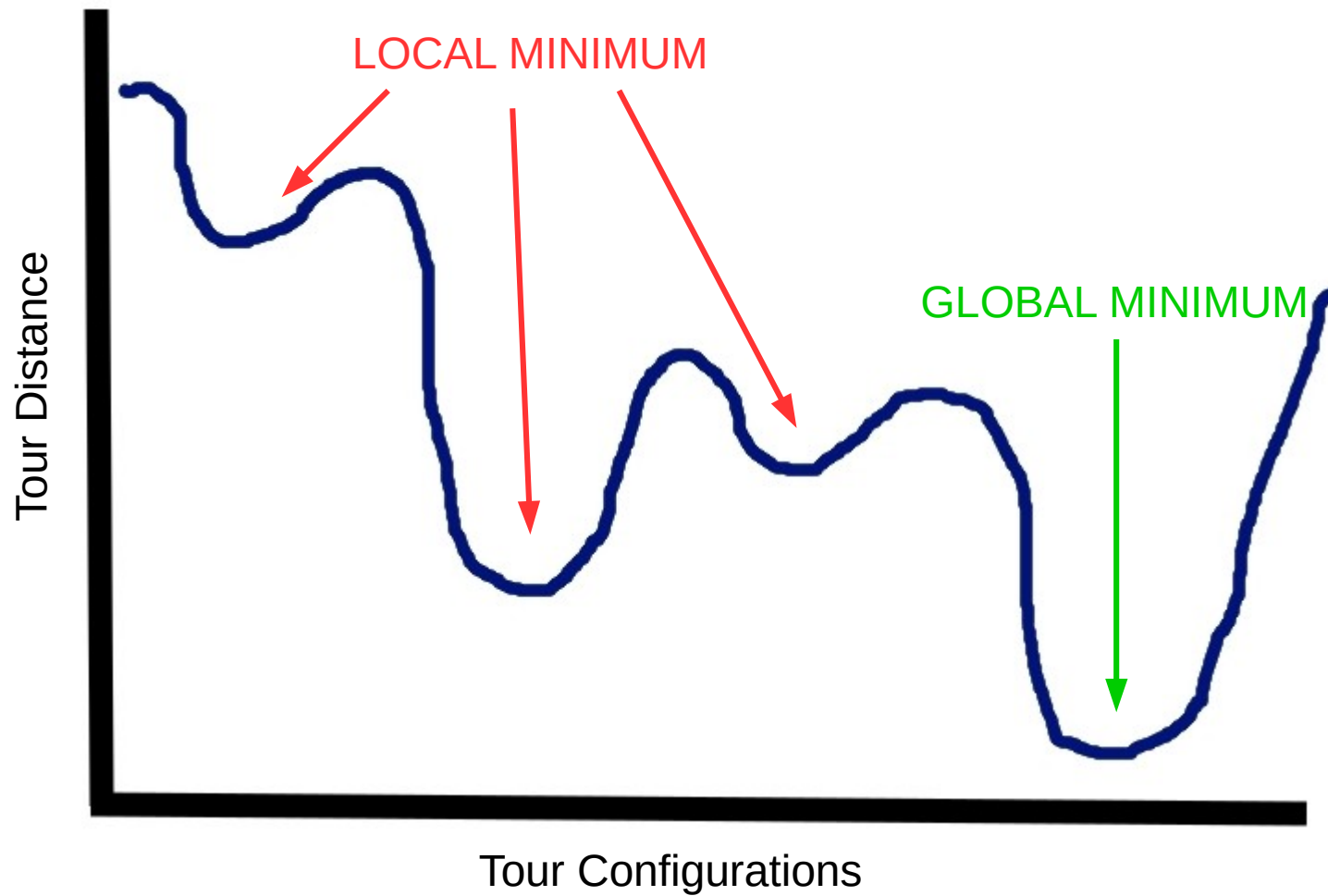
The Traveling Salesman Problem (TSP) is one of the most elusive and studied computer science problems since the 1950's.

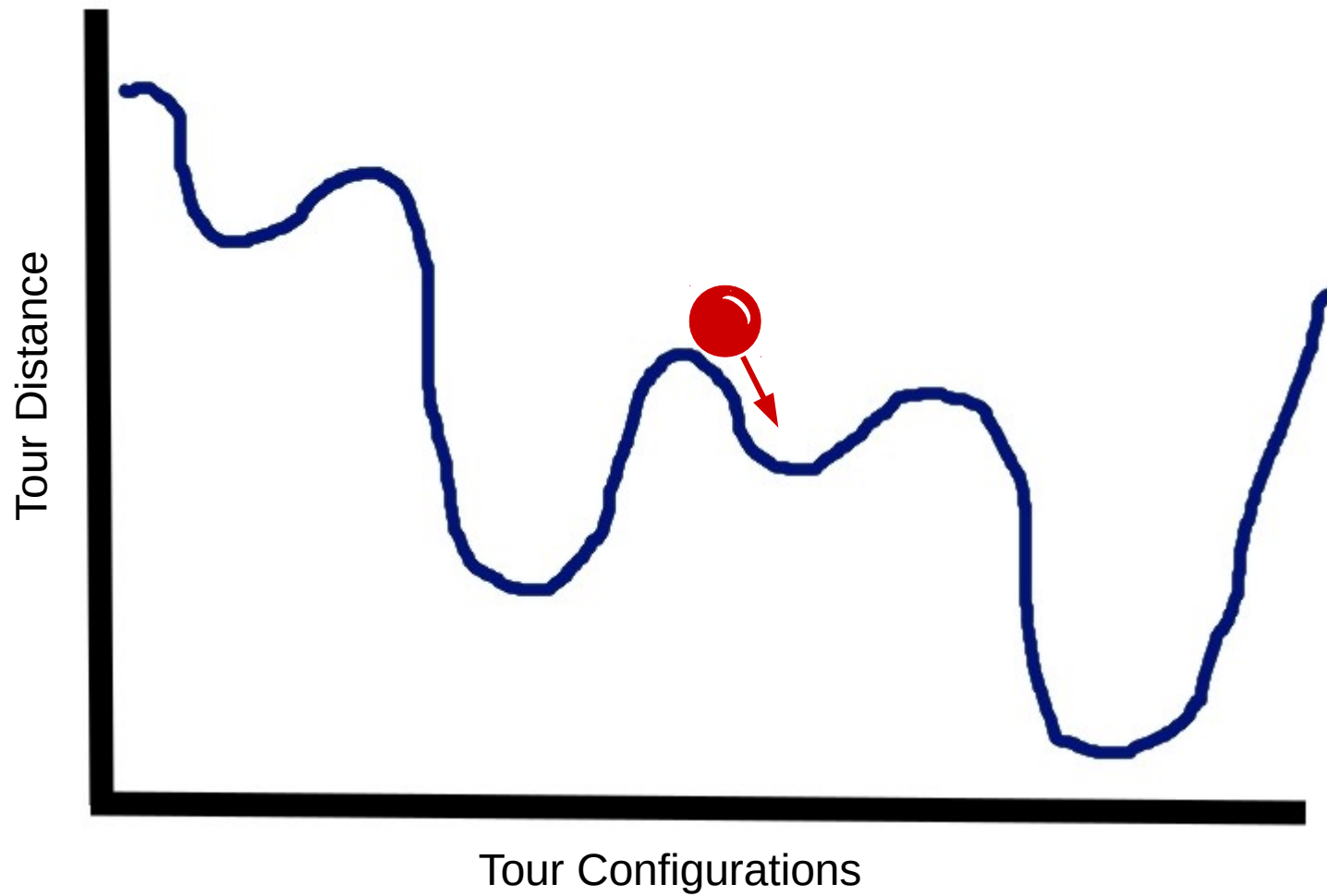
Objective: Find the shortest round-trip tour across several geographic points/cities.

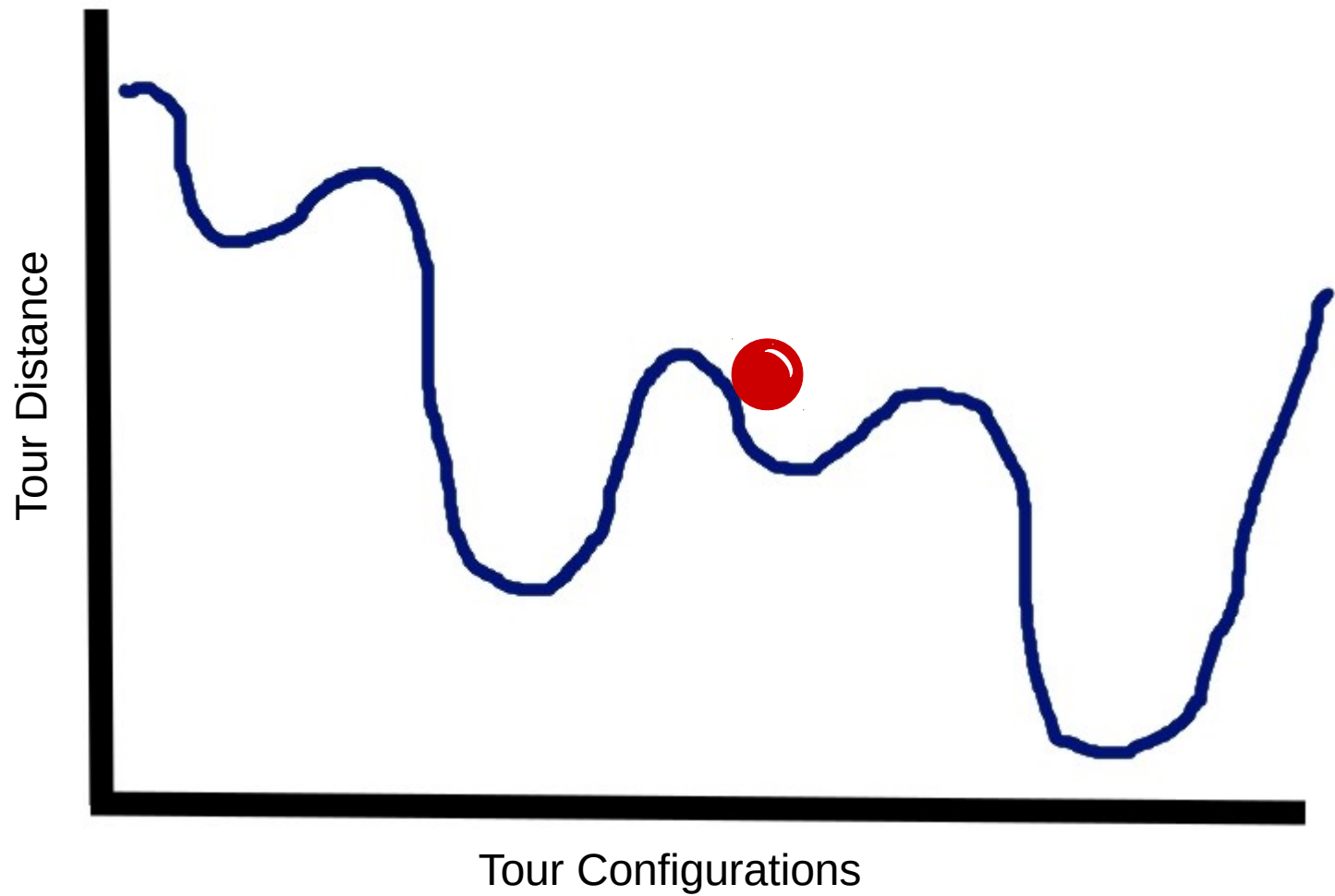
The Challenge: Just **60** cities = **8.3×10^{81}** possible tours
That's more tour combinations than there are observable atoms in the universe!

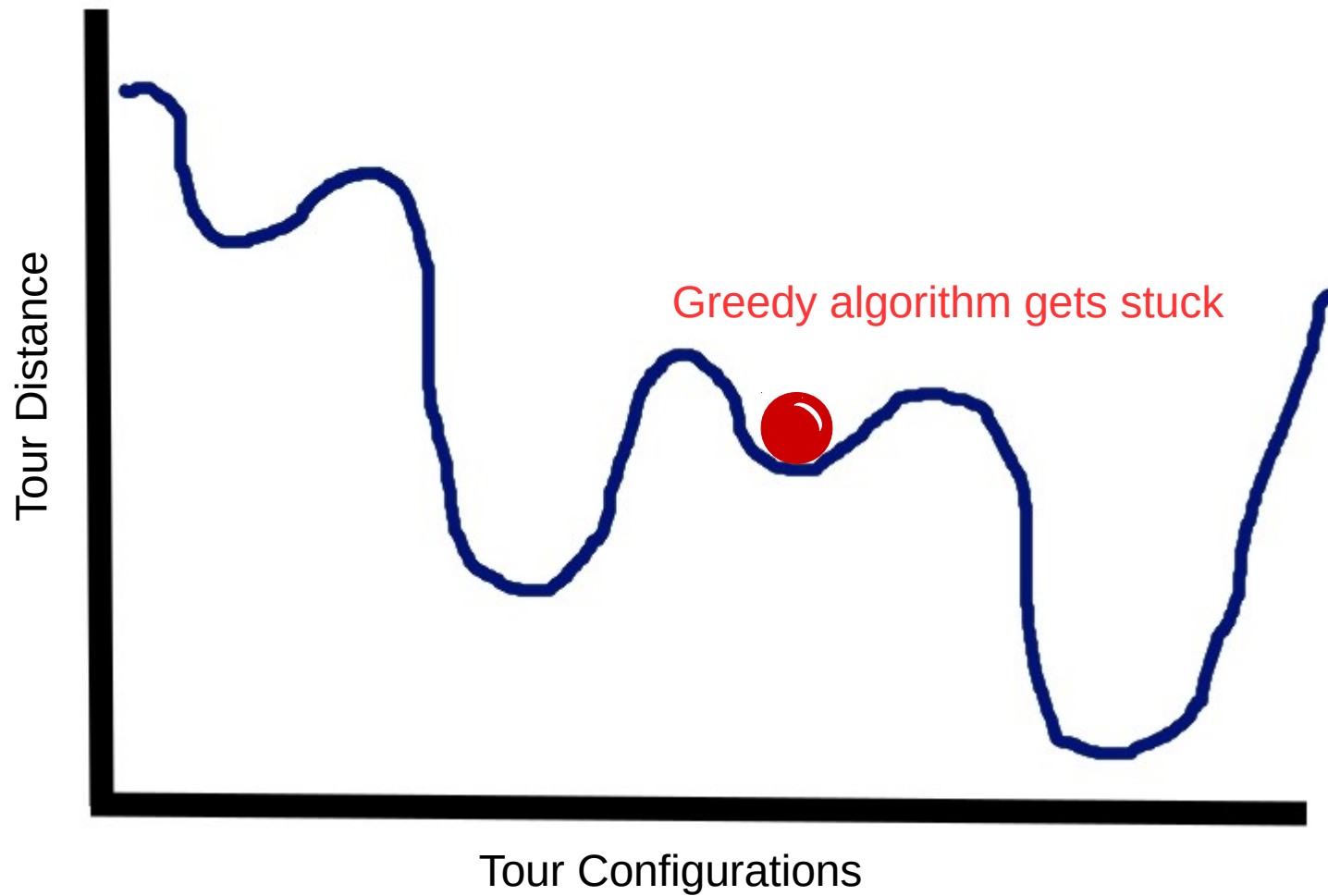


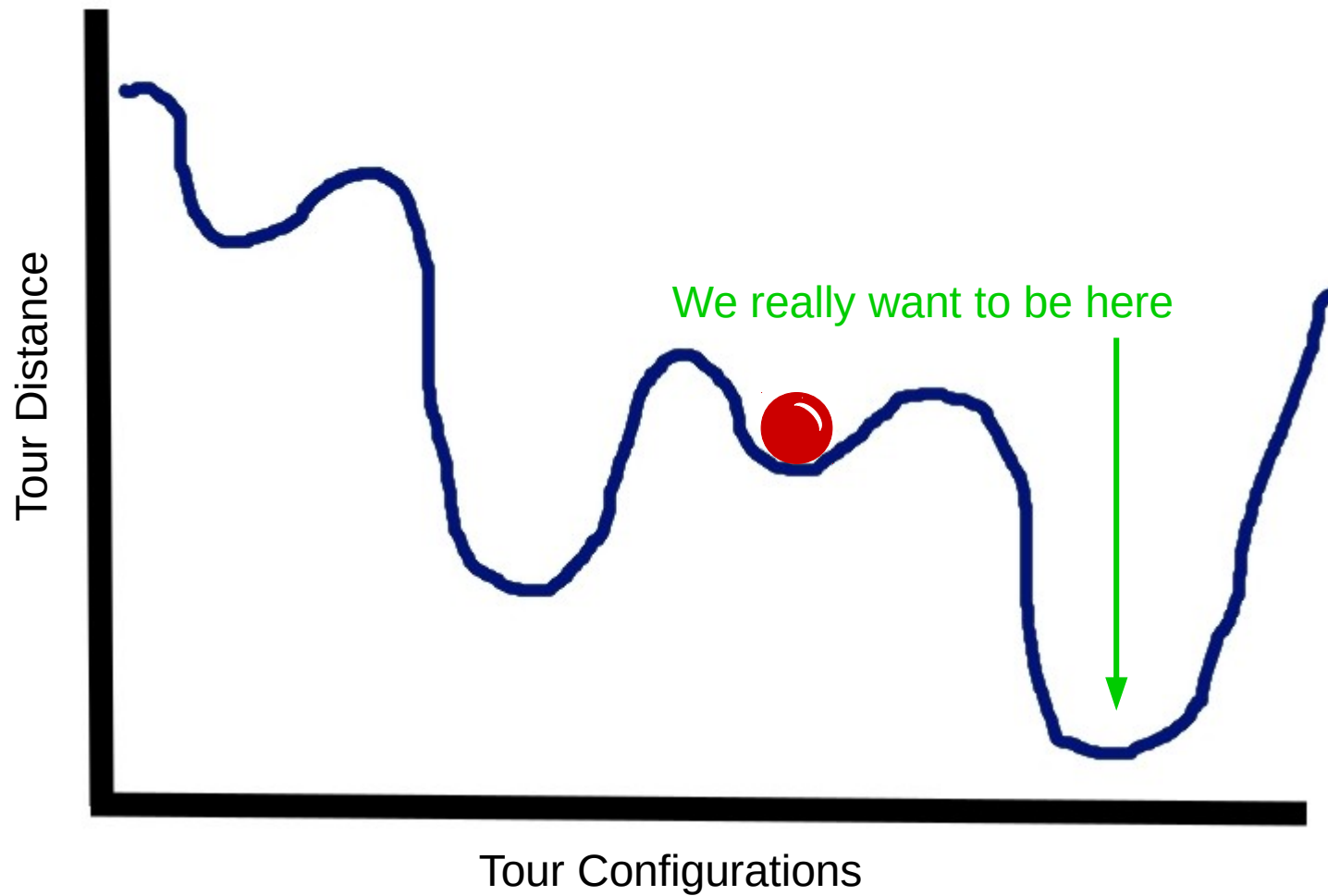


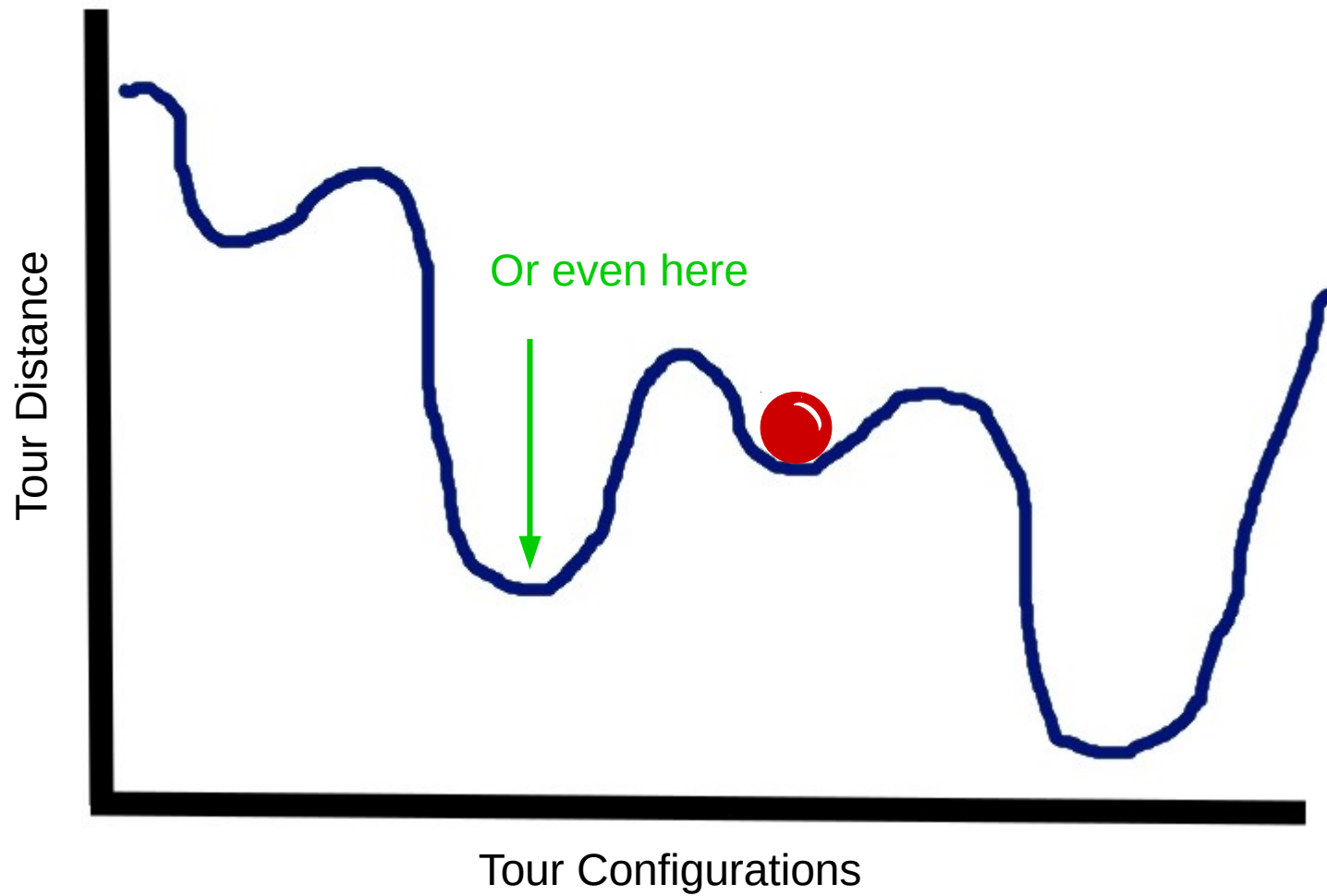


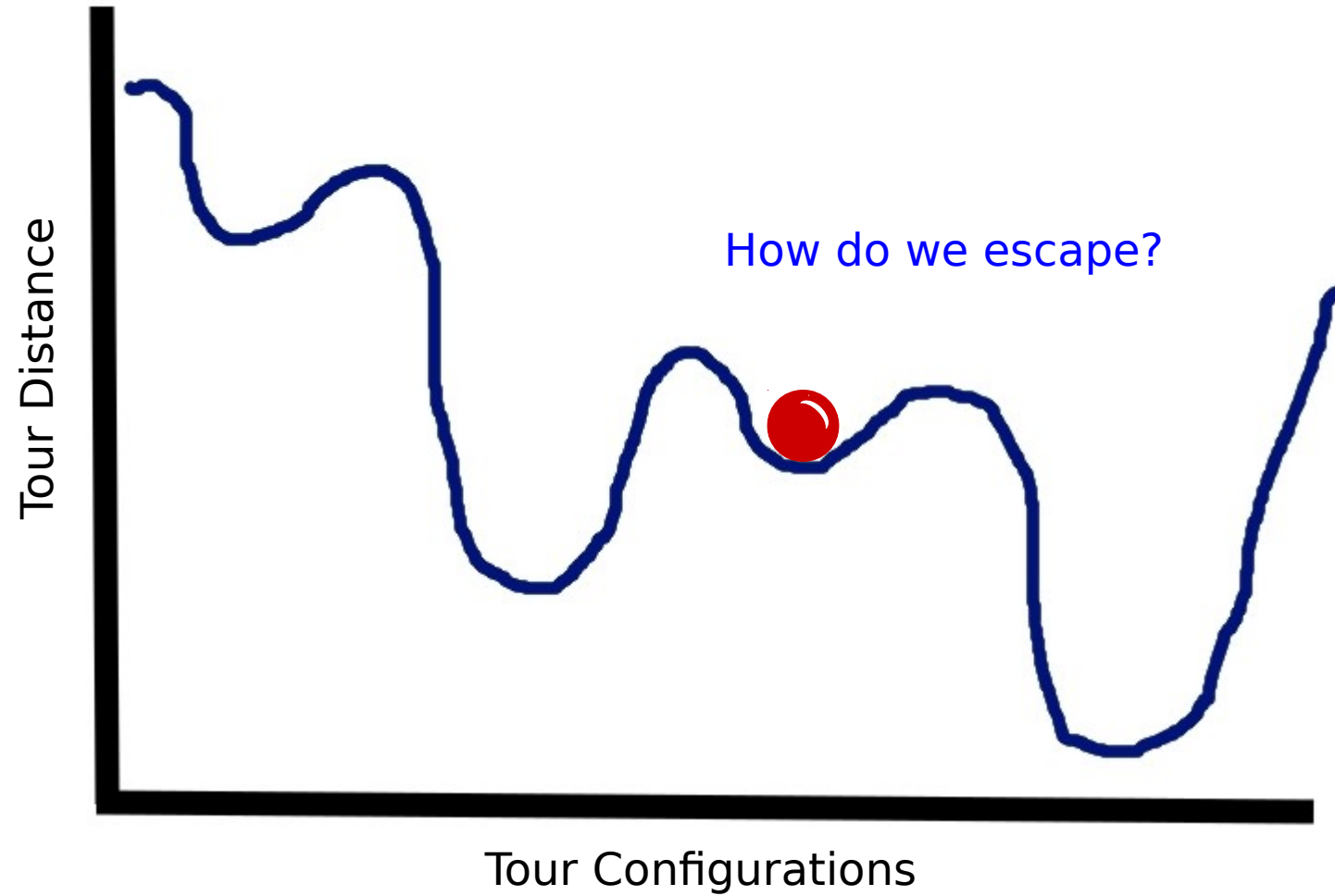


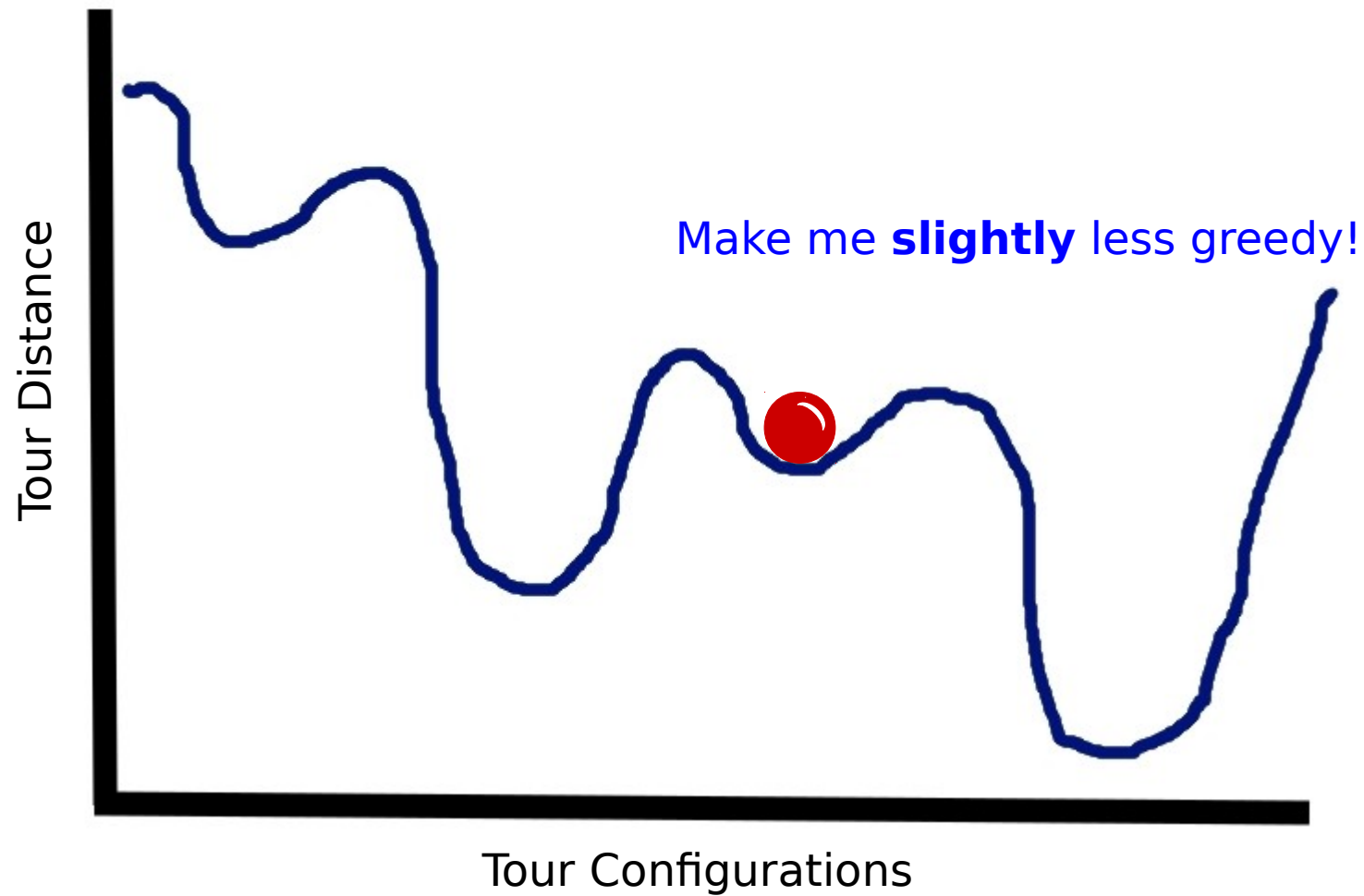


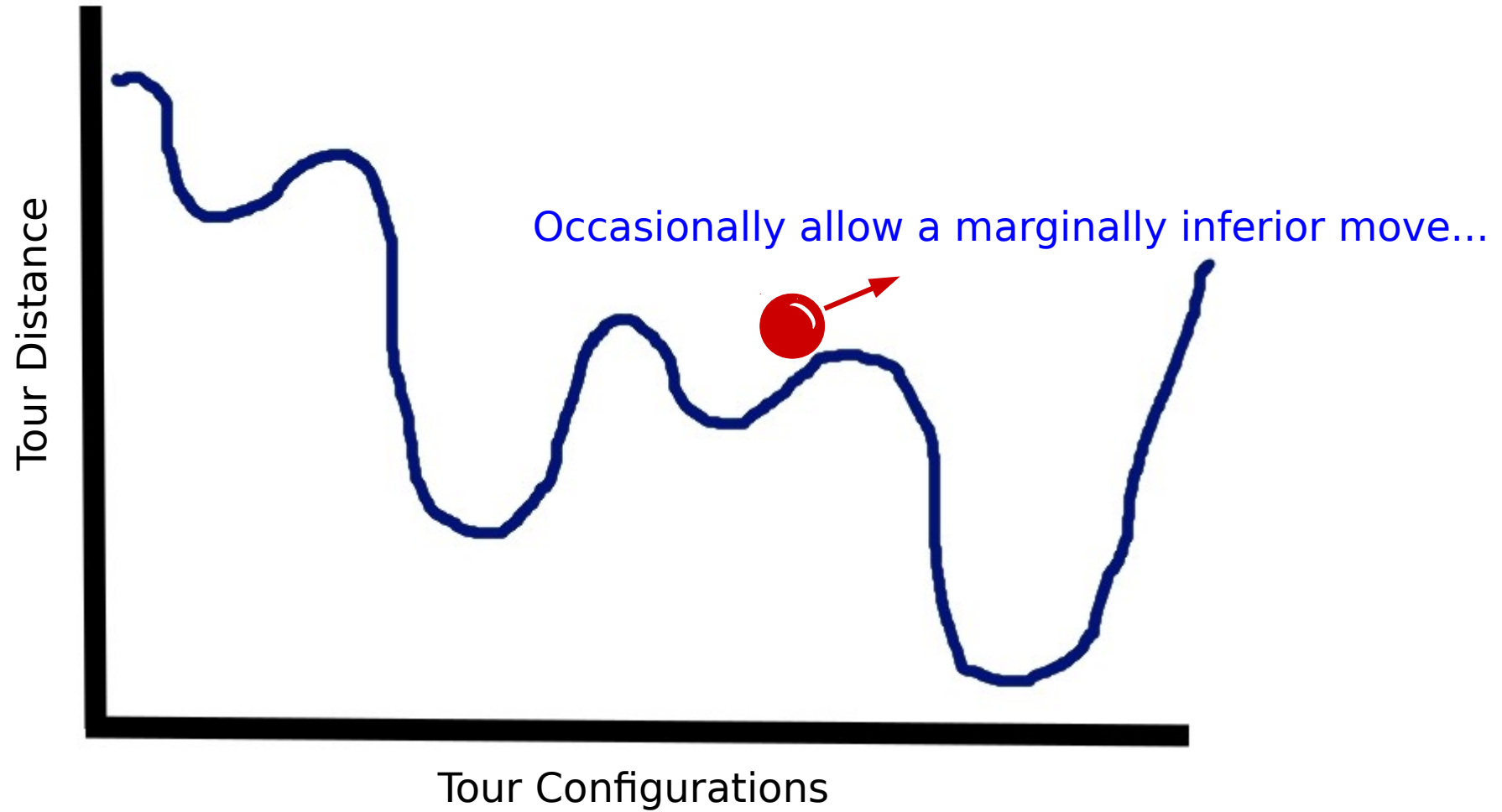


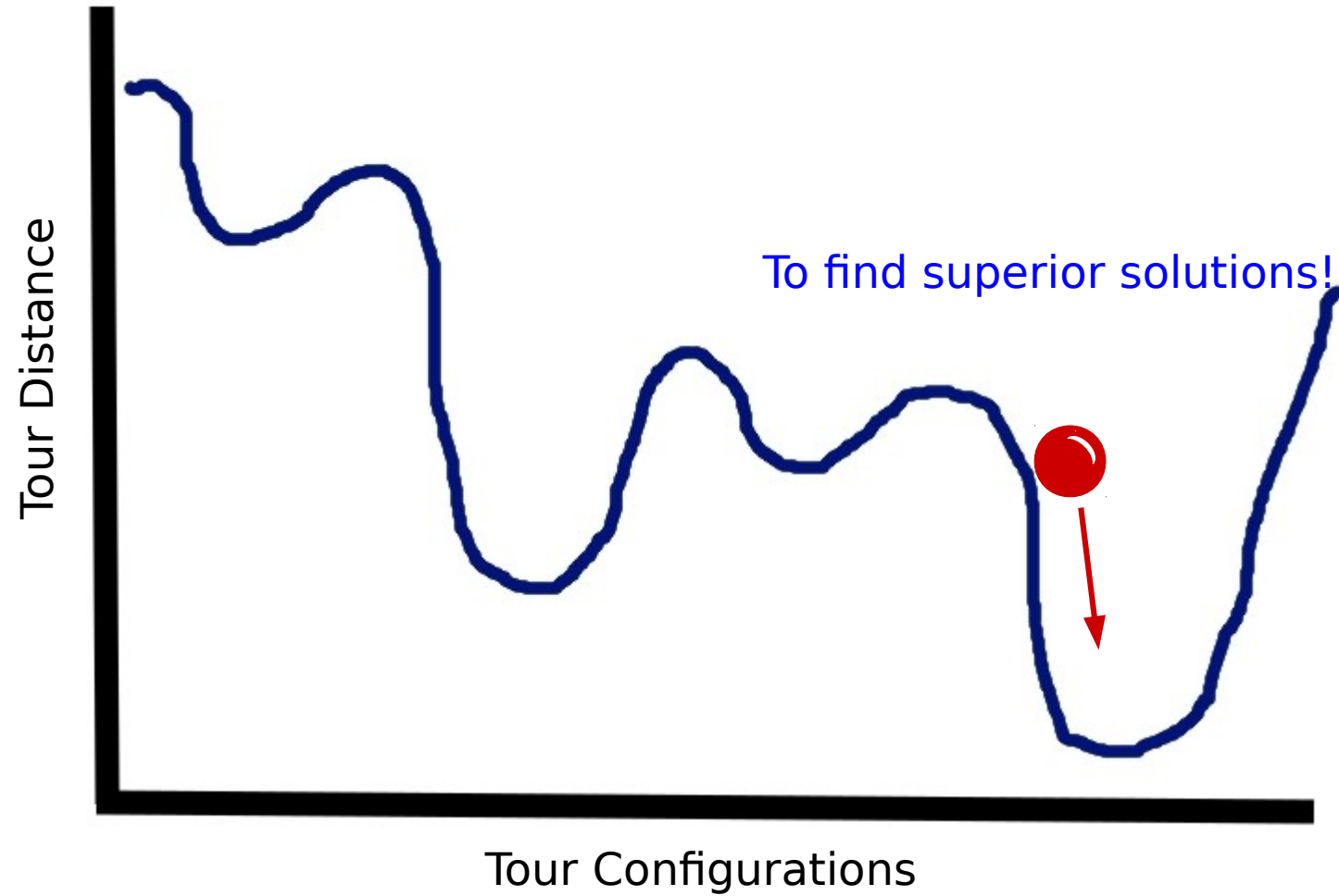


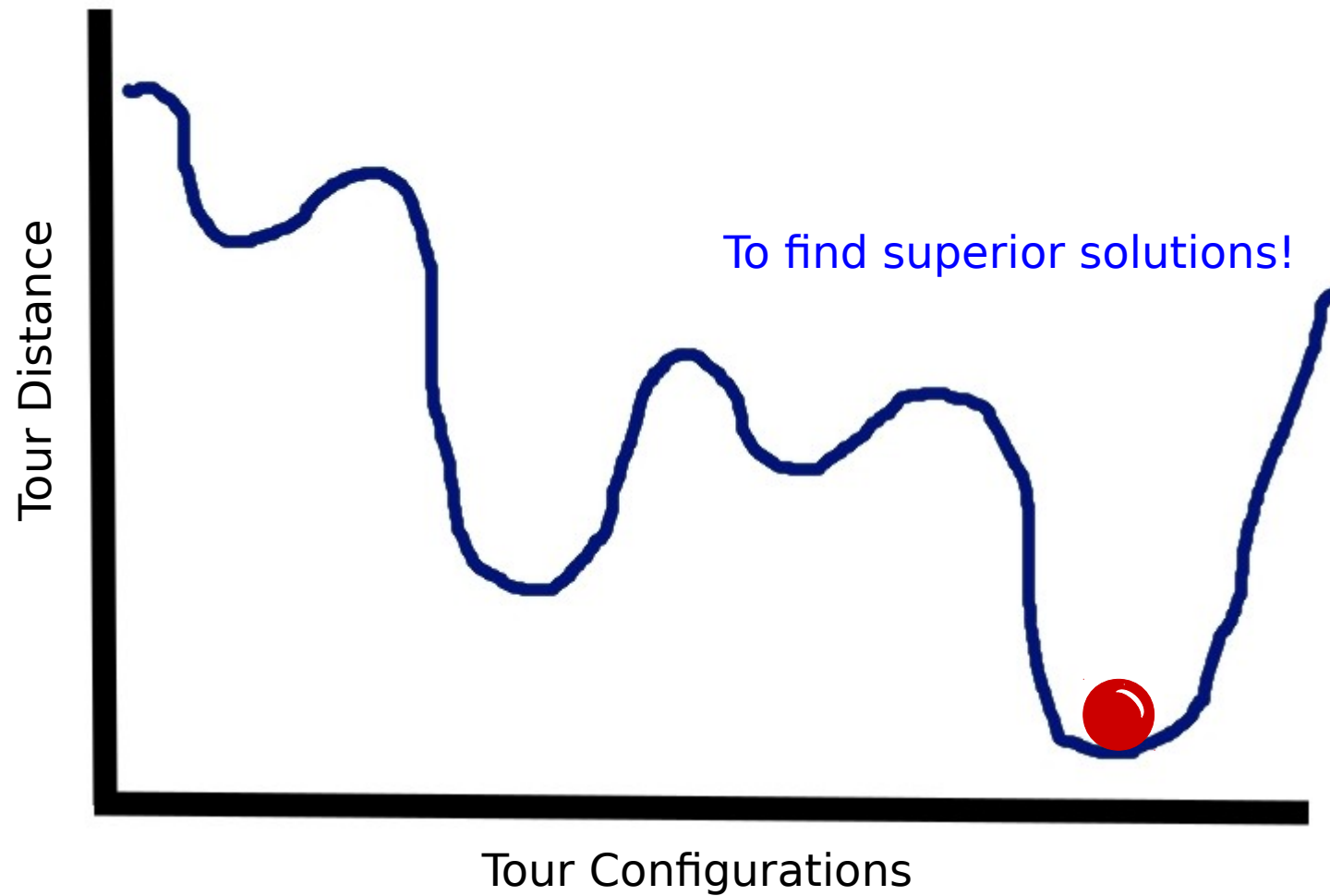












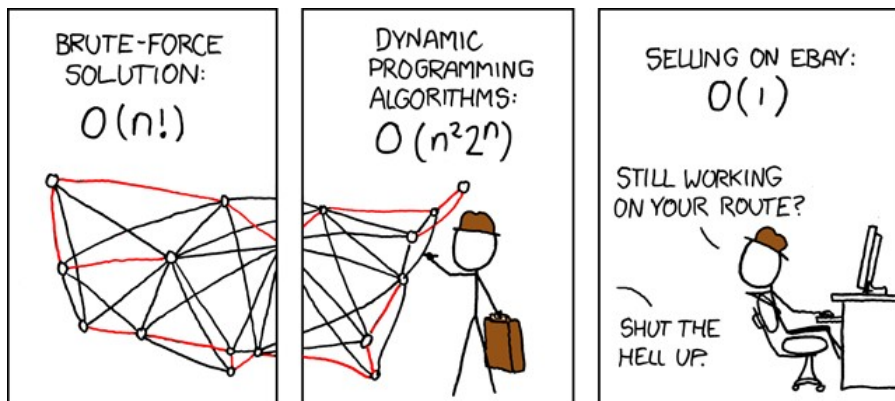
Source Code

Traveling Salesman Demo

https://github.com/thomasniel/traveling_salesman_demo

Traveling Salesman Plotter

https://github.com/thomasniel/traveling_salesman_plotter



SOURCE: xkcd.com

Generating a Schedule

You need to generate a schedule for a single classroom with the following classes:

Psych 101 (1 hour, 2 sessions/week)
English 101 (1.5 hours, 2 sessions/week)
Math 300 (1.5 hours, 2 sessions/week)
Psych 300 (3 hours, 1 session/week)
Calculus I (2 hours, 2 sessions/week)
Linear Algebra I (2 hours, 3 sessions/week)
Sociology 101 (1 hour, 2 sessions/week)
Biology 101 (1 hour, 2 sessions/week)
Supply Chain 300 (2.5 hours, 2 sessions/week)
Orientation 101 (1 hour, 1 session/week)

Available scheduling times are Monday through Friday, 8:00AM-11:30AM, 1:00PM-5:00PM
Slots are scheduled in 15 minute increments.

Generating a Schedule

Visualize a grid of each 15-minute increment from Monday through Sunday, intersected with each possible class.

Each cell will be a 1 or 0 indicating whether that's the start of the first class.

	MON	MON	MON	MON	MON	MON	MON	MON		SUN
	12:00 AM	12:15 AM	12:30 AM	12:45 AM	1:00 AM	1:15 AM	1:30 AM	1:45 AM	...	11:55 PM
Psych 101	0	0	0	0	0	0	0	0	...	0
English 101	0	0	0	0	0	0	0	0	...	0
Math 300	0	0	0	0	0	0	0	0	...	0
Psych 300	0	0	0	0	0	0	0	0	...	0
Calculus I	0	0	0	0	0	0	0	0	...	0
Linear Algebra I	0	0	0	0	0	0	0	0	...	0
Sociology 101	0	0	0	0	0	0	0	0	...	0
Biology 101	0	0	0	0	0	0	0	0	...	0
Supply Chain 300	0	0	0	0	0	0	0	0	...	0
Orientation 101	0	0	0	0	0	0	0	0	...	0

Generating a Schedule

Next visualize how overlaps will occur.

Notice how a 9:00AM Psych 101 class will clash with a 9:15AM Sociology 101.

We can sum all blocks that affect the 9:45AM block and ensure they don't exceed 1.

		MON	MON	MON	MON	MON	MON	MON	MON		SUN
	...	9:00 AM	9:15 AM	9:30 AM	9:45 AM	10:00 AM	10:15 AM	10:30 AM	10:45 AM	...	11:55 PM
Psych 101	...	1	0	0	0	0	0	0	0	...	0
English 101	...	0	0	0	0	0	0	0	0	...	0
Math 300	...	0	0	0	0	0	0	0	0	...	0
Psych 300	...	0	0	0	0	0	0	0	0	...	0
Calculus I	...	0	0	0	0	0	0	0	0	...	0
Linear Algebra I	...	0	0	0	0	0	0	0	0	...	0
Sociology 101	...	0	1	0	0	0	0	0	0	...	0
Biology 101	...	0	0	0	0	0	0	0	0	...	0
Supply Chain 300	...	0	0	0	0	0	0	0	0	...	0
Orientation 101	...	0	0	0	0	0	0	0	0	...	0

Sum of affecting slots = 2
FAIL, sum must be ≤ 1

Generating a Schedule

Next visualize how overlaps will occur.

Notice how a 9:00AM Psych 101 class will clash with a 9:30AM Sociology 101.

We can sum all blocks that affect the 9:45AM block and ensure they don't exceed 1.

		MON	MON	MON	MON	MON	MON	MON	MON		SUN
	...	9:00 AM	9:15 AM	9:30 AM	9:45 AM	10:00 AM	10:15 AM	10:30 AM	10:45 AM	...	11:55 PM
Psych 101	...	1	0	0	0	0	0	0	0	...	0
English 101	...	0	0	0	0	0	0	0	0	...	0
Math 300	...	0	0	0	0	0	0	0	0	...	0
Psych 300	...	0	0	0	0	0	0	0	0	...	0
Calculus I	...	0	0	0	0	0	0	0	0	...	0
Linear Algebra I	...	0	0	0	0	0	0	0	0	...	0
Sociology 101	...	0	0	1	0	0	0	0	0	...	0
Biology 101	...	0	0	0	0	0	0	0	0	...	0
Supply Chain 300	...	0	0	0	0	0	0	0	0	...	0
Orientation 101	...	0	0	0	0	0	0	0	0	...	0

Sum of affecting slots = 2
FAIL, sum must be ≤ 1

Generating a Schedule

Next visualize how overlaps will occur.

Notice how a 9:00AM Psych 101 class will clash with a 9:45AM Sociology 101.

We can sum all blocks that affect the 9:45AM block and ensure they don't exceed 1.

		MON	MON	MON	MON	MON	MON	MON	MON		SUN
	...	9:00 AM	9:15 AM	9:30 AM	9:45 AM	10:00 AM	10:15 AM	10:30 AM	10:45 AM	...	11:55 PM
Psych 101	...	1	0	0	0	0	0	0	0	...	0
English 101	...	0	0	0	0	0	0	0	0	...	0
Math 300	...	0	0	0	0	0	0	0	0	...	0
Psych 300	...	0	0	0	0	0	0	0	0	...	0
Calculus I	...	0	0	0	0	0	0	0	0	...	0
Linear Algebra I	...	0	0	0	0	0	0	0	0	...	0
Sociology 101	...	0	0	0	1	0	0	0	0	...	0
Biology 101	...	0	0	0	0	0	0	0	0	...	0
Supply Chain 300	...	0	0	0	0	0	0	0	0	...	0
Orientation 101	...	0	0	0	0	0	0	0	0	...	0

Sum of affecting slots = 2
FAIL, sum must be ≤ 1

Generating a Schedule

If the “sum” of all slots affecting a given block are no more than 1, then we have no conflicts!

		MON	MON	MON	MON	MON	MON	MON	MON		SUN
	...	9:00 AM	9:15 AM	9:30 AM	9:45 AM	10:00 AM	10:15 AM	10:30 AM	10:45 AM	...	11:55 PM
Psych 101	...	1	0	0	0	0	0	0	0	...	0
English 101	...	0	0	0	0	0	0	0	0	...	0
Math 300	...	0	0	0	0	0	0	0	0	...	0
Psych 300	...	0	0	0	0	0	0	0	0	...	0
Calculus I	...	0	0	0	0	0	0	0	0	...	0
Linear Algebra I	...	0	0	0	0	0	0	0	0	...	0
Sociology 101	...	0	0	0	0	1	0	0	0	...	0
Biology 101	...	0	0	0	0	0	0	0	0	...	0
Supply Chain 300	...	0	0	0	0	0	0	0	0	...	0
Orientation 101	...	0	0	0	0	0	0	0	0	...	0

Sum of affecting slots = 1
SUCCESS!

Generating a Schedule

For every “block”, we must sum all affecting slots (shaded below) which can be identified from the class durations.

This sum must be no more than 1.

		MON	MON	MON	MON	MON	MON	MON	MON	MON	MON	MON	MON	MON		SUN
	...	7:00 AM	7:15 AM	7:30 AM	7:45 AM	8:00 AM	8:15 AM	8:30 AM	8:45 AM	9:00 AM	9:15 AM	9:30 AM	9:45 AM	10:00 AM	...	11:55 PM
Psych 101	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
English 101	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Math 300	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Psych 300	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Calculus I	...	0	0	0	0	1	0	0	0	0	0	0	0	0	...	0
Linear Algebra I	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Sociology 101	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Biology 101	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Supply Chain 300	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
Orientation 101	...	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0

Generating a Schedule

Taking this concept even further, we can account for all recurrences.

The “affected slots” for a given block can query for all recurrences for each given class.

[View image here.](#)

Generating a Schedule

Plug these variables and *feasible* constraints into the optimizer or a tree search algorithm (which I'll show next), and you will get a solution.

Most of the work will be finding the affecting slots for each block.



Hold that Thought, Let's Talk About State Space Search

Imagine you are presented a Sudoku.

Rather than do an exhaustive brute-force search, think in terms of constraint programming to reduce the search space.

First, sort the cells by the count of possible values they have left:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Solving a Sudoku

[4,4] → 5

[2,6] → 7

[7,7] → 3

[8,6] → 4

[1,4] → 2, 5

[0,7] → 2, 3

[3,2] → 2, 3

[4,2] → 3, 4

[5,2] → 2, 4

[3,5] → 5, 9

[5,5] → 1, 4

[4,6] → 3, 5

[5,8] → 2, 6

[6,7] → 3, 6

[0,2] → 1, 2, 3

[1,3] → 1, 2, 5

...

[2,6] → 1, 3, 4, 5, 7, 9

Put cells in a list
sorted by possible
candidate count

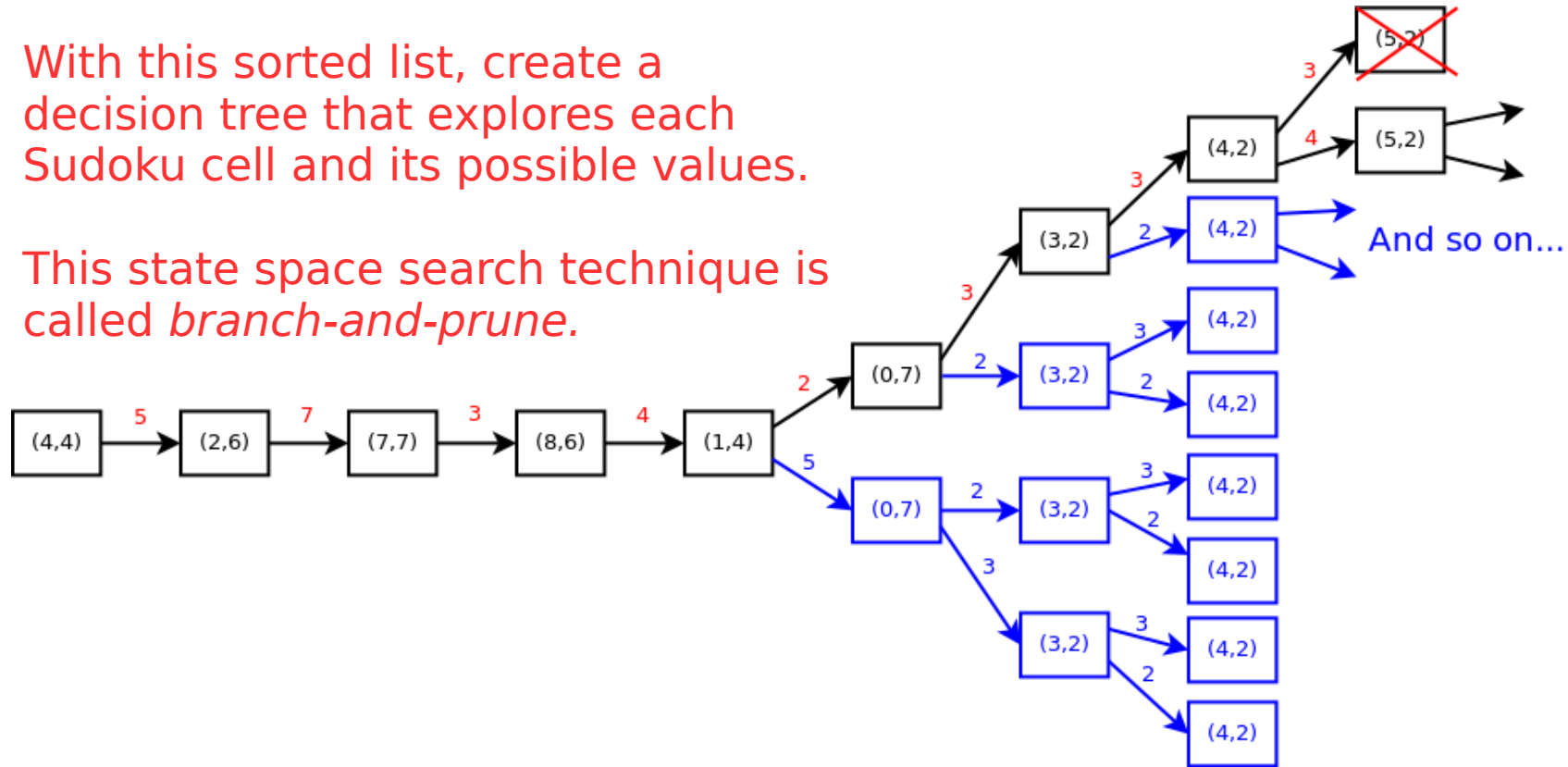
	0	1	2	3	4	5	6	7	8
0	5	3			7				
1	6			1	9	5			
2		9	8					6	
3	8				6				3
4	4			8		3			1
5	7				2				6
6		6					2	8	
7				4	1	9			5
8					8			7	9

Solving a Sudoku

[4,4] → 5
[2,6] → 7
[7,7] → 3
[8,6] → 4
[1,4] → 2, 5
[0,7] → 2, 3
[3,2] → 2, 3
[4,2] → 3, 4
[5,2] → 2, 4
[3,5] → 5, 9
[5,5] → 1, 4
[4,6] → 3, 5
[5,8] → 2, 6
[6,7] → 3, 6
[0,2] → 1, 2, 3
[1,3] → 1, 2, 5
...
[2,6] → 1, 3, 4, 5, 7, 9

With this sorted list, create a decision tree that explores each Sudoku cell and its possible values.

This state space search technique is called *branch-and-prune*.

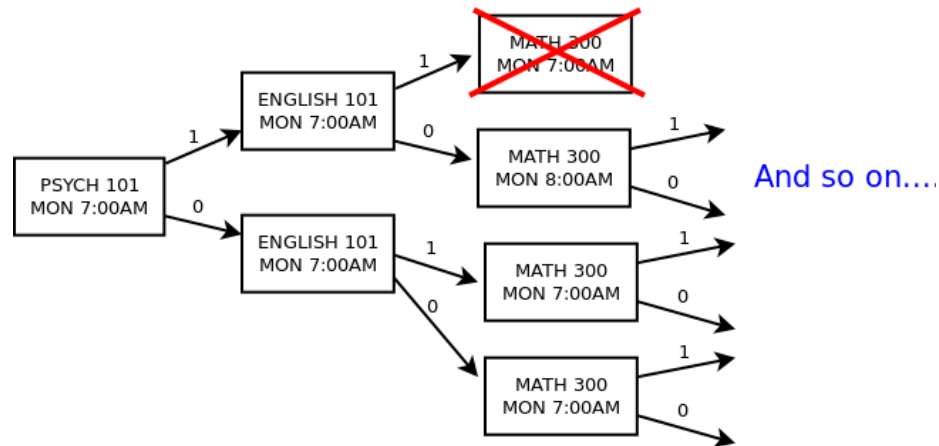


Branch-and-Prune for Scheduling

You could solve the scheduling problem from scratch with branch-and-prune.

Start with the most “constrained” slots first to narrow your search space (e.g. slots fixed to zero first, followed by Monday slots for 3-recurrence classes).

HINT: Proactively prune the tree as you go, eliminating any slots ahead that must be zero due to a “1” decision propagating an occupied state.



Generating a Schedule

If you want to schedule against multiple rooms, plot each variable using three dimensions.

	MON 8:00	MON 8:15	MON 8:30	MON 8:45	MON 9:00	MON 9:15	MON 9:30	MON 9:45
PSYCH 300					1	0	0	0
MATH 300								
PSYCH 101	1	0	0	0				
ROOM 1					1	0	0	0
ROOM 2	1	0	0	0				
ROOM 3								

Continuous Labor Shifts

You have three drivers who charge the following rates:

Driver 1: \$10 / hr

Driver 2: \$12 / hr

Driver 3: \$15 / hr

From 6:00 to 22:00, schedule one driver at a time to provide coverage, and minimize cost.

Each driver must work 4-6 hours a day. Driver 2 cannot work after 11:00.

Stay Calm

Variables and Constants

S_i = Shift start time of each i driver

E_i = Shift end time for each i driver

R_i = Hourly rate for each i driver

δ_{ij} = Binary (1,0) between two ij drivers

M = Length of planning window

Minimize

$$\sum_{i=1}^3 R_i (E_i - S_i)$$

Constraints

$$4 \leq E_i - S_i \leq 6$$

$$16 = \sum_{i=1}^3 E_i - S_i$$

$$E_2 \leq 11$$

$$S_i \geq E_j - M \delta_{ij}$$

$$S_j \geq E_i - M (1 - \delta_{ij})$$

Source Code

Continuous Scheduling Example

<https://github.com/thomasnield/continuous-optimization-example>

Discrete Optimization Summary

Discrete Optimization is a best-kept secret well-known in operations research.

Machine learning itself is an optimization problem, finding the right values for variables to minimize an error function.

Many folks misguidedly think of neural networks and other machine learning when discrete optimization would be more appropriate.

Recommended Java Libraries:

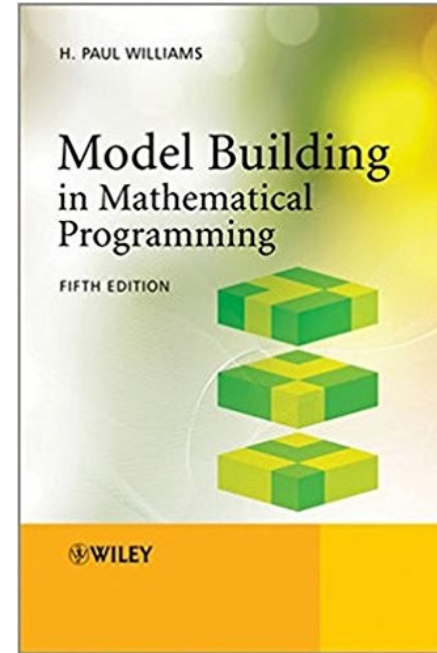
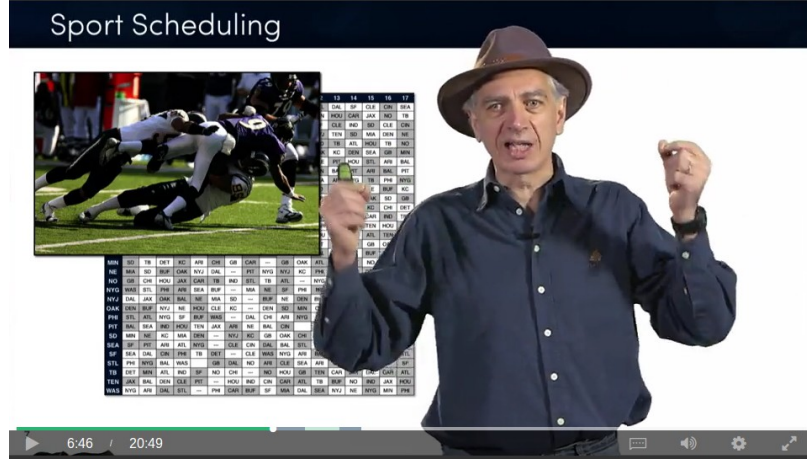
OjAlgo!

OptaPlanner

Learn More About Discrete Optimization

coursera *Discrete Optimization*

Sport Scheduling



Fun with Random Numbers

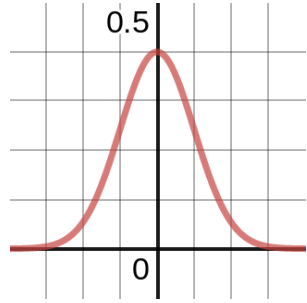
You may be familiar with random number generators

When any number is equally likely to be generated, this is known as a *Uniform* distribution

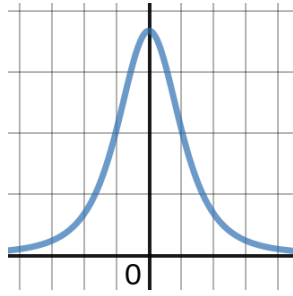
But what if we wanted some numbers to be generated more likely than others?

This can be a powerful tool and can be used to, for instance, fit a line to points. It can also be used to create simulations.

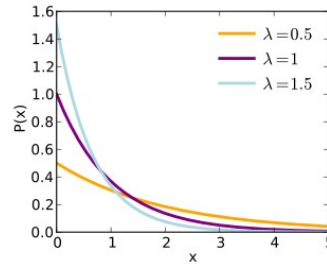
Some Useful Distributions



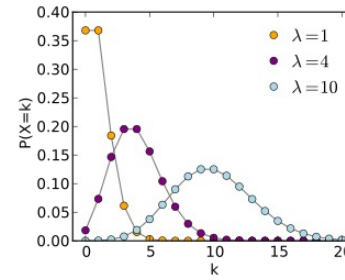
Normal



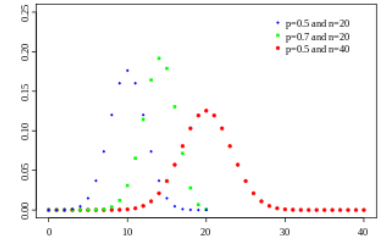
T-Distribution



Exponential



Poisson



Binomial

Mathematical Modeling Part II: Classification w/ Naive Bayes



Classifying Things

Probably the most common task in machine learning is classifying data:

How do I identify images of **dogs** vs **cats**?

What **words** are being said in a piece of audio?

Is this email **spam** or **not spam**?

What attributes define **high-risk**, **medium-risk**, and **low-risk** loan applicants?

How do I predict if a shipment will be **late**, **early**, or **on-time**?

There are many techniques to classify data, with pros/cons depending on the task:

Neural Networks

Support Vector Machines

Decision Trees/Random Forests

Naive Bayes

Linear/Non-linear regression

Naive Bayes

Let's focus on Naive Bayes because it is simple to implement and effective for a common task: *text categorization*.

Naive Bayes is an adaptation of Bayes Theorem that can predict a category **C** for an item **T** with *multiple* features **F**.

A common usage example of Naive Bayes is email spam, where each word is a feature and **spam/not spam** are the possible categories.

Implementing Naive Bayes

Naive Bayes works by mapping probabilities of each individual feature occurring/not occurring for a given category (e.g. a word occurring in **spam/not spam**).

A category can be predicted for a new set of features by...

- 1) For a given category, combine the probabilities of each feature **occurring** and **not occurring** by multiplying them.

$$\text{Occur Product} = P_{f1} * P_{f2} * \dots * P_{fn}$$

$$\text{Not Occur Product} = !P_{f1} * !P_{f2} * \dots * !P_{fn}$$

- 2) Divide the products to get the probability for that category.

$$\text{Combined Probability} = \frac{(\text{Occur Product})}{(\text{Occur Product}) + (\text{Not Occur Product})}$$

Implementing Naive Bayes

3) Calculate this for every category, and select the one with highest probability.

Dealing with floating point underflow.

A big problem is multiplying small decimals for a large number of features may cause a floating point underflow.

To remedy this, transform each probability with **log()** or **ln()** and sum them, then call **exp()** to convert the result back!

$$\text{Occur Product} = \exp(\ln(P_{f1}) + \ln(P_{f2}) + \dots \ln(P_{fn}))$$

$$\text{Not Occur Product} = \exp(\ln(!P_{f1}) + \ln(!P_{f2}) + \dots \ln(!P_{fn}))$$

Implementing Naive Bayes

One last consideration, never let a feature have a 0 probability for any category!

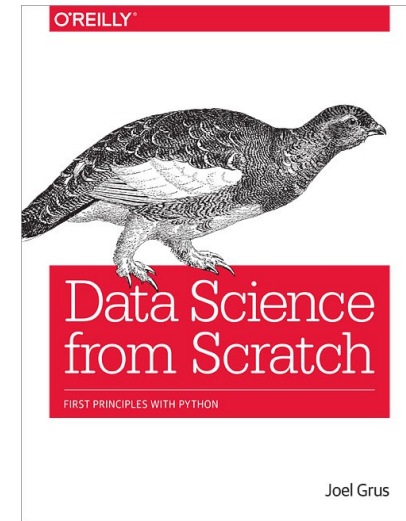
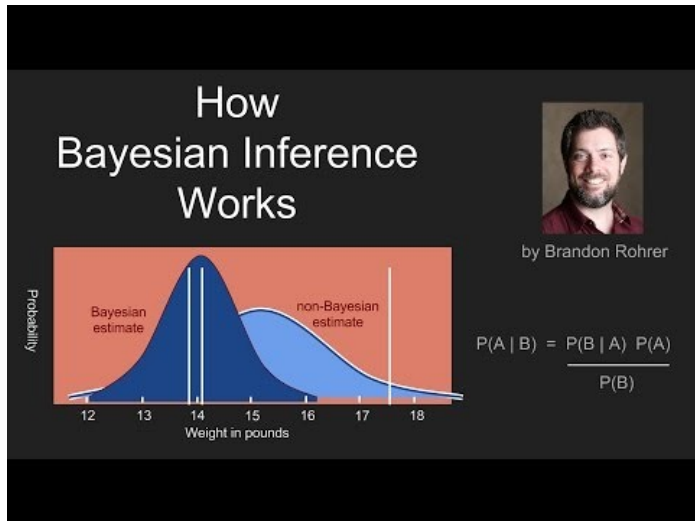
Always leave *a little* possibility it could belong to any category so you don't have 0 multiplication or division mess anything up.

This can be done by adding a small value to each probability's numerator and denominator (e.g. **0.5** and **1.0**).

$$\textit{CombinedProbability} = \frac{0.5 + (\text{Occur Product})}{1.0 + (\text{Occur Product}) + (\text{Not Occur Product})}$$

Learn More About Bayes

Brandon Rohrer - YouTube



Source Code

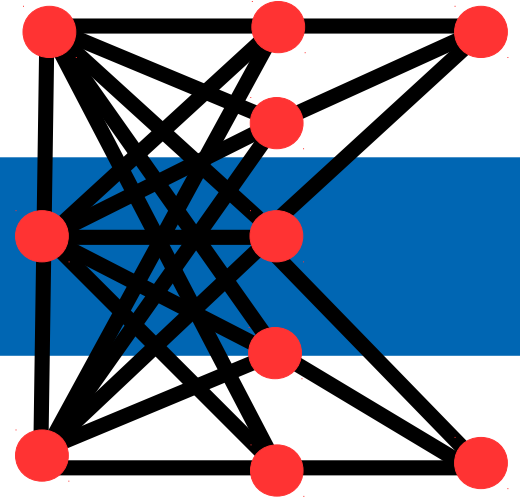
Bank Transaction Categorizer Demo

https://github.com/thomasniel/bayes_user_input_prediction

Email Spam Classifier Demo

https://github.com/thomasniel/bayes_email_spam

Part III: Neural Networks

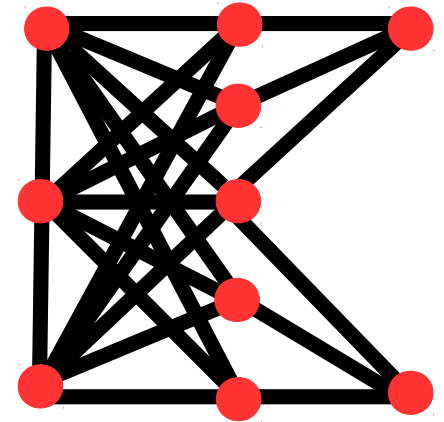


What Are Neural Networks?

Neural Networks are a machine learning tool that takes numeric inputs and predicts numeric outputs.

A series of multiplication, addition, and nonlinear functions are applied to the numeric inputs.

The mathematical operations above are iteratively tweaked until the desired output is met.



The Problem

Suppose we wanted to take a background color (in RGB values) and predict a light/dark font for it.

Hello

Hello

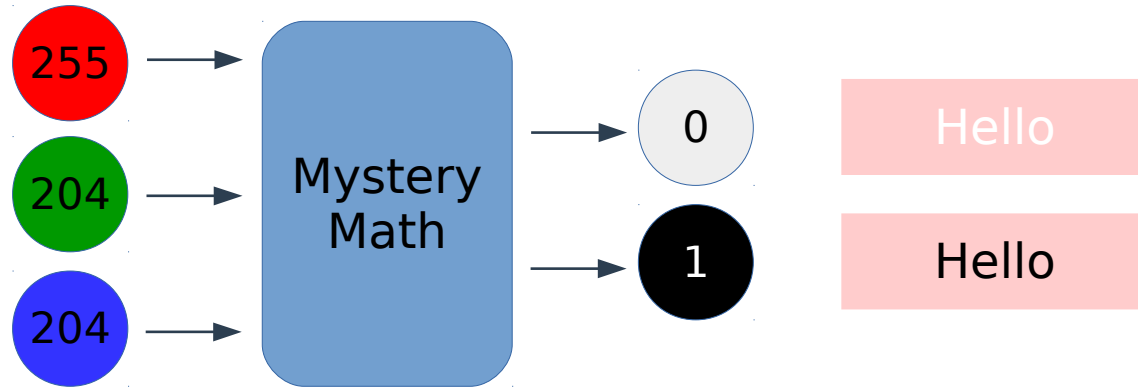
If you [search around Stack Overflow](#), there is a nice formula to do this:

$$L = (.299R + .587G + .114B)/255$$

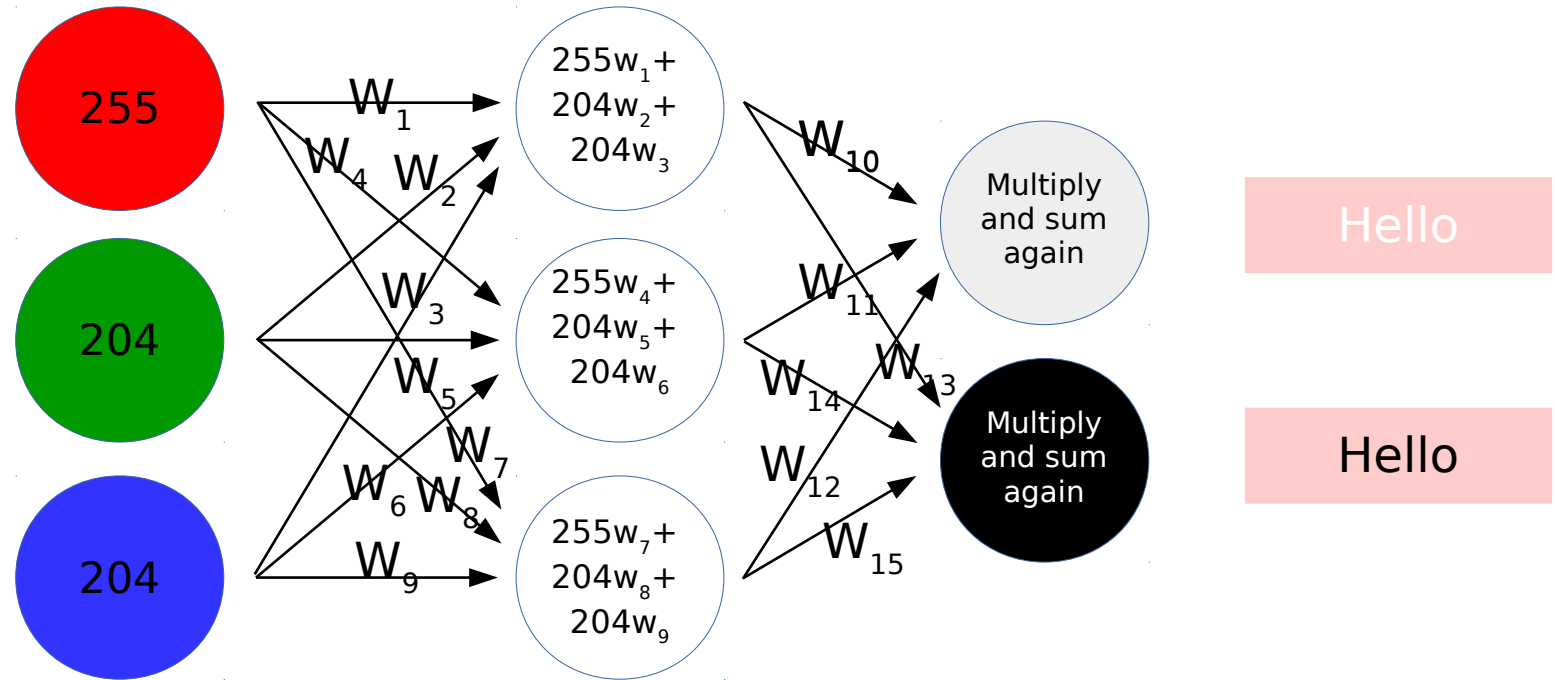
But what if we do not know the formula? Or one hasn't been discovered?

A Simple Neural Network

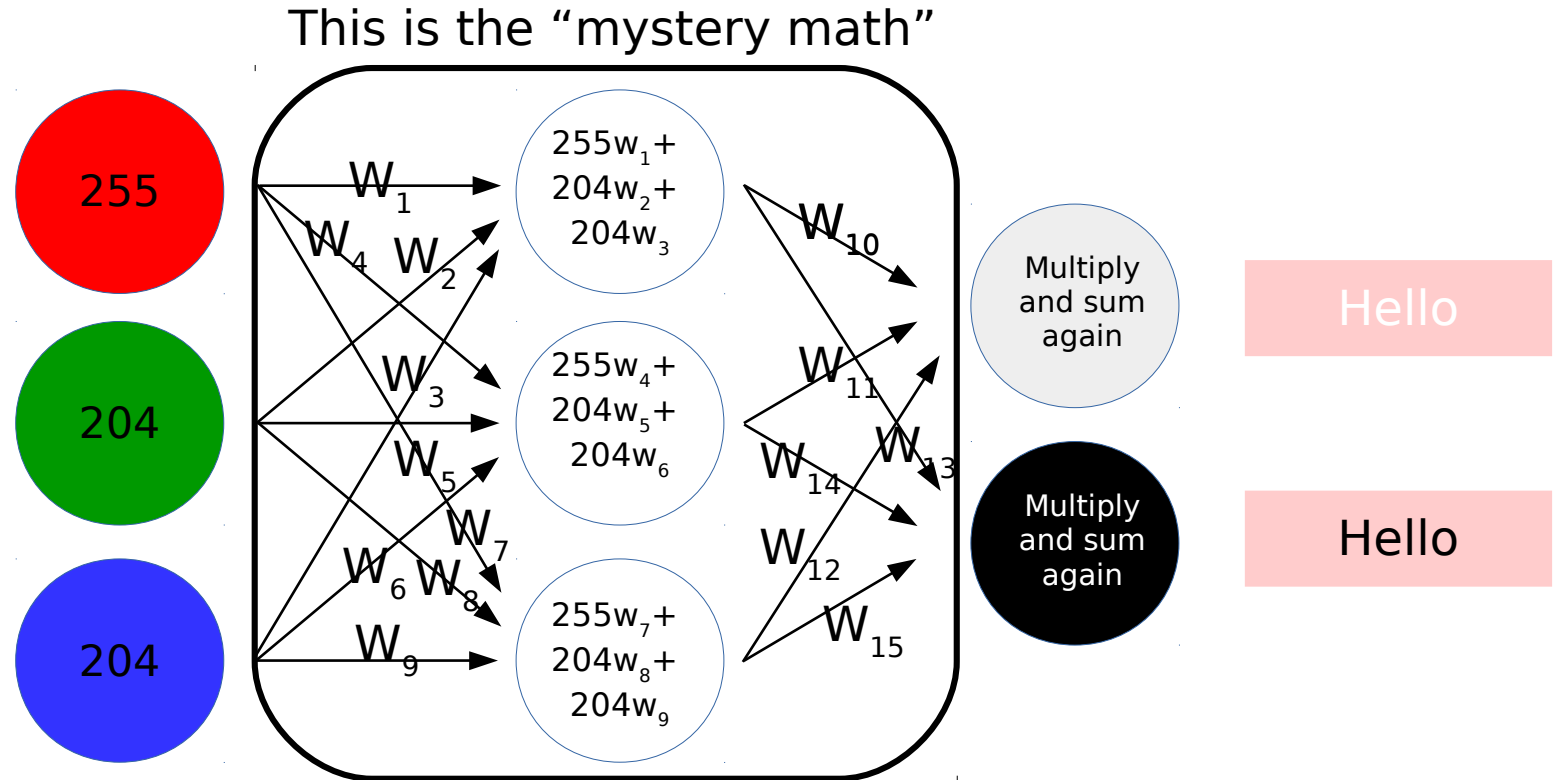
Let's represent background color as 3 numeric RGB inputs, and predict whether a **DARK/LIGHT** font should be used.



A Simple Neural Network

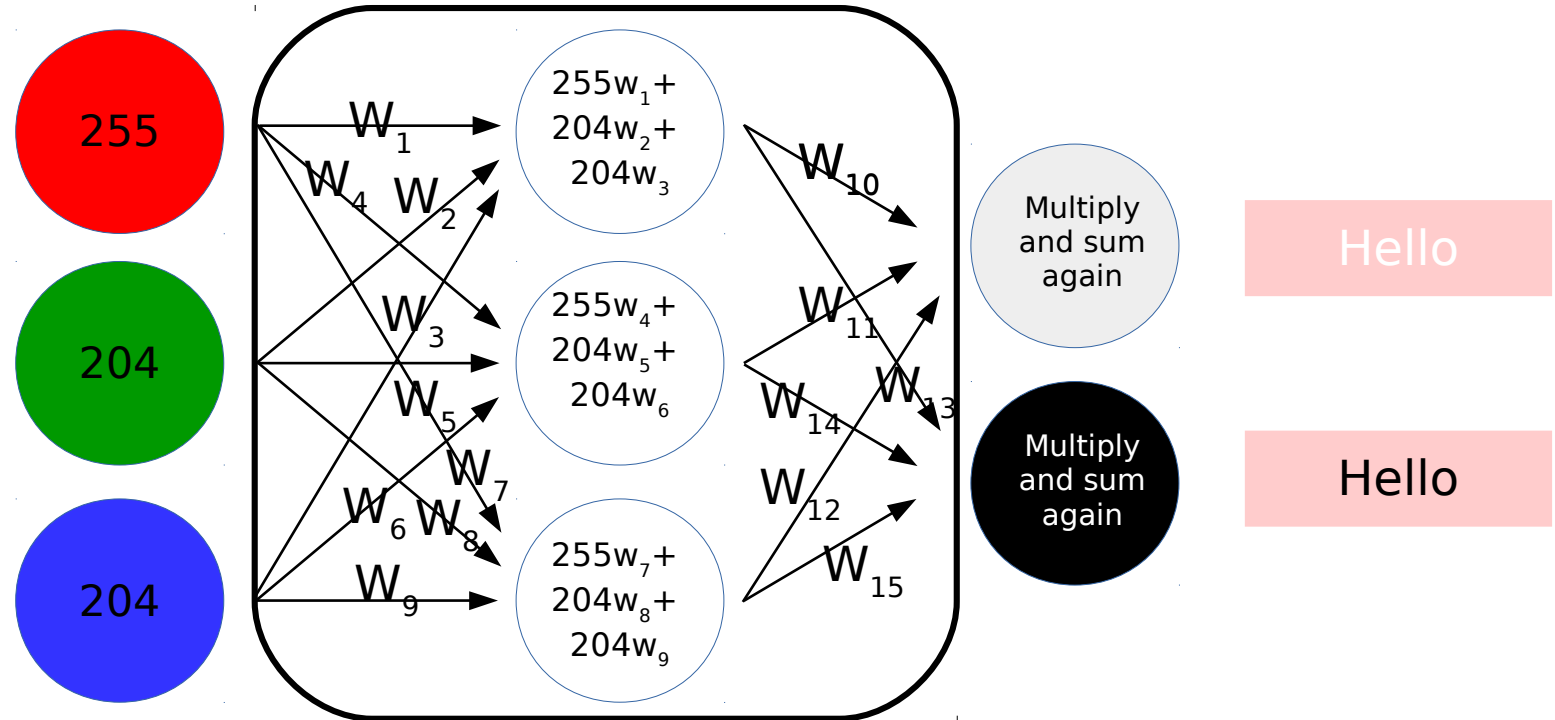


A Simple Neural Network



A Simple Neural Network

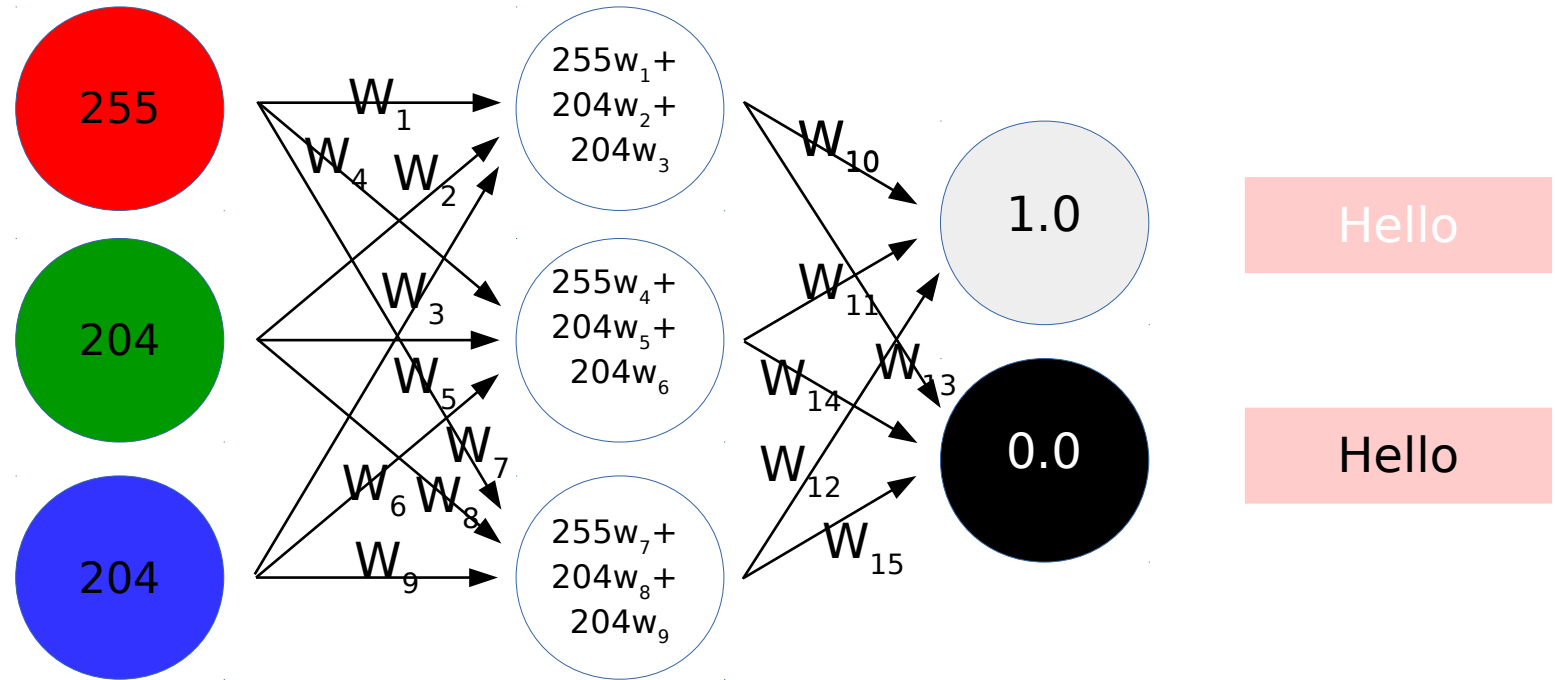
Each weight w_x value is between -1.0 and 1.0



A Simple Neural Network

Million Dollar Question:

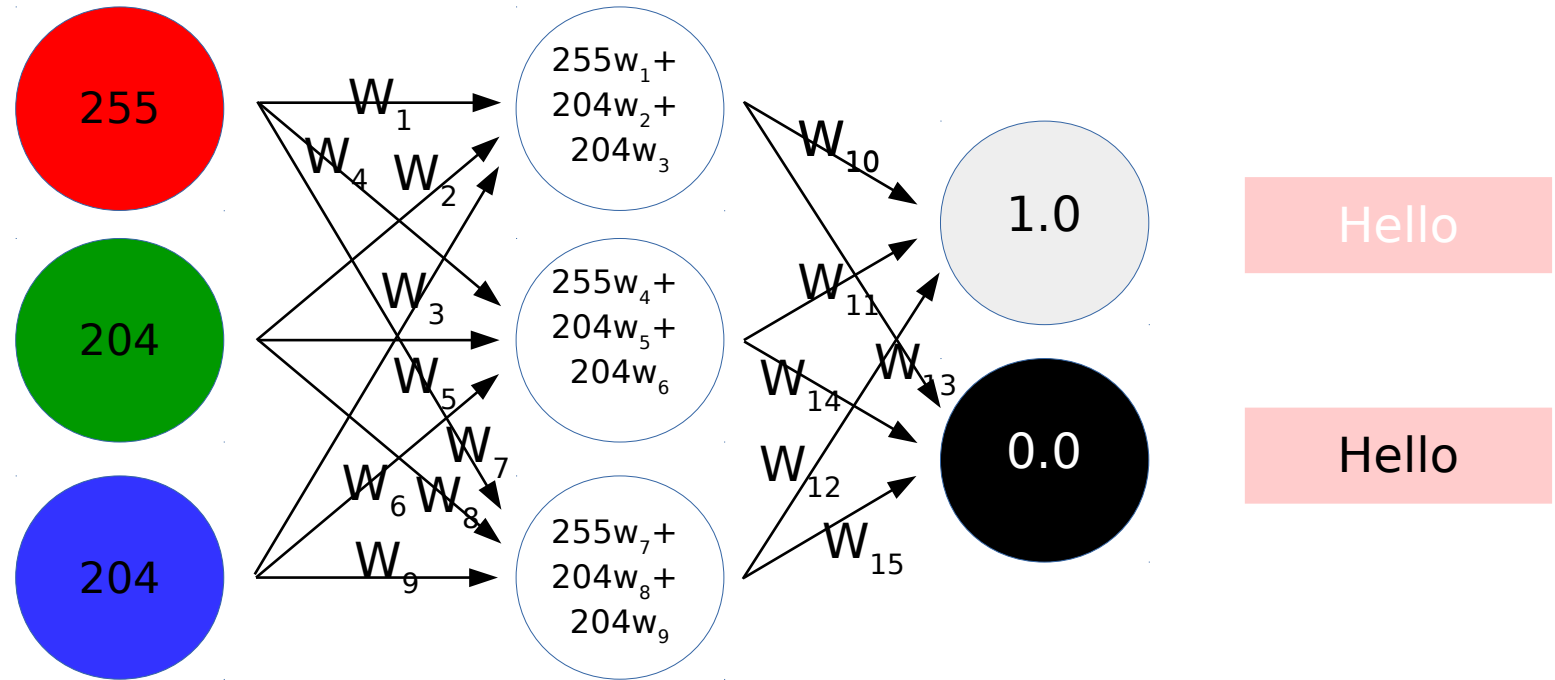
What are the *optimal* weight values to get the desired output?



A Simple Neural Network

Million Dollar Question:

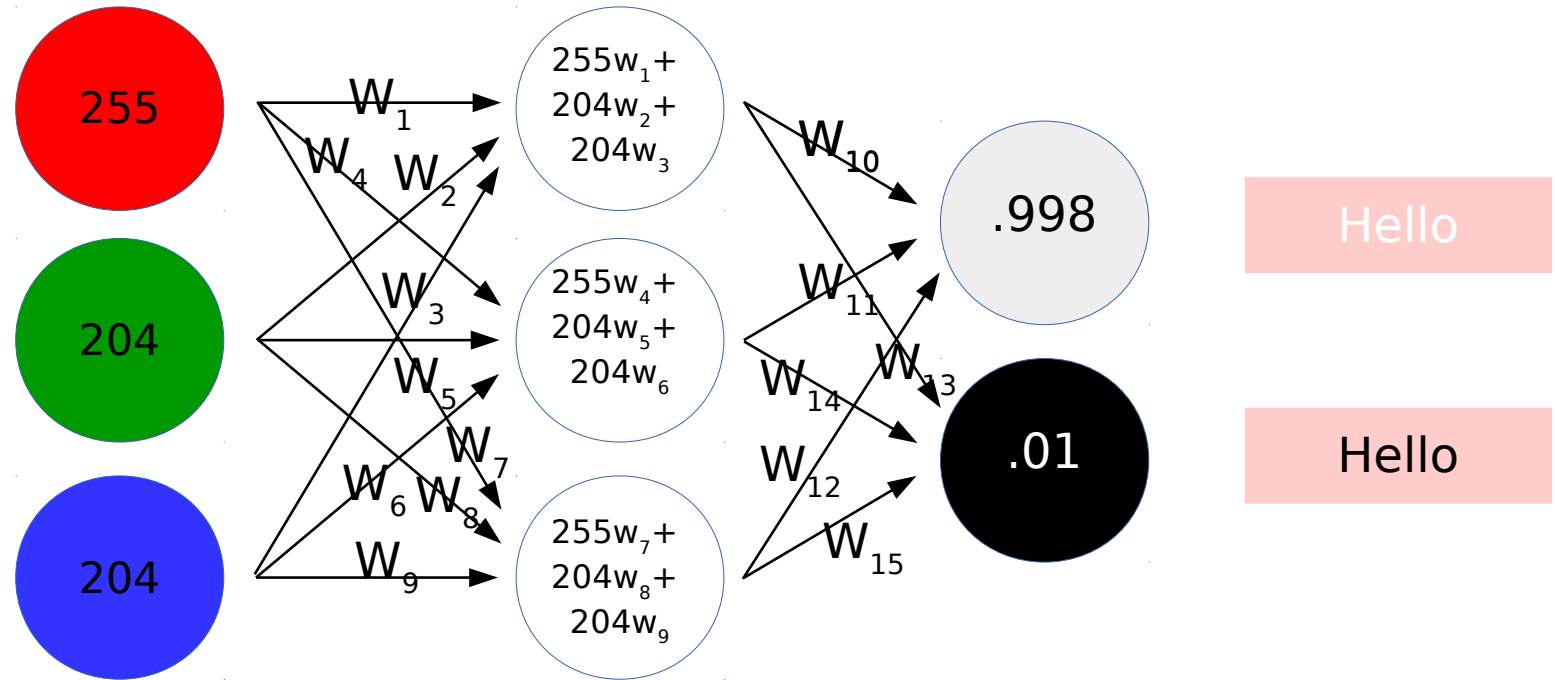
What are the *optimal* weight values to get the desired output?



A Simple Neural Network

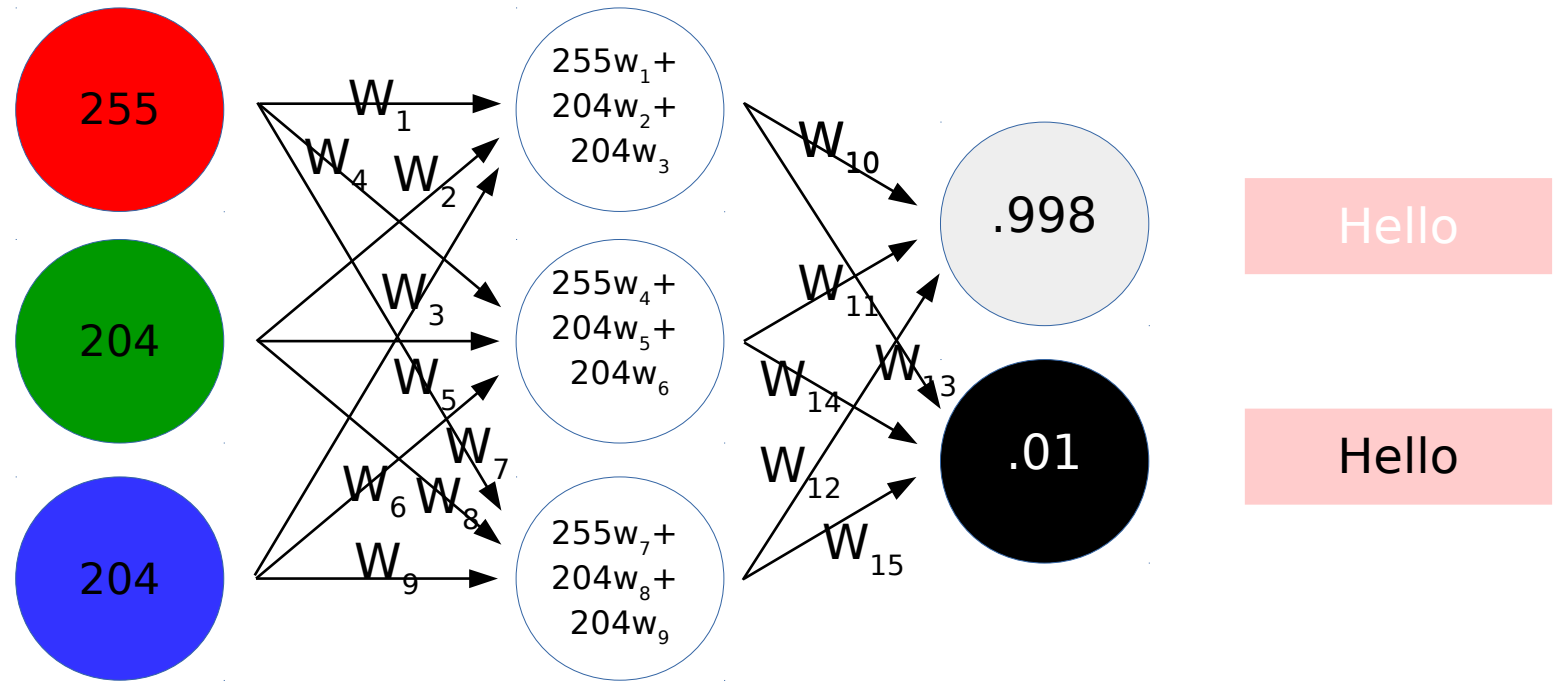
Answer:

This is an optimization problem!

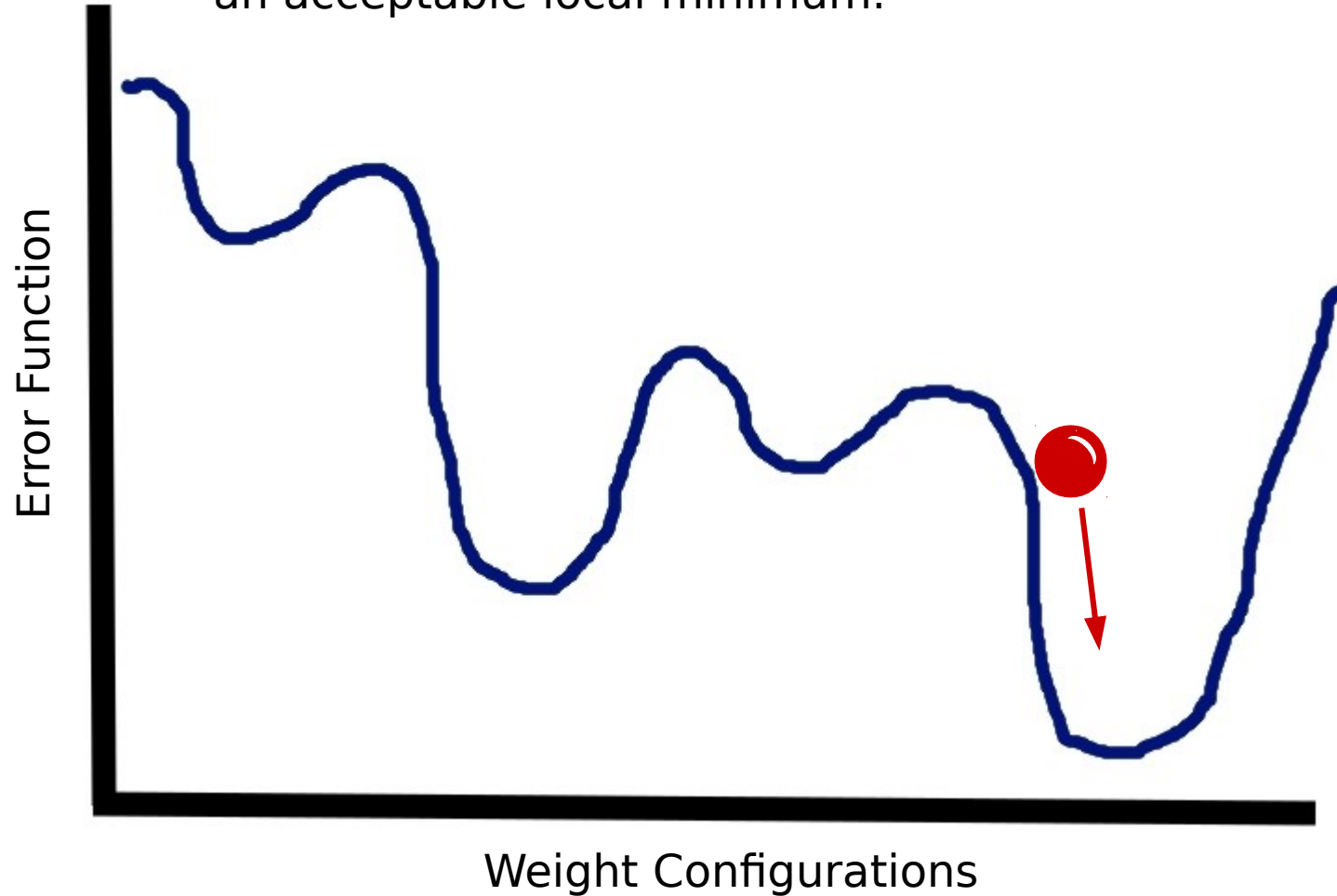


A Simple Neural Network

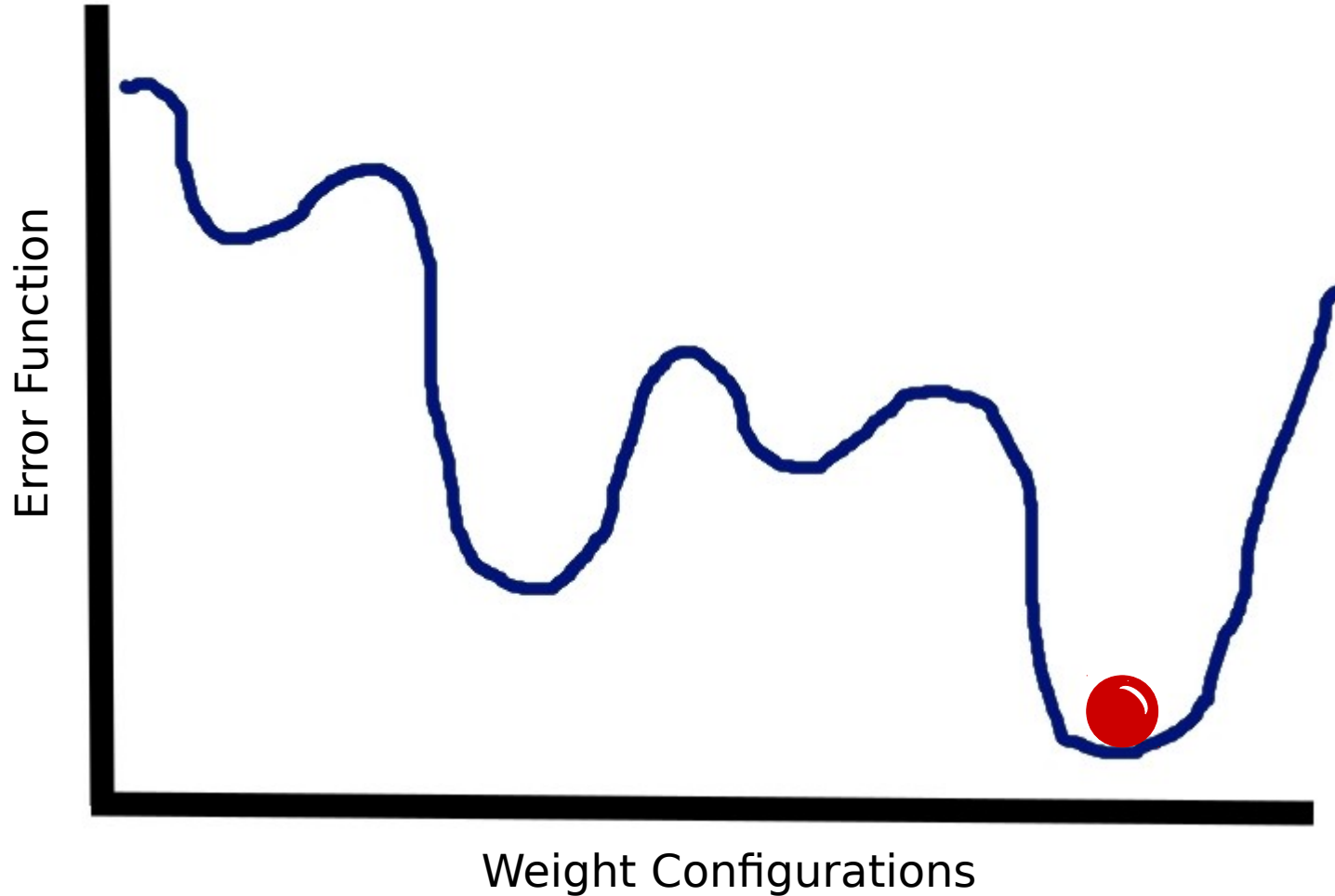
We need to solve for the weight values that gets our training colors as close to their desired outputs as possible.



Just like the Traveling Salesman Problem,
you need to explore configurations seeking
an acceptable local minimum.



Stochastic gradient descent, simulated annealing, and other optimization techniques can help tune a neural network.

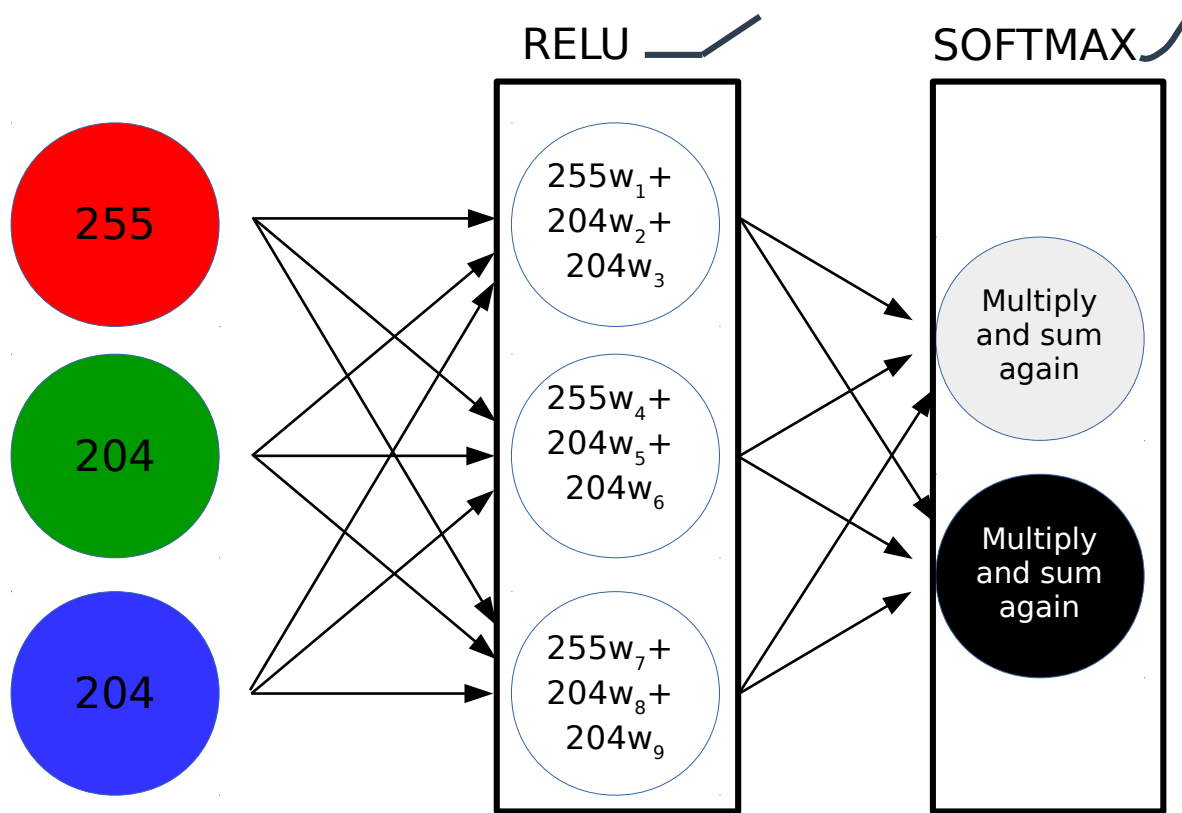


Activation Functions

You might also consider using **activation functions** on each layer.

These are nonlinear functions that smooth, scale, or compress the resulting sum values.

These make the network operate more naturally and smoothly.



Activation Functions

Four common neural network activation functions implemented using kotlin-stdlib

```
import kotlin.math.exp
import kotlin.math.max

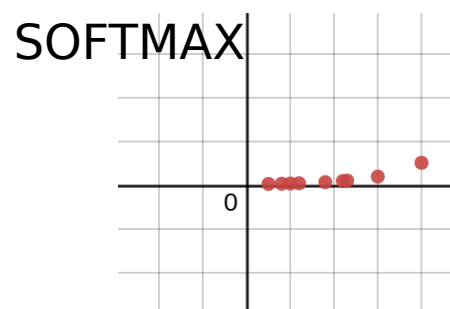
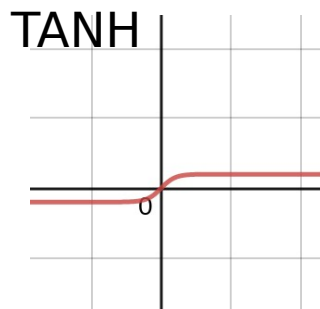
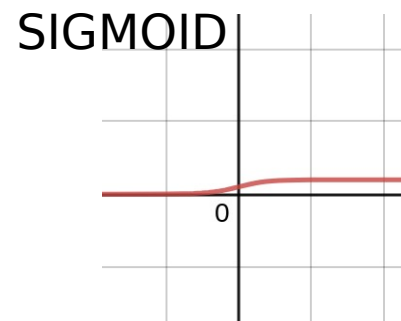
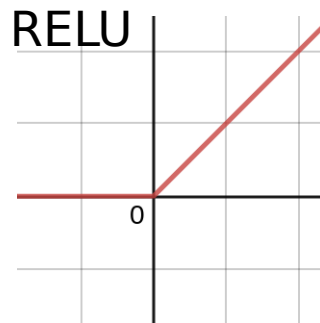
fun sigmoid(x: Double) = 1.0 / (1.0 + exp(-x))

fun relu(x: Double) = max(0.0, x)

fun softmax(x: Double, allValues: DoubleArray) =
    exp(x) / allValues.map { exp(it) }.sum()

fun tanh(x: Double) = kotlin.math.tanh(x)
```

<https://www.desmos.com/calculator/jwjn5rwfy6>



Other Design Decisions

While optimizing weights is a core part of neural networks, there are other components to consider too:

How many middle *layers* are needed?

How many *nodes* are needed in each layer?

What *activation functions* should be applied to each layer?

Middle Layers - Should the layers be recurrent, recursive, convolutional, etc?

Loss function - How should we measure error?

Learning Rate - How aggressively should the optimization move towards the local minimum?

The Practicality of Neural Networks

Despite some seemingly intelligent applications like image recognition, neural networks have drawbacks:

Requires **LOTS** of labeled data for training.

Difficult for the data scientist to understand convoluted layers and nodes.

Has a quick diminishing return for problem spaces outside of image and natural language processing.

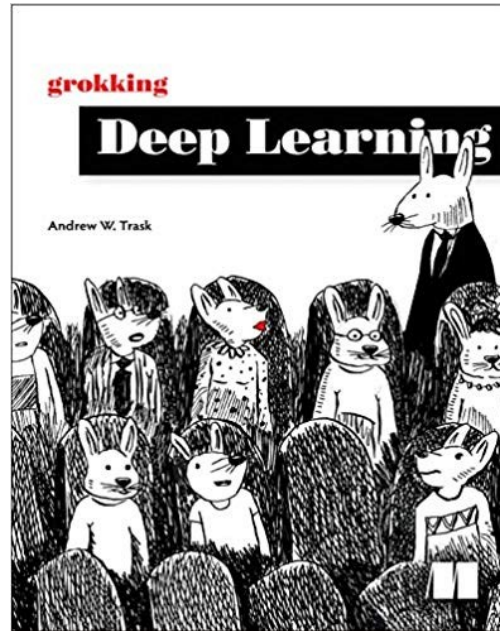
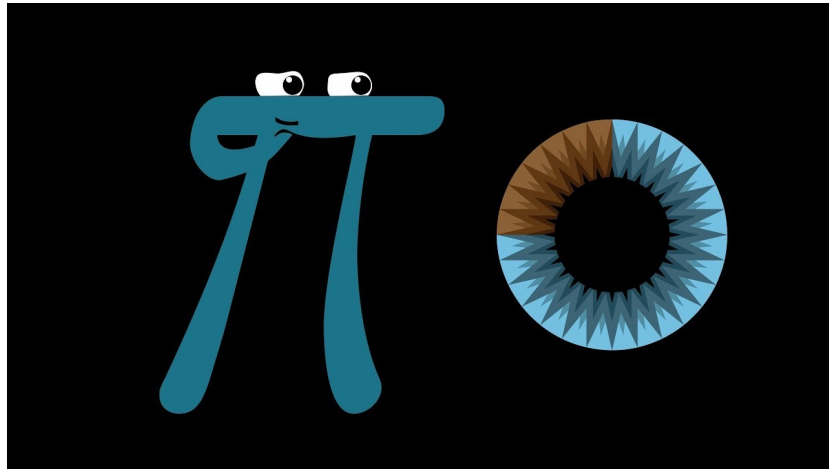
Neural networks generate excitement because many believe they will generalize solutions to most problems one day.

In reality, specialized and incumbent algorithms will outperform neural networks for most practical problems.

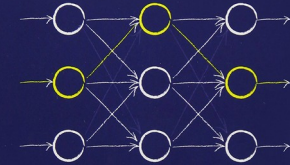
There are many academic papers citing research applying neural networks to the Traveling Salesman Problem and while this is interesting, the results are underwhelming.

Learn More About Neural Networks

3Blue1Brown - YouTube




MAKE YOUR
OWN
NEURAL NETWORK



*A gentle journey through the mathematics of
neural networks, and making your own
using the Python computer language.*


TARIQ RASHID

Also Read My Recent Blog Article

 Towards Data Science [Follow](#)

[Sign in](#) [Get started](#)


[DATA SCIENCE](#) [MACHINE LEARNING](#) [PROGRAMMING](#) [VISUALIZATION](#) [AI](#) [JOURNALISM](#) [CONTRI](#)



The breakthrough "[MAC Hack VI](#)" chess program in 1965.

Is Deep Learning Already Hitting its Limitations?

And Is Another AI Winter Coming?

 **Thomas Nield** [Follow](#)
Jan 5 · 11 min read

Many believed an algorithm would transcend humanity with cognitive awareness. Machines would discern and learn tasks without human intervention and replace workers in droves. They quite literally would be able

Source Code

Kotlin Neural Network Example

https://github.com/thomasniel/kotlin_simple_neural_network

Going Forward



The Obstacle of Generality

Generalized terms like “AI” are helpful to spur positive change and embrace new mindsets.

However such terms do not help with solution planning and execution, and can even create red herrings and frustration.

Silicon Valley may have categorically different problems from your industry.

Only a handful of problem spaces (e.g. image and language processing) are optimally solved with “deep learning” and “neural networks.”

Unsupervised learning is **very** much in infancy, and therefore general AI is nowhere in sight.

The Obstacle of Generality

How to avoid the “Obstacle of Generality”:

Be detailed and specific about your problem.

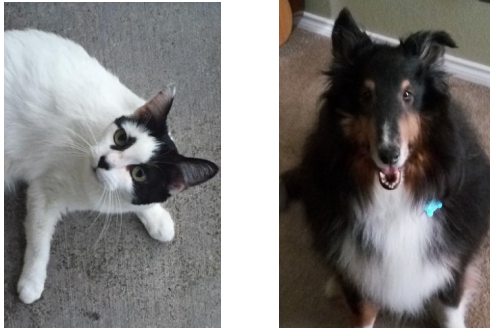
Find models that align to your problem's nature.

Choose a model that provides a clear and intuitive path to a solution.

Approach convoluted models with a healthy amount of skepticism.

Use the Right “AI” for the Job

Neural Networks



- **Image/Audio/Video Recognition**
“Cat” and “Dog” photo classifier
- **Natural language processing**
- **Any fuzzy, difficult problems that have no clear model but lots of data**
Self-driving vehicles
Difficult nonlinear regressions
Problems w/ mysterious unknowns

Bayesian Inference



- **Text classification**
Email spam, sentiment analysis, document categorization
- **Document summarization**
- **Probability inference**
Disease diagnosis, updating predictions

Discrete Optimization



- **Routing and Scheduling**
Staff, transportation, classrooms, sports tournaments, server jobs
- **On-Time Optimization**
Transportation, manufacturing
- **Industry**
Manufacturing, farming, nutrition, energy, engineering, finance

GitHub and Slides

https://github.com/thomasniel/JavaMug2019_Many_Types_of_AI

Appendix



Pop Culture

Traveling Salesman (2012 Movie)

<http://a.co/d/76UYvXd>

Silicon Valley (HBO) - The “Not Hotdog” App

<https://youtu.be/vlci3C4JkL0>

Silicon Valley (HBO) - Making the “Not Hotdog” App

<https://tinyurl.com/y97ajsac>

XKCD - Traveling Salesman Problem

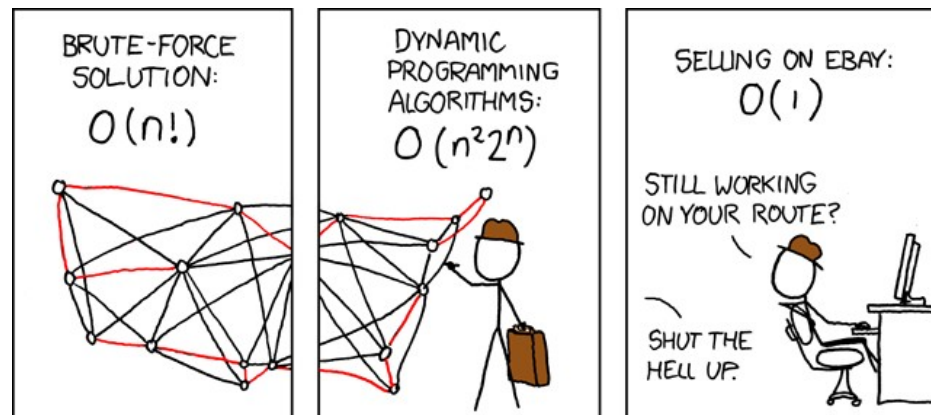
<https://www.xkcd.com/399/>

XKCD - NP-Complete

<https://www.xkcd.com/287/>

XKCD - Machine Learning

<https://xkcd.com/1838/>



SOURCE: xkcd.com

Areas to Explore

Machine Learning

- Linear Regression
- Nonlinear Regression
- Neural Networks
- Bayes Theorem/Naive Bayes
- Support Vector Machines
- Decision Trees/Random Forests
- K-means (nearest neighbor)
- XGBoost

Optimization

- Discrete Optimization
- Linear/Integer/Mixed Programming
- Dynamic Programming
- Constraint programming
- Metaheuristics

Java/Kotlin ML and Optimization Libraries

Java/Kotlin Library	Python Equivalent	Description
ND4J	NumPy	Numerical computation Java library
DeepLearning4J	TensorFlow	Deep learning Java/Scala/Kotlin library
SMILE	scikit-learn	Comprehensive machine learning suite for Java
ojAlgo and okAlgo	PuLP, NumPy	Linear algebra and optimization library for Java
Apache Commons Math	scikit-learn	Math, statistics, and ML for Java
TableSaw / Krangl	Pandas	Data frame libraries for Java/Kotlin
Kotlin-Statistics	scikit-learn	Statistical/probability operators for Kotlin
JavaFX / Vegas / Data2Viz	matplotlib	Charting libraries

Online Class Resources

Coursera - Discrete Optimization

<https://www.coursera.org/learn/discrete-optimization/home/>

Coursera - Machine Learning

<https://www.coursera.org/learn/machine-learning/home/welcome>

YouTube Channels and Videos

Thomas Nield

<https://youtu.be/F6RiAN1A8n0>

Brandon Rohrer

<https://www.youtube.com/c/BrandonRohrer>

3Blue1Brown

https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AjtAw

YouTube - P vs NP and the Computational Complexity Zoo

<https://youtu.be/YX40hbAHx3s>

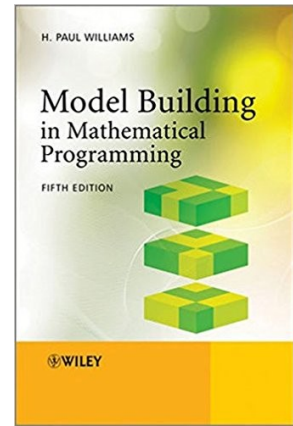
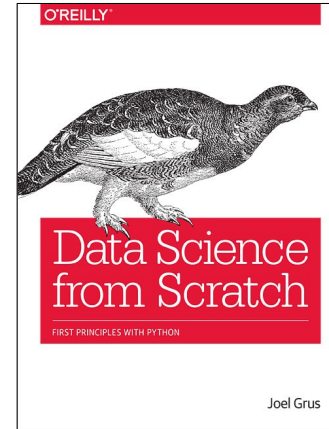
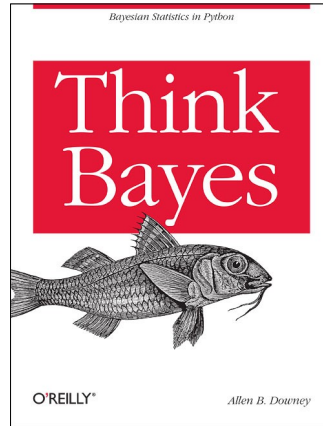
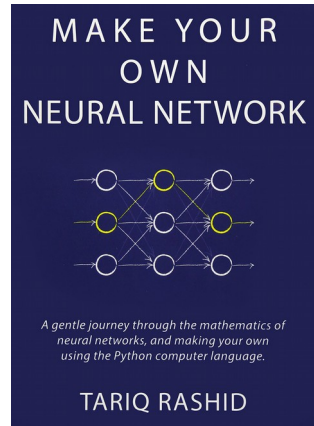
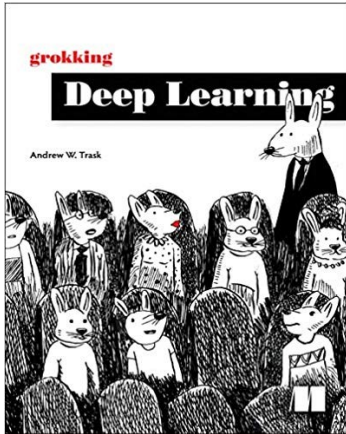
The Traveling Salesman Problem Visualization

<https://youtu.be/SC5CX8drAtU>

The Traveling Salesman w/ 1000 Cities (Video)

https://youtu.be/W-aAjd8_bUc

Books



Interesting Articles

Does A.I. Include Constraint Solvers?

<https://www.optaplanner.org/blog/2017/09/07/DoesAIIncludeConstraintSolvers.html>

Can You Make Swiss Trains Even More Punctual?

<https://medium.com/crowdai/can-you-make-swiss-trains-even-more-punctual-ec9aa73d6e35>

The SkyNet Salesman

<https://multithreaded.stitchfix.com/blog/2016/07/21/skynet-salesman/>

Interesting Articles

Essential Math for Data Science

<https://towardsdatascience.com/essential-math-for-data-science-why-and-how-e88271367fbd>

The Unreasonable Reputation of Neural Networks

<http://thinkingmachines.mit.edu/blog/unreasonable-reputation-neural-networks>

Mario is Hard, and that's Mathematically Official

<https://www.newscientist.com/article/mg21328565.100-mario-is-hard-and-thats-mathematically-official/>

Interesting Papers

The Lin-Kernighan Traveling Salesman Heuristic

http://akira.ruc.dk/~keld/research/LKH/LKH-1.3/DOC/LKH_REPORT.pdf

The Traveling Salesman: A Neural Network Perspective

http://www.iro.umontreal.ca/~dift6751/paper_potvin_nn_tsp.pdf

The Interplay of Optimization and Machine Learning Research

<http://jmlr.org/papers/volume7/MLOPT-intro06a/MLOPT-intro06a.pdf>