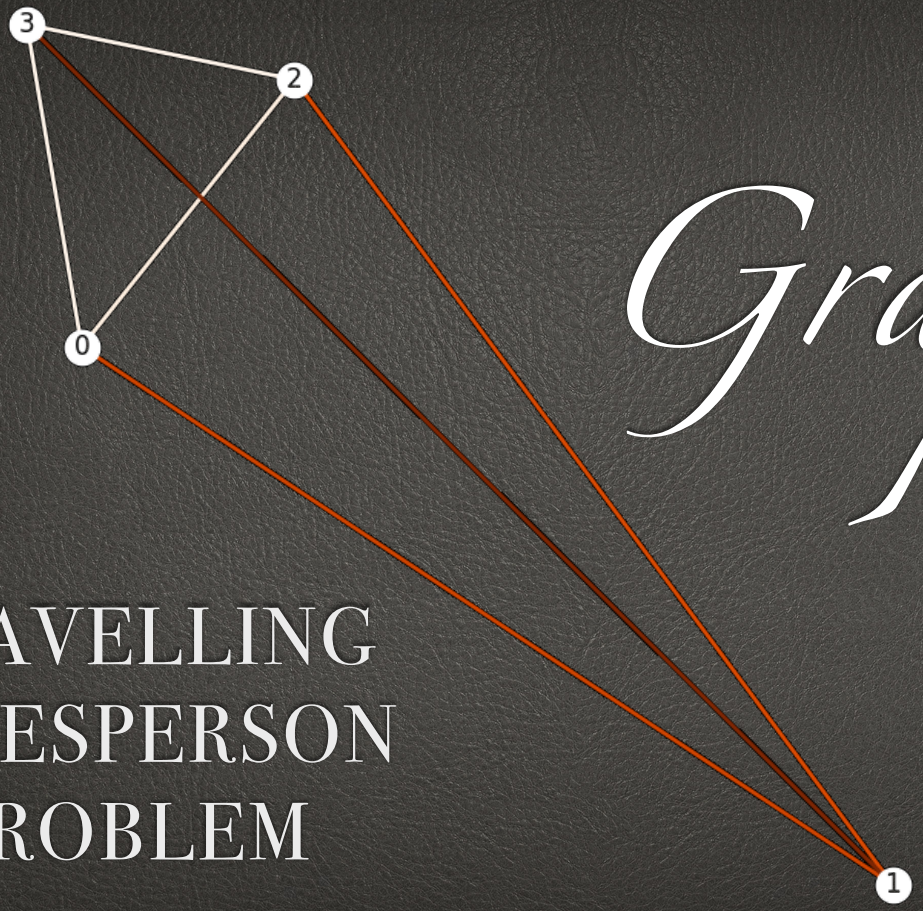
Discrete Math meets Deep Learning

a hands-on case study with TSP



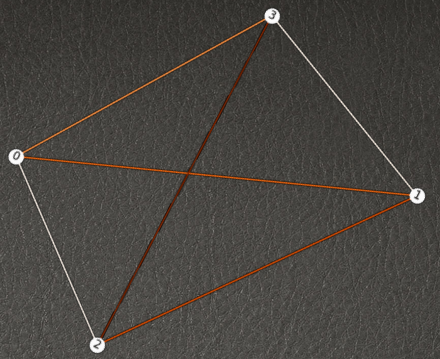
Graph

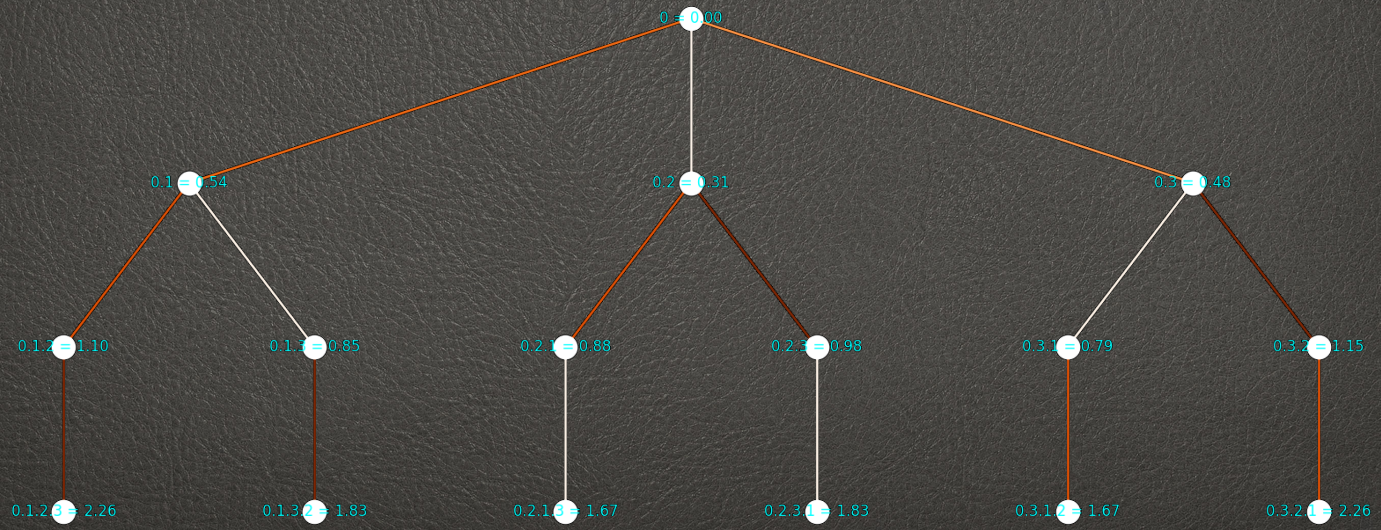
TRAVELLING SALESPERSON PROBLEM

😸 If we solve it, we solve tons of things

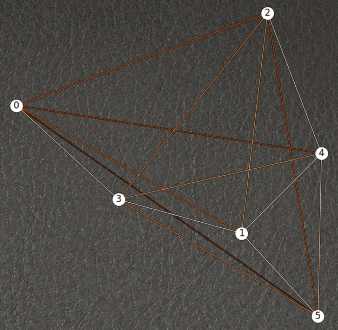
It's an NP-hard optimization problem 😿

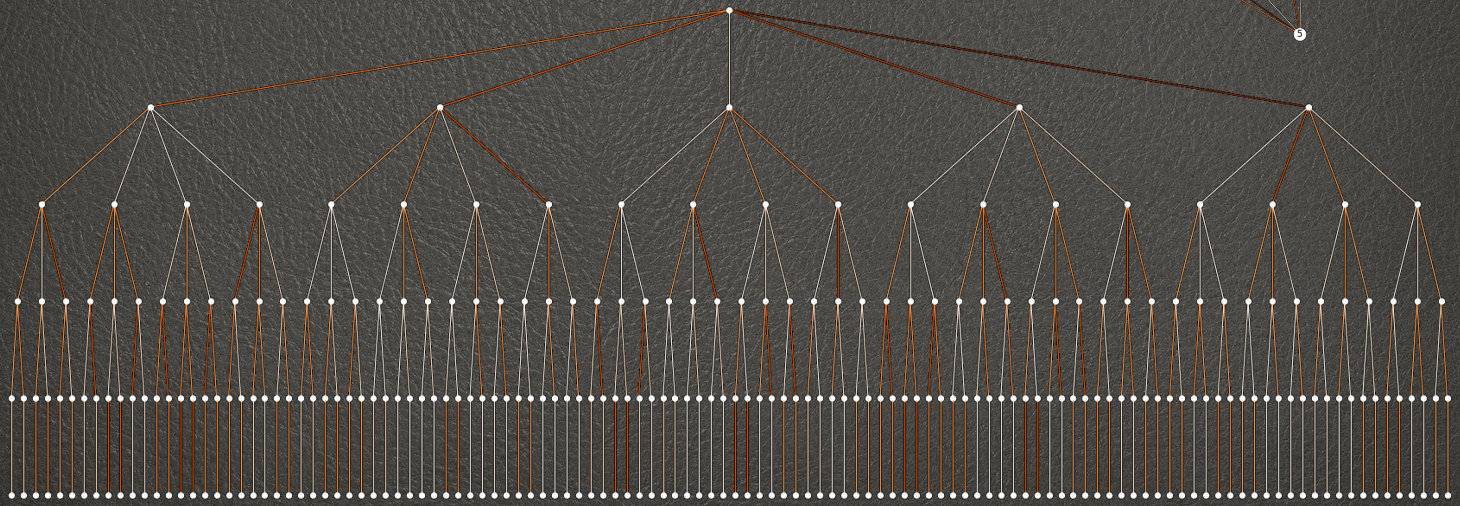
Exhaustive search





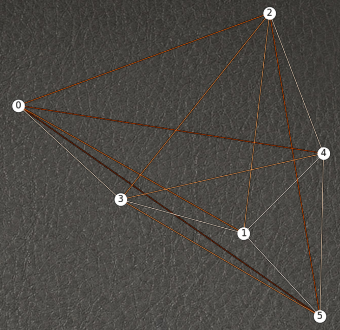
Combinatorial explosion

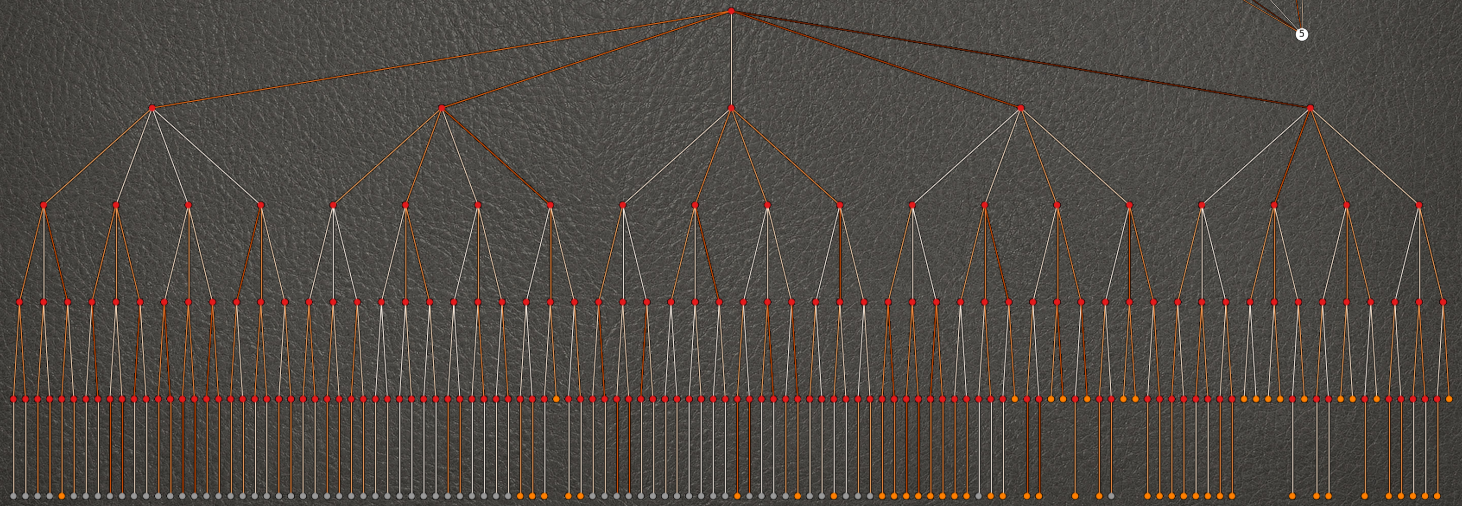




With just six 🐱

Pruning helps, a little





With just six 😺

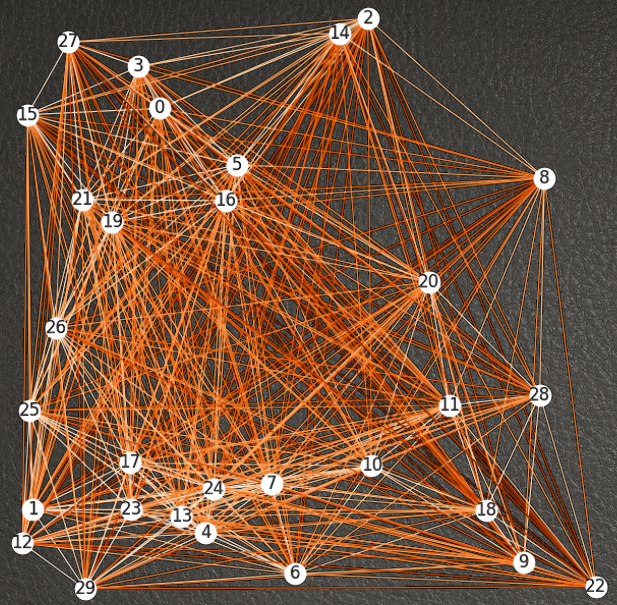
Global optimum

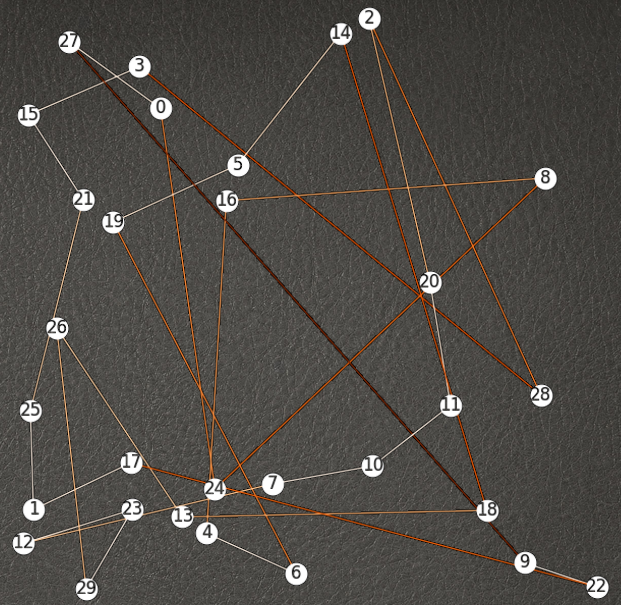
Example runtime in milliseconds

n	Exhaustive	Pruned
8	88	30
9	825	264
10	9021	1371
11	*lost patience*	3234
12	*lost patience*	23815
13	*lost patience*	*lost patience*

😺 Nothing can produce a lower cost

Those trees grow too big 😿



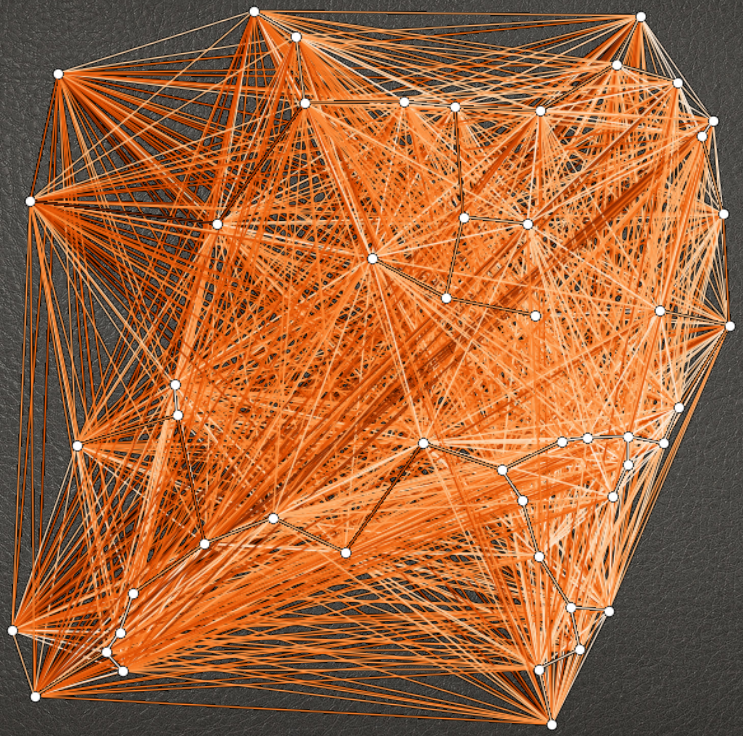


Replicas of self-avoiding random walks

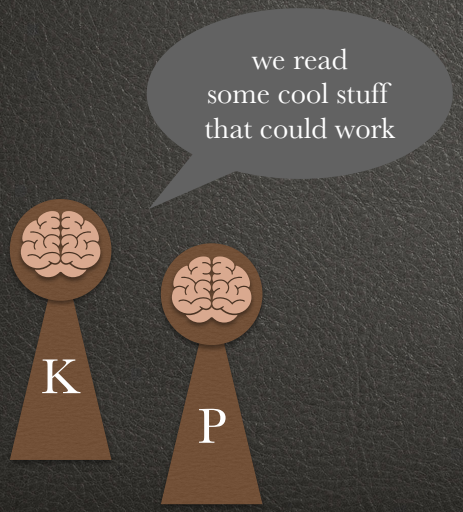
😸 Super fast even for large graphs

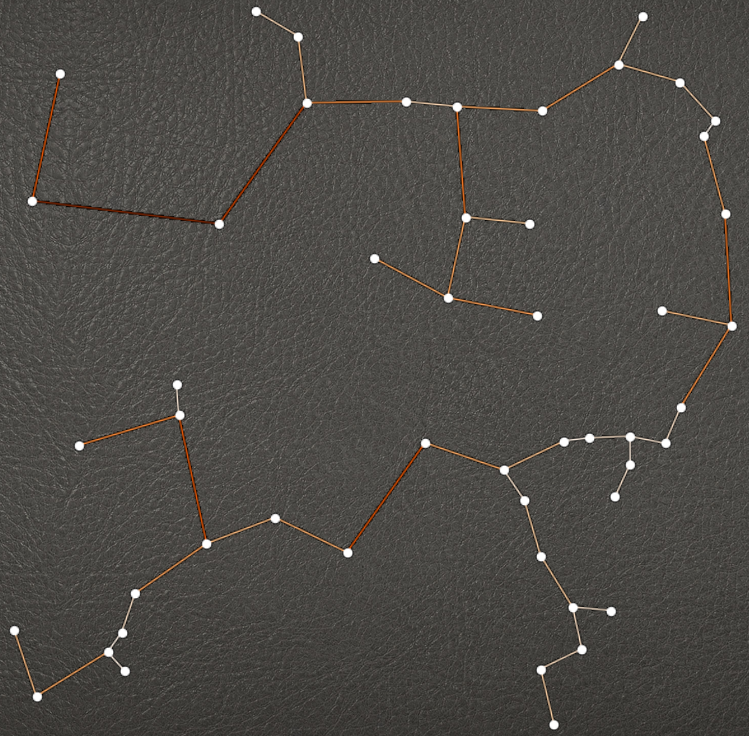
No guarantee of optimality 😿

WHAT ELSE CAN WE DO THAT IS FAST TO COMPUTE?



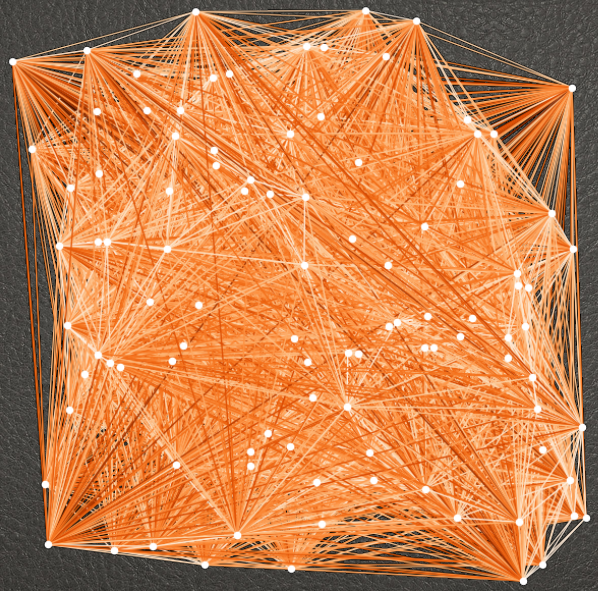
WHAT ELSE CAN WE DO THAT IS FAST TO COMPUTE?





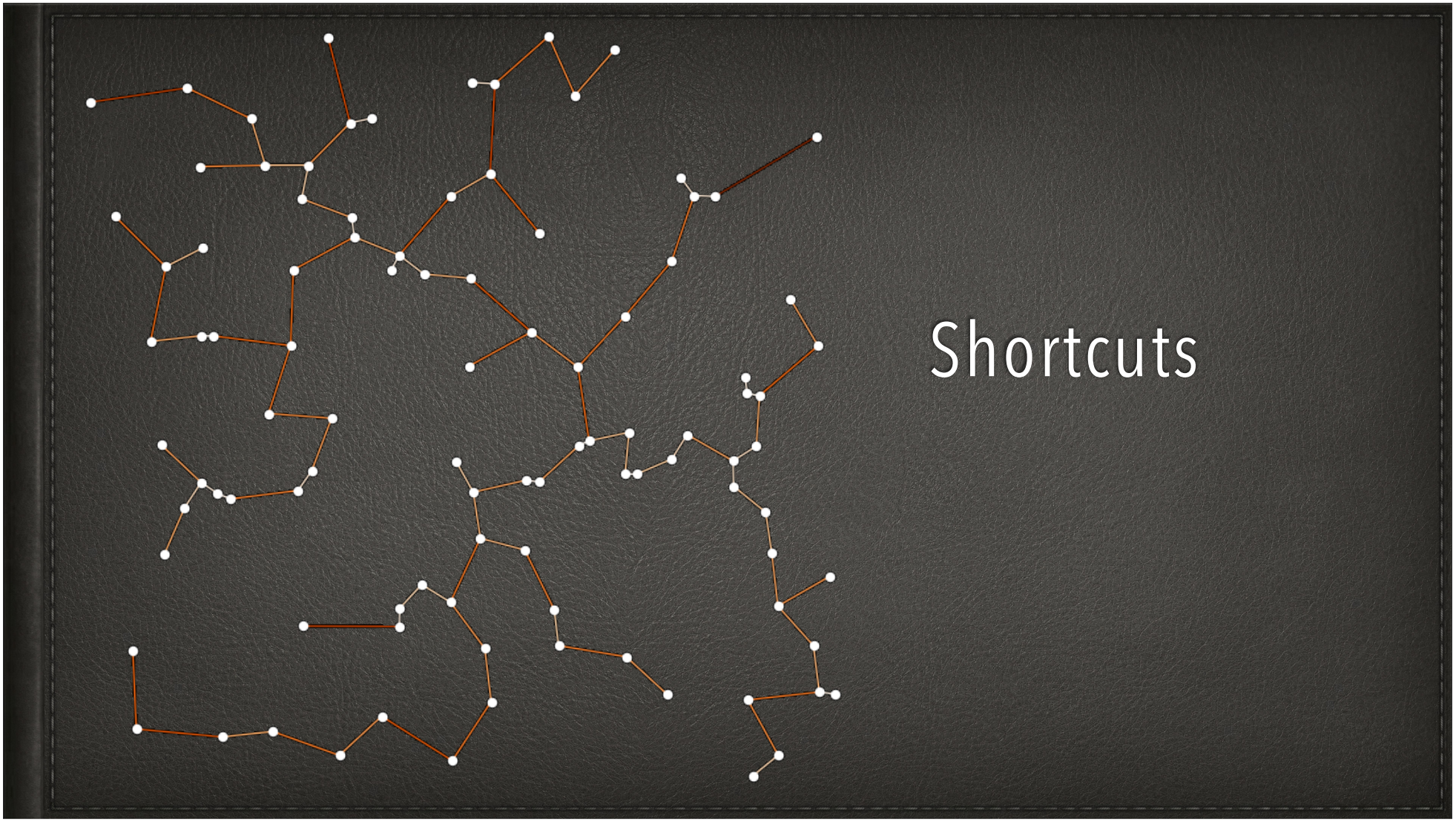
That's not a cycle, fellas 😿

Minimum spanning tree

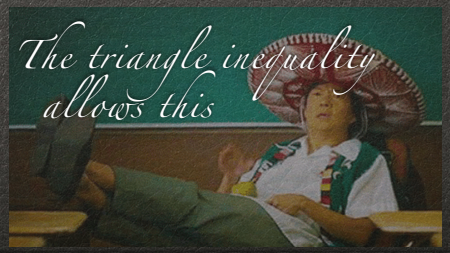


😺 This is quick to build and we can traverse it back and forth with a cost at most twice the optimum

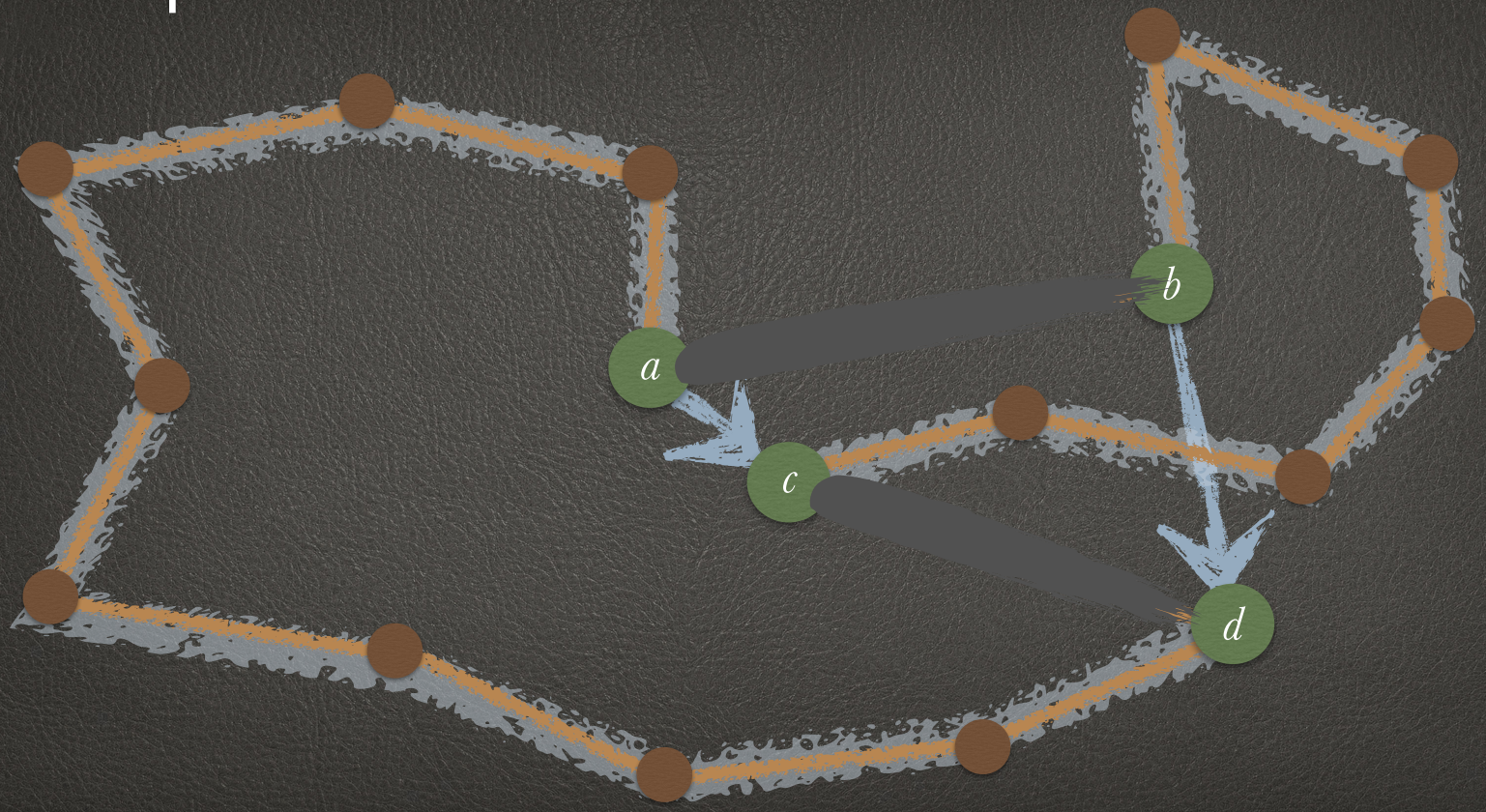
That's a cycle, yes, but it is still infeasible 😿


Shortcuts

Shortcuts

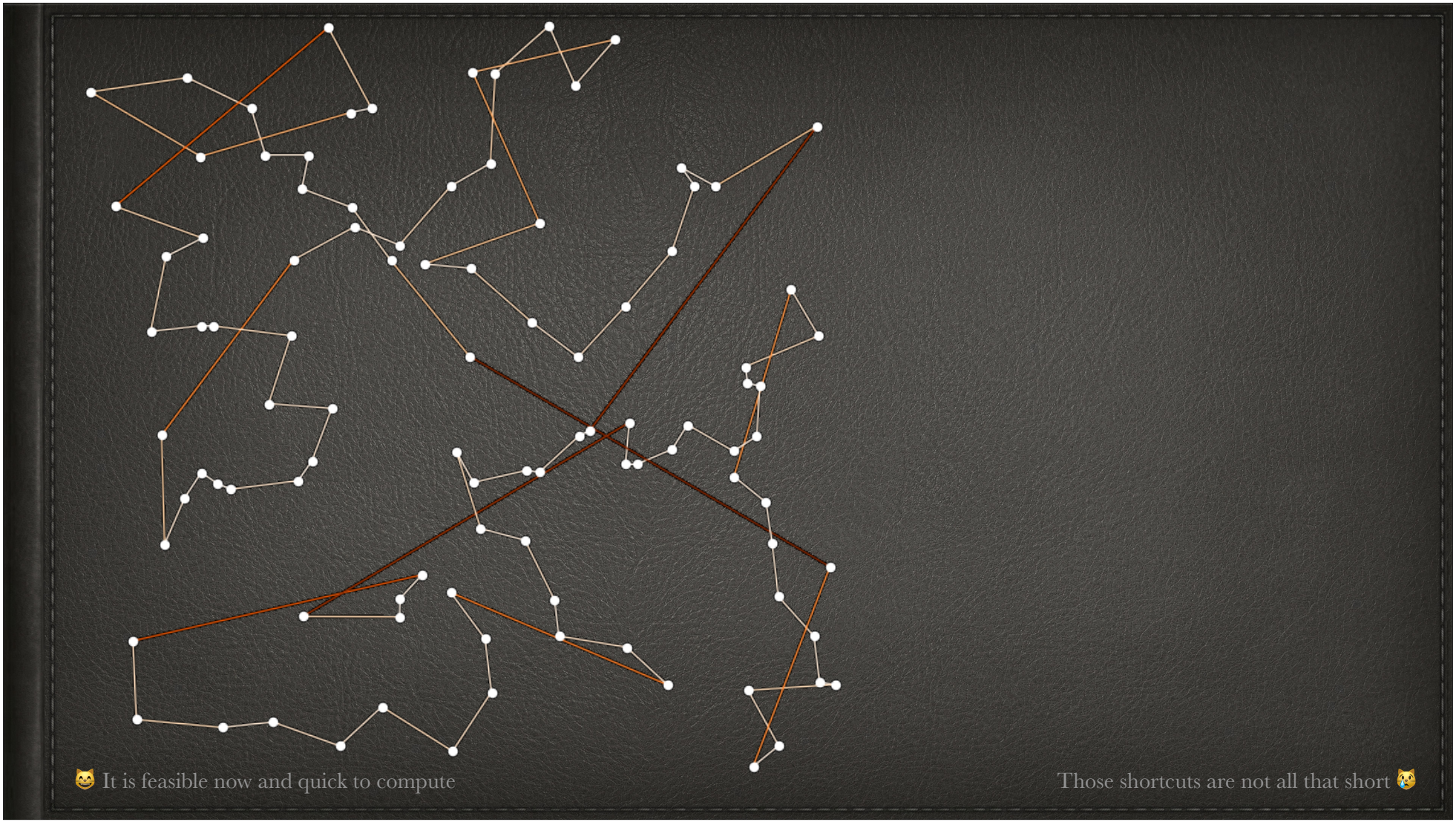


2-opt



😸 That could be cheaper than before

This was obviously never an MST, but just bear with me


😺 It is feasible now and quick to compute
Those shortcuts are not all that short 😿

Local search

😺 It is feasible now and quick to compute

Those shortcuts are not all that short 😿

What if we don't know the edge costs?

But we can observe some routes and their costs

Deep learning

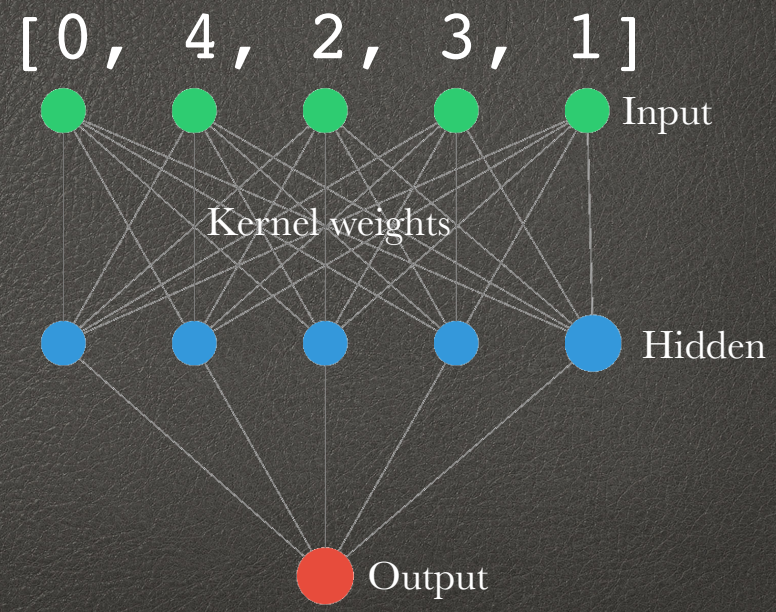
- A balanced **data set** with *good* and bad *routes*
- *Split* it into **X** for training and **X*** for testing
- Assign *labels* **y** and **y*** *"costs < threshold"*
- Add an **input layer**
- Later we will use more *hidden layers*
- Add an **output layer** (with *one* output)
- **Fit** that **X** to that **y**
- **Test** the model with **X*** and **y***

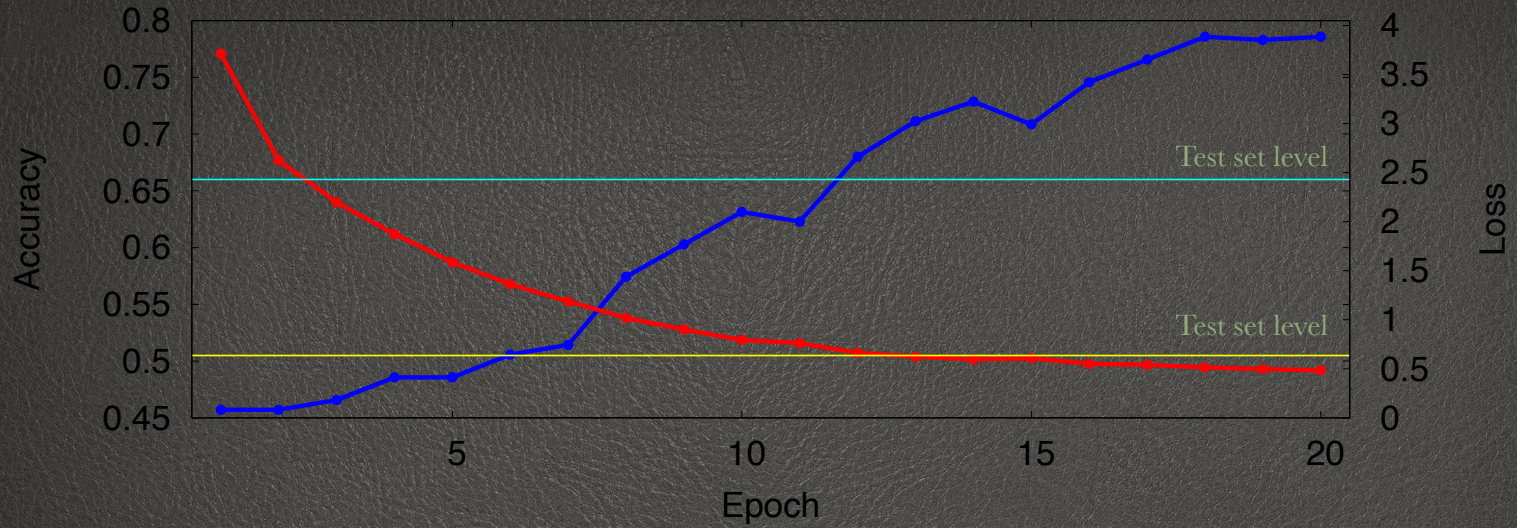
Assume your graph will not change a lot over time

```
(0, 3, 13, 14, 4, 10, 1, 5, 16, 8, 2, 12, 9, 15, 18, 7, 6, 17, 11, 19, 0) -> 5.090 -> True
(0, 12, 17, 5, 13, 11, 15, 4, 3, 1, 9, 8, 2, 16, 19, 6, 14, 7, 18, 10, 0) -> 10.250 -> False
(0, 3, 19, 14, 13, 11, 5, 1, 2, 8, 16, 12, 9, 10, 15, 7, 18, 4, 6, 17, 0) -> 5.140 -> True
(0, 3, 17, 4, 14, 13, 6, 11, 15, 10, 18, 7, 9, 12, 8, 2, 16, 5, 1, 19, 0) -> 5.054 -> True
(0, 12, 17, 2, 1, 15, 13, 14, 18, 3, 9, 19, 6, 16, 7, 4, 10, 8, 5, 11, 0) -> 10.963 -> False
(0, 3, 6, 10, 11, 12, 1, 2, 5, 9, 16, 8, 7, 15, 18, 4, 14, 13, 17, 19, 0) -> 5.194 -> True
(0, 11, 14, 16, 1, 4, 15, 18, 6, 13, 8, 17, 5, 2, 10, 7, 19, 3, 9, 12, 0) -> 11.006 -> False
(0, 19, 11, 1, 5, 2, 8, 9, 12, 16, 7, 15, 18, 6, 4, 10, 14, 13, 17, 3, 0) -> 4.498 -> True
(0, 1, 12, 2, 14, 17, 3, 9, 4, 5, 6, 16, 18, 7, 15, 10, 8, 19, 11, 13, 0) -> 10.236 -> False
(0, 3, 17, 10, 15, 18, 7, 12, 9, 16, 2, 5, 8, 1, 11, 19, 6, 4, 14, 13, 0) -> 5.141 -> True
(0, 17, 14, 4, 18, 13, 6, 1, 12, 19, 11, 9, 7, 15, 5, 16, 2, 10, 3, 8, 0) -> 8.841 -> False
(0, 3, 17, 14, 13, 6, 11, 9, 8, 2, 5, 1, 16, 12, 7, 15, 18, 10, 4, 19, 0) -> 4.944 -> True
(0, 11, 9, 5, 15, 18, 17, 12, 13, 6, 7, 16, 3, 19, 4, 2, 1, 14, 10, 8, 0) -> 11.381 -> False
(0, 9, 11, 16, 15, 14, 8, 19, 4, 6, 12, 2, 7, 1, 18, 13, 5, 10, 3, 17, 0) -> 11.382 -> False
(0, 15, 3, 9, 11, 2, 10, 7, 17, 1, 14, 6, 13, 19, 8, 5, 18, 12, 4, 16, 0) -> 12.502 -> False
(0, 3, 11, 4, 14, 13, 6, 1, 8, 5, 16, 2, 12, 9, 15, 7, 18, 10, 17, 19, 0) -> 5.616 -> True
(0, 19, 11, 12, 9, 8, 1, 5, 2, 16, 7, 15, 18, 10, 6, 4, 13, 14, 17, 3, 0) -> 4.570 -> True
(0, 19, 1, 5, 8, 16, 9, 12, 7, 15, 18, 4, 10, 2, 11, 17, 13, 14, 3, 6, 0) -> 5.651 -> True
(0, 3, 13, 14, 11, 9, 16, 2, 8, 5, 1, 12, 7, 15, 18, 10, 6, 4, 17, 19, 0) -> 5.188 -> True
(0, 18, 9, 5, 19, 4, 3, 17, 16, 6, 14, 7, 13, 15, 1, 12, 2, 10, 8, 11, 0) -> 11.196 -> False
```

Those are bad inputs as such; we will represent them as "this vertex is visited in this position" instead

(0, 4, 2, 3, 1, 0)


[0, 4, 2, 3, 1]
Input
Kernel weights
Hidden
Output



Epoch: execute an *optimizer*,
processing data by *batches*,
updating *weights* after each batch

Prediction

```
for BRW wanted False got False (raw: 0.248)
for RW0 wanted False got False (raw: 0.062)
for RW1 wanted False got False (raw: 0.090)
for RW2 wanted False got False (raw: 0.020)
for RW3 wanted False got False (raw: 0.199)
for RW4 wanted False got True  (raw: 0.559)
for MST wanted True  got True  (raw: 0.900)
for SA  wanted True  got True  (raw: 0.846)
```

that kinda works

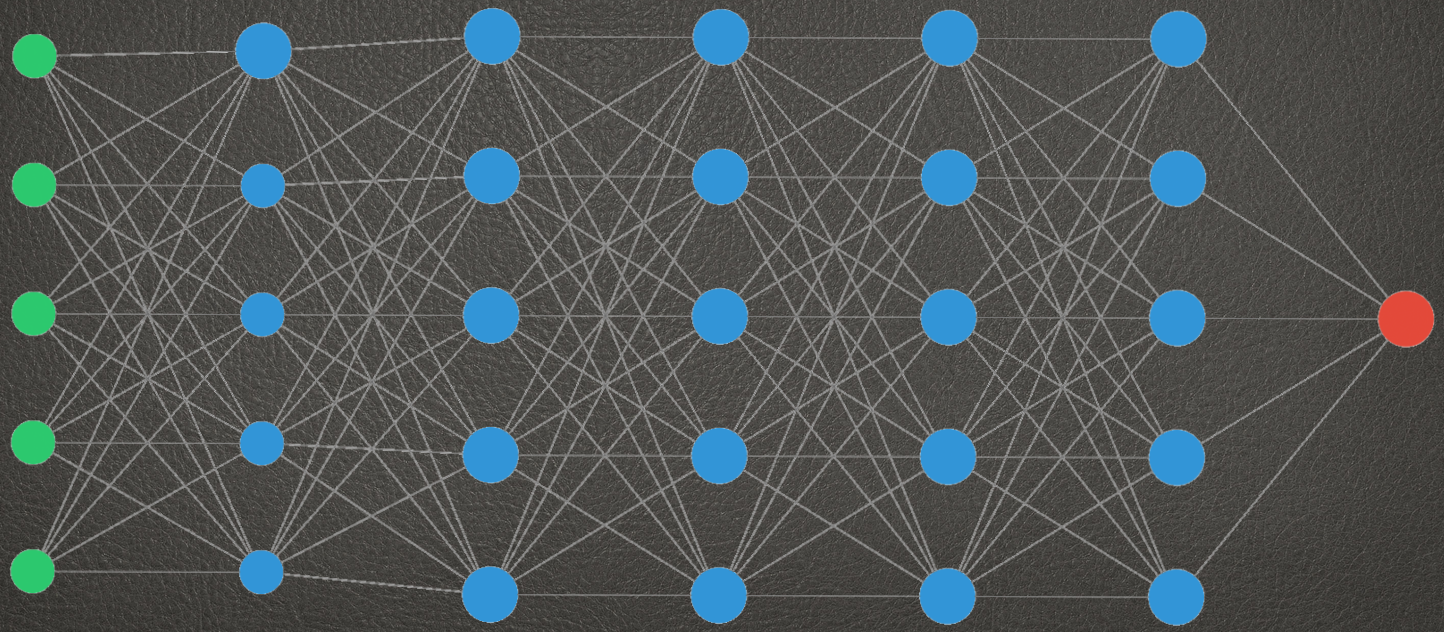
Are the kernel weights by chance the edges?

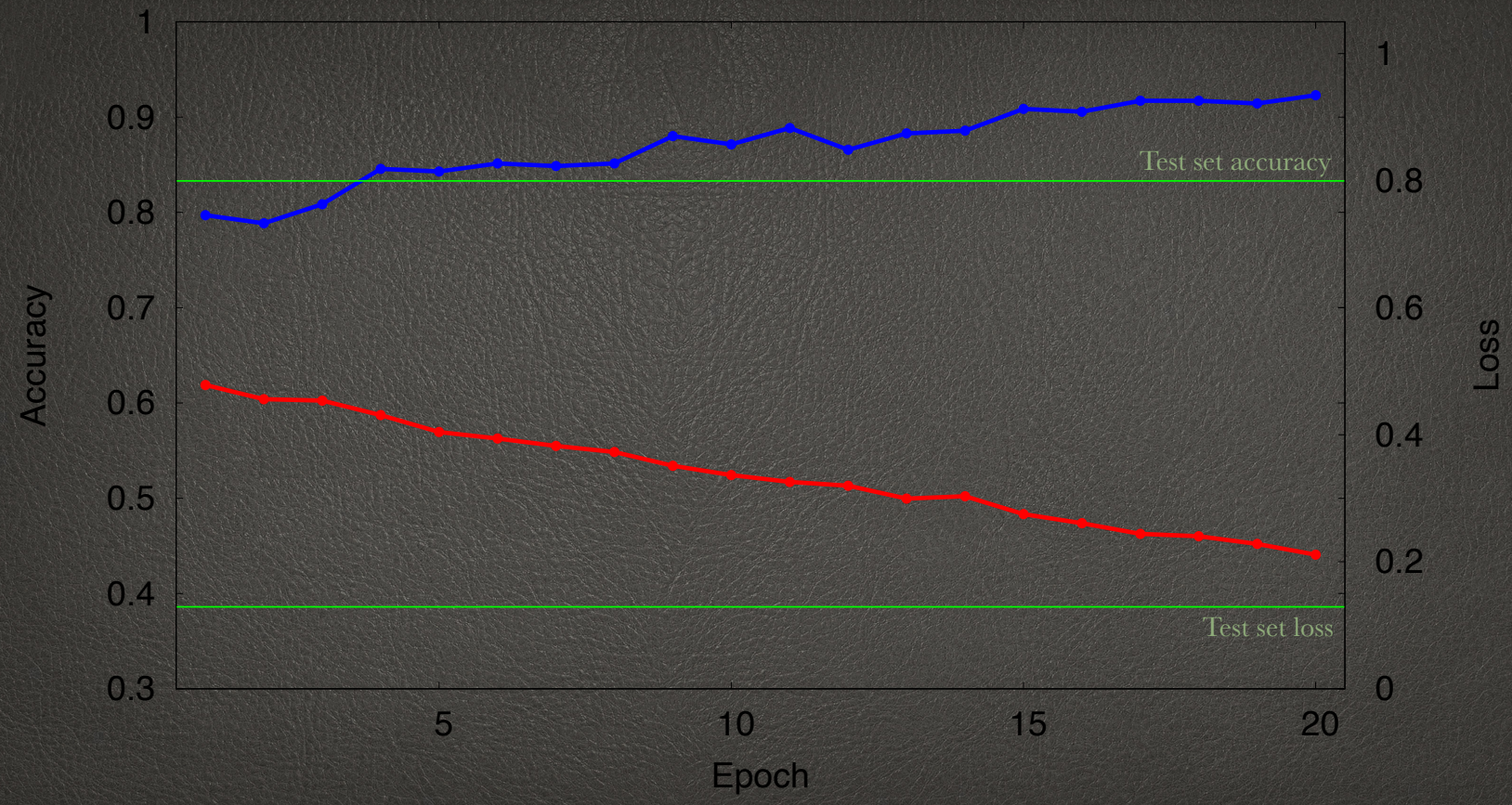
Distance matrix Kernel weights

Both normalized to [0, 1] for comparability

Pearson correlation -0.004 😺

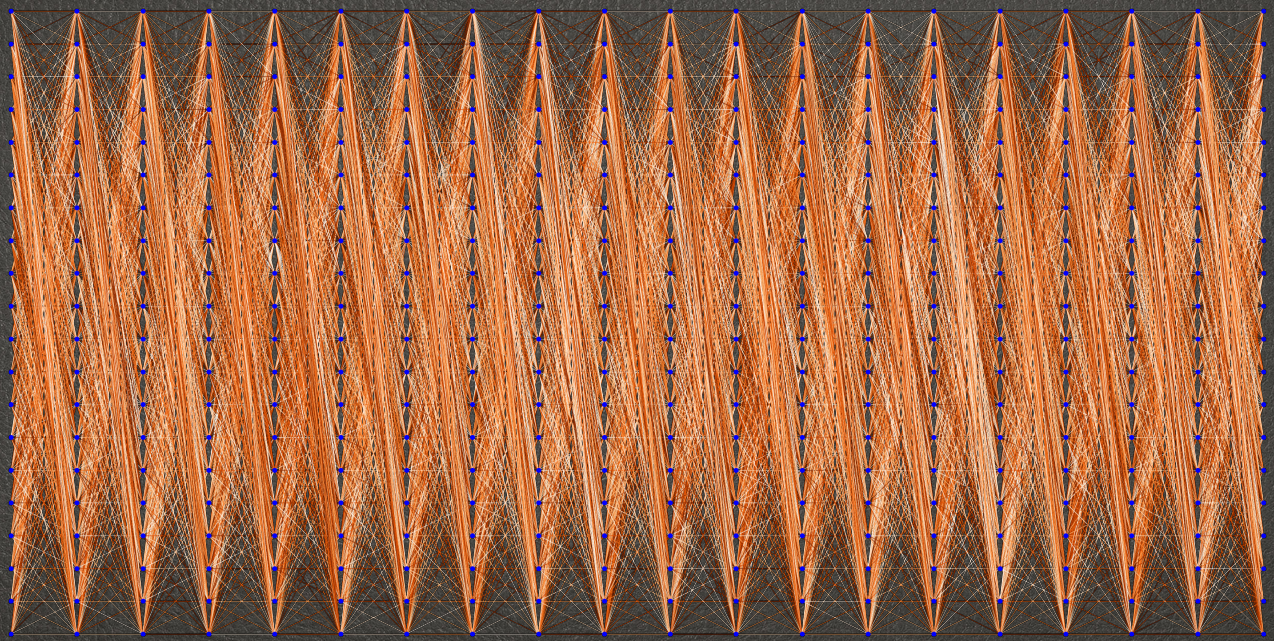
How about a second model, then?




1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Accuracy
5
10
15
20
Epoch
1
0.8
0.6
0.4
0.2
0
Loss
Test set accuracy
Test set loss

Noice

What are the kernel weights now?



https://tinyurl.com/tspdemocode