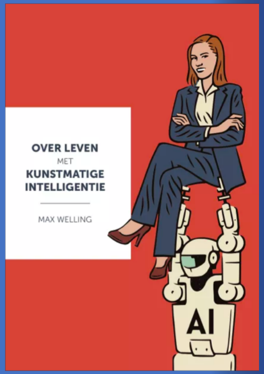
# Learning to Solve OR Problems


Max Welling

University of Amsterdam

Qualcomm Technologies

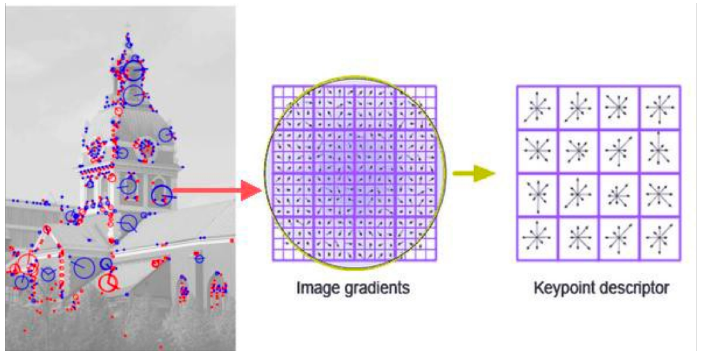




Joint work with Wouter Kool
PhD student AMLAB
(thanks to Wouter for slides)

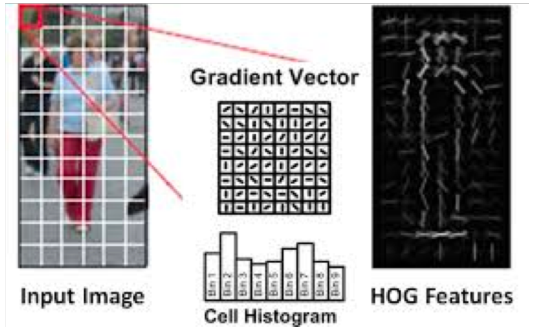
# Outline

- Motivation: man versus machine
- Intro Reinforcement learning
- Solving the TSP with RL
- Afterthoughts

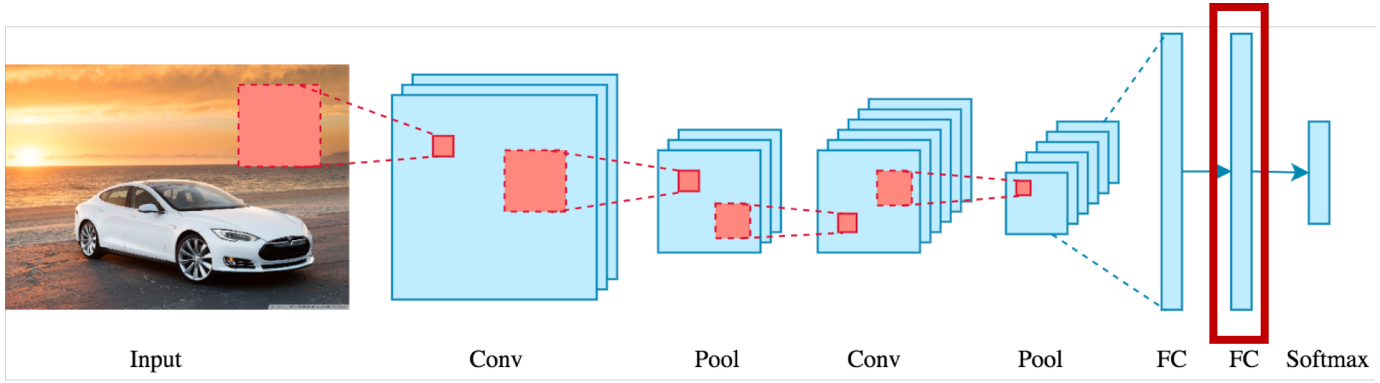
# What happened around 2010?




Sift features



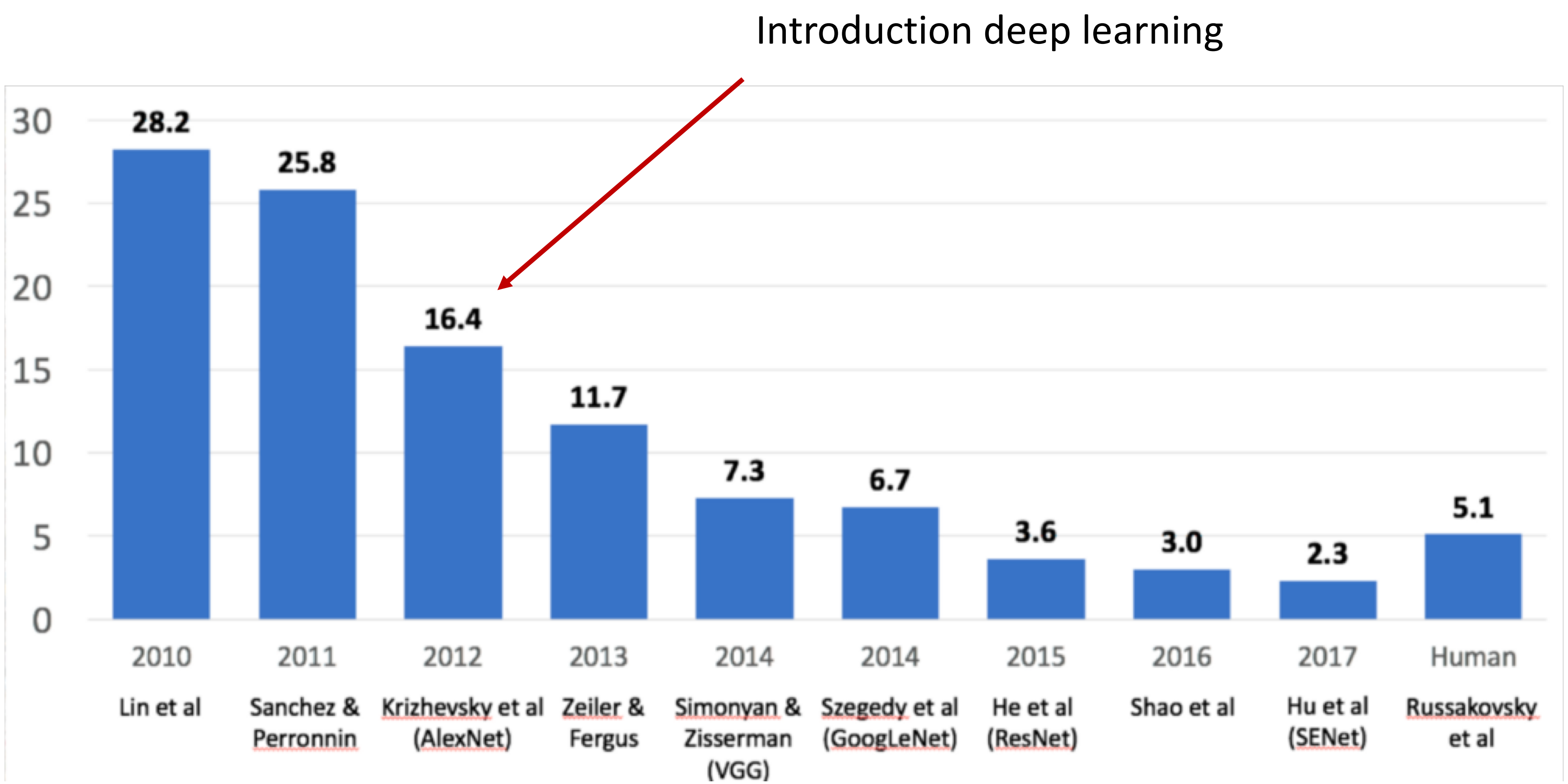

HOG features




Learned features

hand designed features → learned features

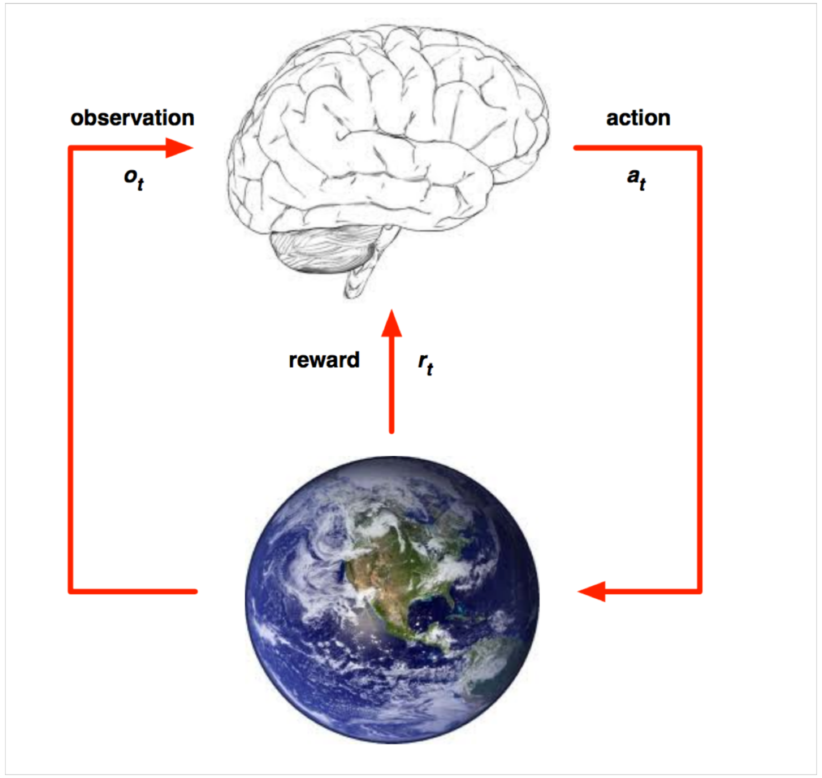
# ImageNet Competition



# What can we learn from this in OR?

- What are the equivalent of “hand-designed designed features” in OR?
- Can we replace those hand-designed components by learning them?

# Introduction Reinforcement Learning




- Agent acts in the real world
- Agents tries to maximize total future reward
- Environment delivers back observations and reward signal
- Tradeoff between information gathering (exploration) and maximizing immediate reward (exploitation).

# Bellman Equation

For a given policy, compute the value V(s) of each state:

Discount factor <1 to discount future

$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right]$$

Reward received for transition s→s' under action a.

Policy: probability of taking action "a" in state "s").

Transition probability for moving to state s', given state s and action a.

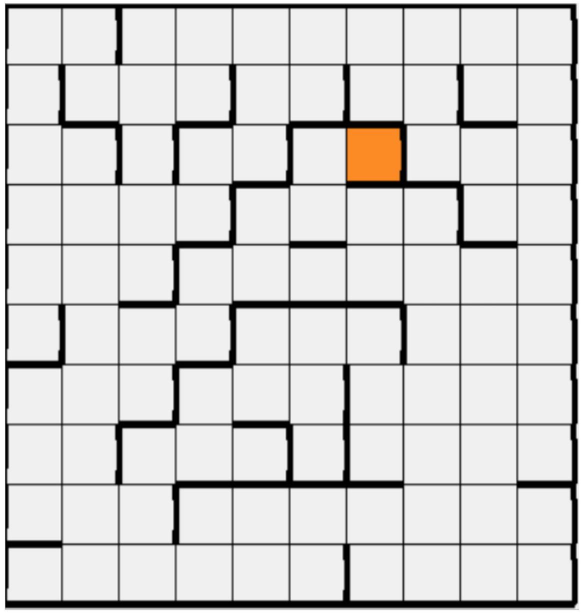
# Policy Improvement

Given optimal values for given policy, choose the policy that moves you to the state with highest value:

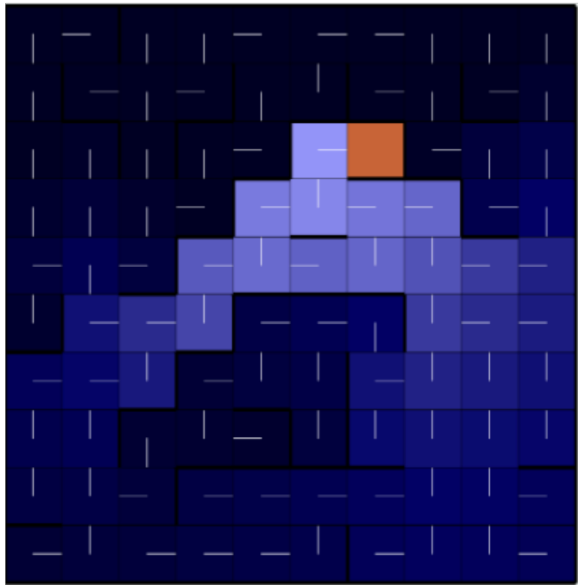
$$\pi(a|s) = \arg\max_a \underbrace{\sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right]}_{\text{Starting in state ’s’, average value of next state ‘s’’ if you take action ‘a’.}}$$

This process converges!

# Intuition

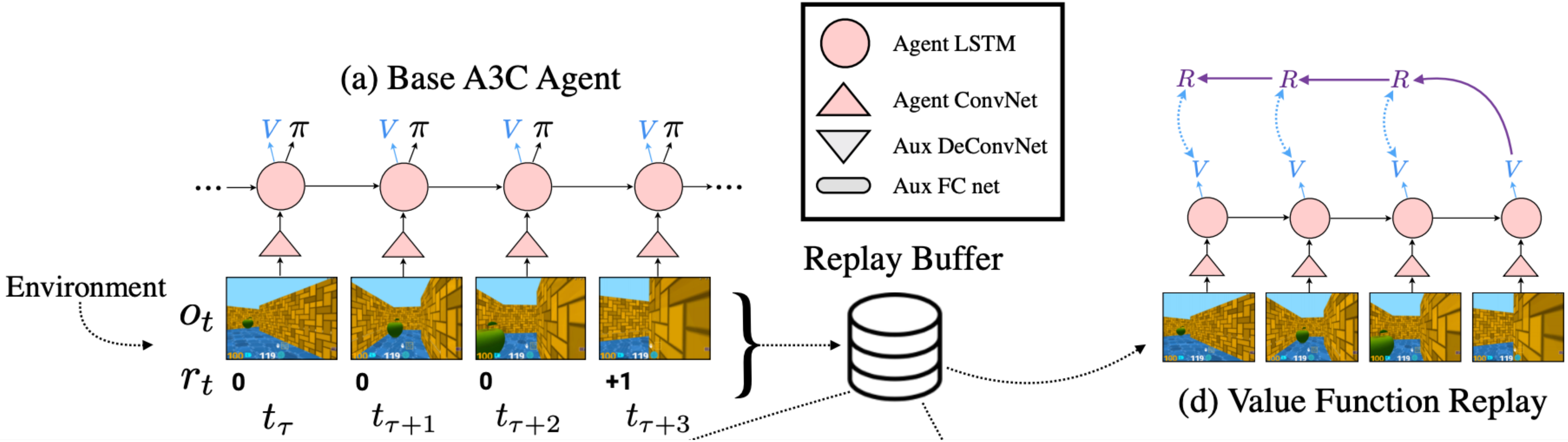


maze



values / policy

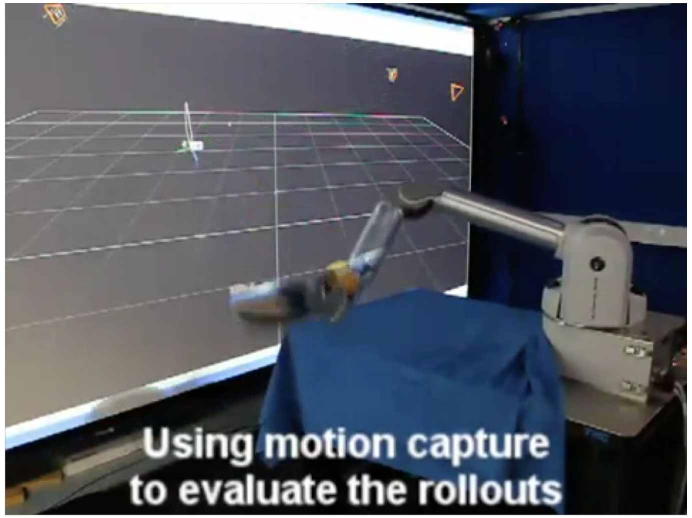
# Modern "Deep RL" a lot more sophisticated




(From Jaderberg et al, 2016)

- Input sequence gets analyzed by CNN
- Data (experiences) are stored in a replay buffer
- Both value and policy are predicted by NN
- State transitions modeled by LSTM
- Future rewards are recorded as targets for V
- Policy is trained with policy-gradient

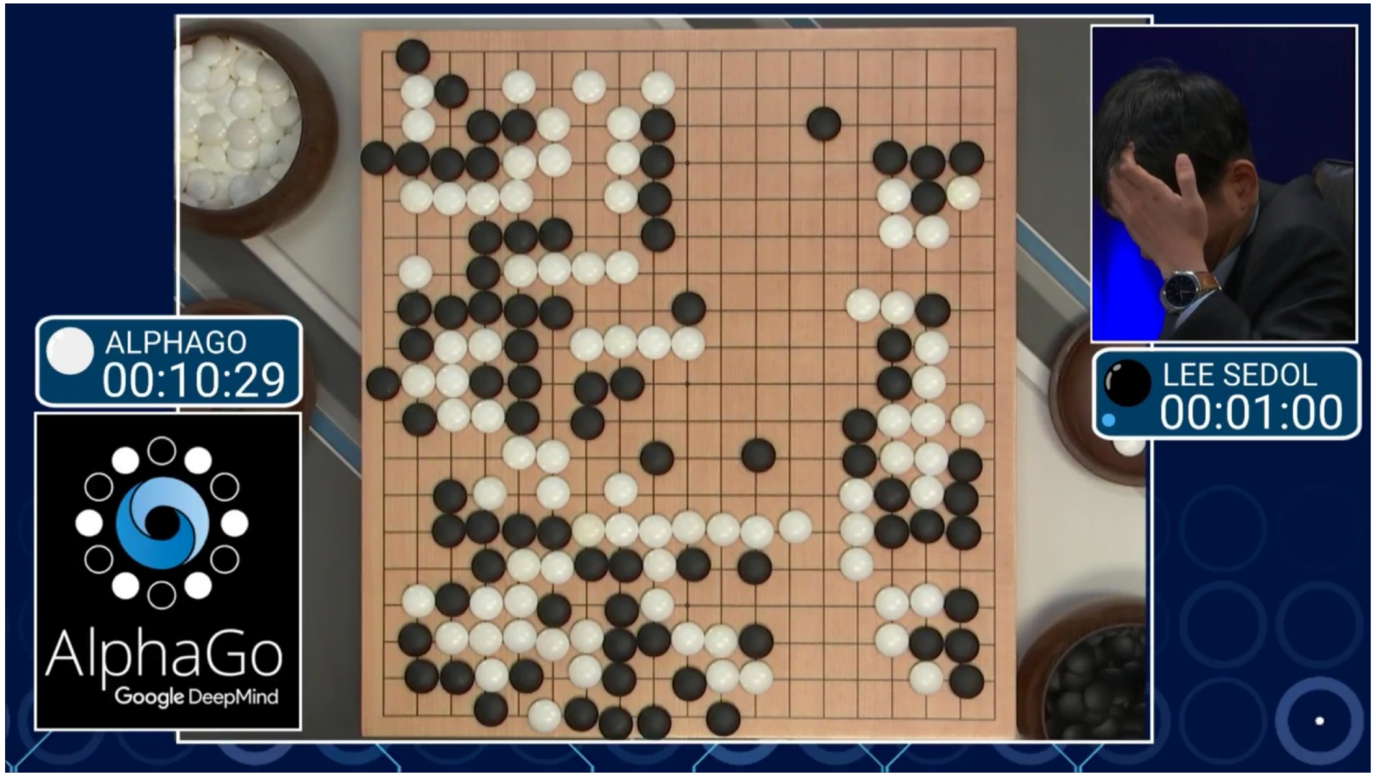
# Pancake Flipping Demonstration


Petar Kormushev, Silvain Calinon, Sarwin Caldwell, Italian Institute of Technology

# AlphaGO: Man against Machine 1-4

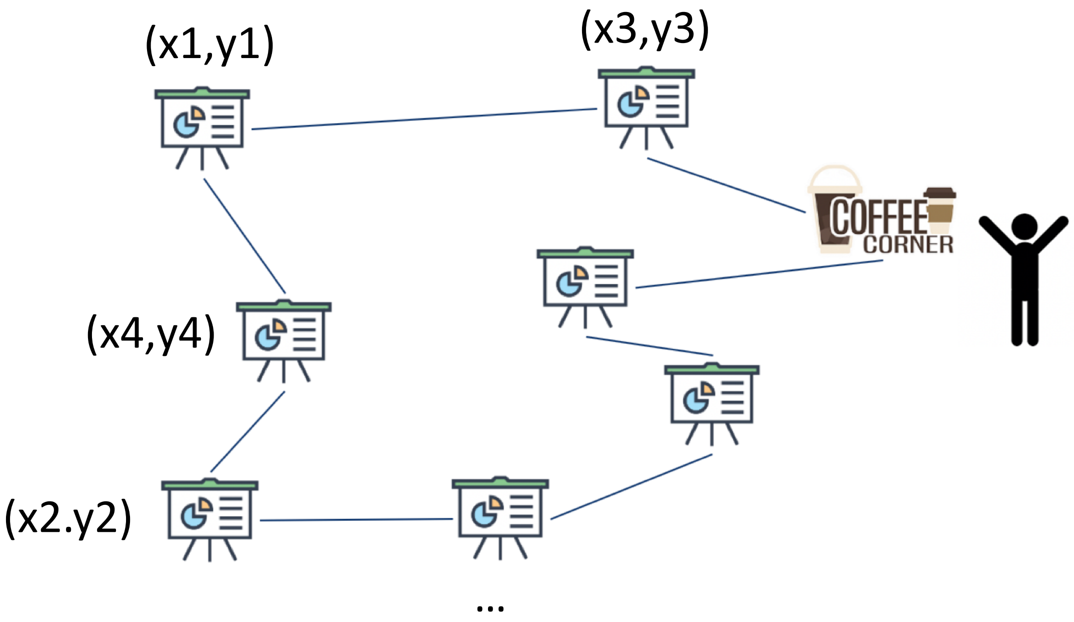
The current best reinforcement learning system




AlphaZero only played against itself and became better than the best human overnight

# Traveling Scientist Problem

Kool et al , ICLR 2019




$(x_1, y_1)$ $(x_2, y_2)$ $(x_3, y_3)$ $(x_4, y_4)$ …. → $(x_3, y_3)$ $(x_1, y_1)$ $(x_4, y_4)$ $(x_2, y_2)$ ….

Learn Policy, $\pi(a|s) \Rightarrow P(\text{next node is } i|\text{previous nodes})$

by generating lots of example trajectories

# Model

Confusion alert: in these slides $\pi$ is not the policy but the tour and 's' is the instance TSP

- Instance $s = ((x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4))$
- Solution $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots)$ e.g. $(3,1,4,2)$
- Model $p(\boldsymbol{\pi}|s) = p(\pi_1, \pi_2, \dots |s)$

Factorize!

$$= p(\pi_1|s)p(\pi_2|s, \pi_1)p(\pi_3|s, \pi_1, \pi_2) \dots$$

$$= \prod_{j=1}^{n} p(\pi_j|s, \pi_{j'}: j' < j)$$

$$\boxed{p_{\theta}(\pi_j|s, \pi_{<j})} = p_{\theta}(\text{next node} \mid \text{partial tour})$$

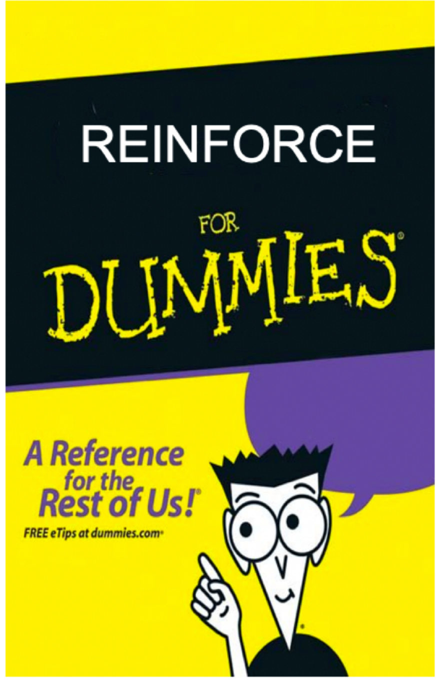
# Randomized algorithm

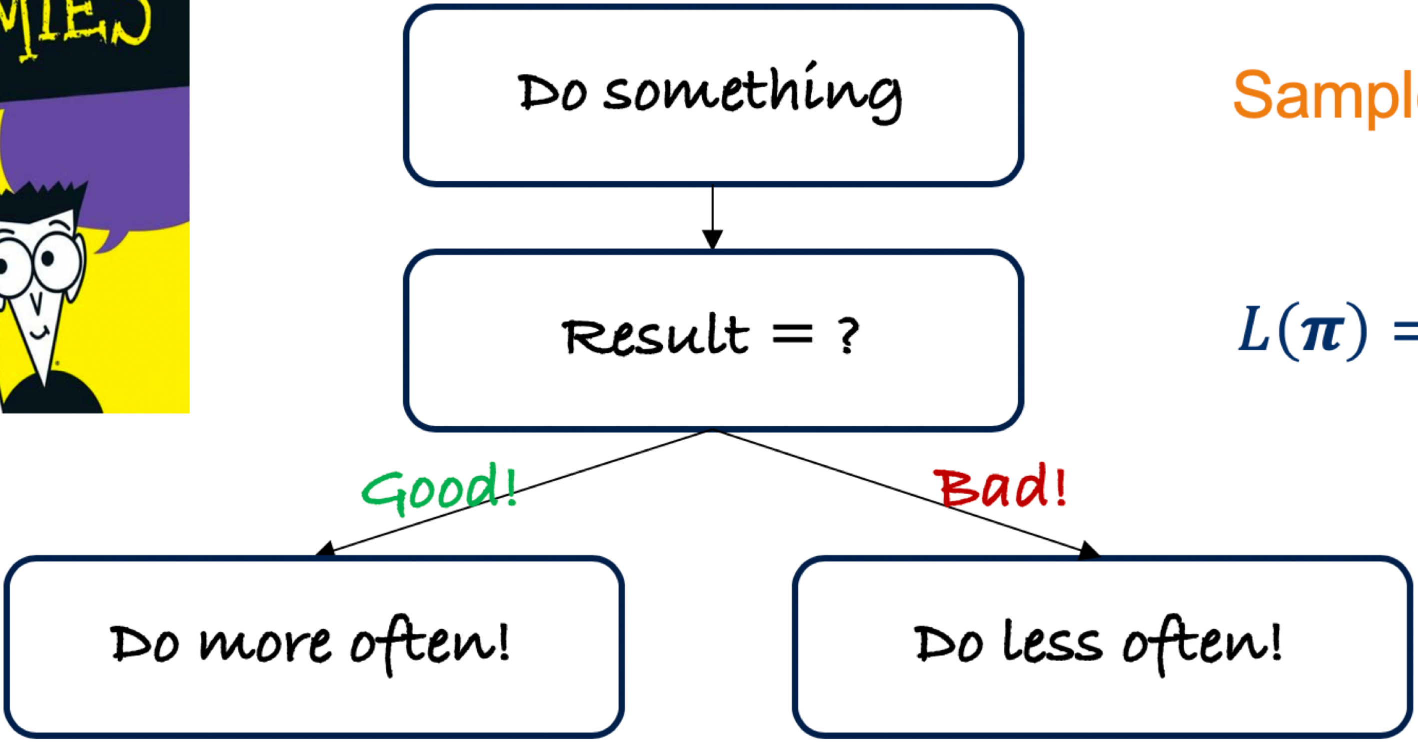
- Sample $\pi_1 \sim p_{\boldsymbol{\theta}}(\pi_1 | s)$
- Sample $\pi_2 \sim p_{\boldsymbol{\theta}}(\pi_2 | s, \pi_1)$
- Sample $\pi_3 \sim p_{\boldsymbol{\theta}}(\pi_3 | s, \pi_1, \pi_2)$
- Etc…
- With tour length $L(\boldsymbol{\pi})$ expected cost of solution:

$$E_{p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)}[L(\boldsymbol{\pi})]$$

How to optimize $\boldsymbol{\theta}$?

Cannot ‘backprop through expectation’!




Sample $\boldsymbol{\pi} \sim p_{\theta}(\cdot\,|s)$

$L(\boldsymbol{\pi}) = 7.43$

We need a *baseline* to compare against

Increase $p_{\theta}(\boldsymbol{\pi}|s)$ ⇧

Decrease $p_{\theta}(\boldsymbol{\pi}|s)$ ⇩

# The Rollout Baseline


Use (rollout) the model but *greedy* instead of sampling!

Sample $\boldsymbol{\pi} \sim p_{\boldsymbol{\theta}}(\cdot\,|s)$

Rollout $\boldsymbol{\pi}^{bl} \sim p_{\boldsymbol{\theta}^{bl}}(\cdot\,|s)$ **(greedy!)**

$L(\boldsymbol{\pi}) < L(\boldsymbol{\pi}^{bl})$ Good!

$L(\boldsymbol{\pi}) > L(\boldsymbol{\pi}^{bl})$ Bad!

Adjust $p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)$ proportional to $L(\boldsymbol{\pi}) - L(\boldsymbol{\pi}^{bl})$

# Learning Algorithm (roughly)

Init $\boldsymbol{\theta}, \boldsymbol{\theta}^{bl} \leftarrow \boldsymbol{\theta}$

For ever(y epoch):

For iteration:

Sample $s$

Sample $\boldsymbol{\pi} \sim p_{\boldsymbol{\theta}}(\cdot\,|s)$

Rollout $\boldsymbol{\pi}^{bl} \sim p_{\boldsymbol{\theta}^{bl}}(\cdot\,|s)$ **(greedy!)**

Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\,\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)\left(L(\boldsymbol{\pi}) - L(\boldsymbol{\pi}^{bl})\right)$

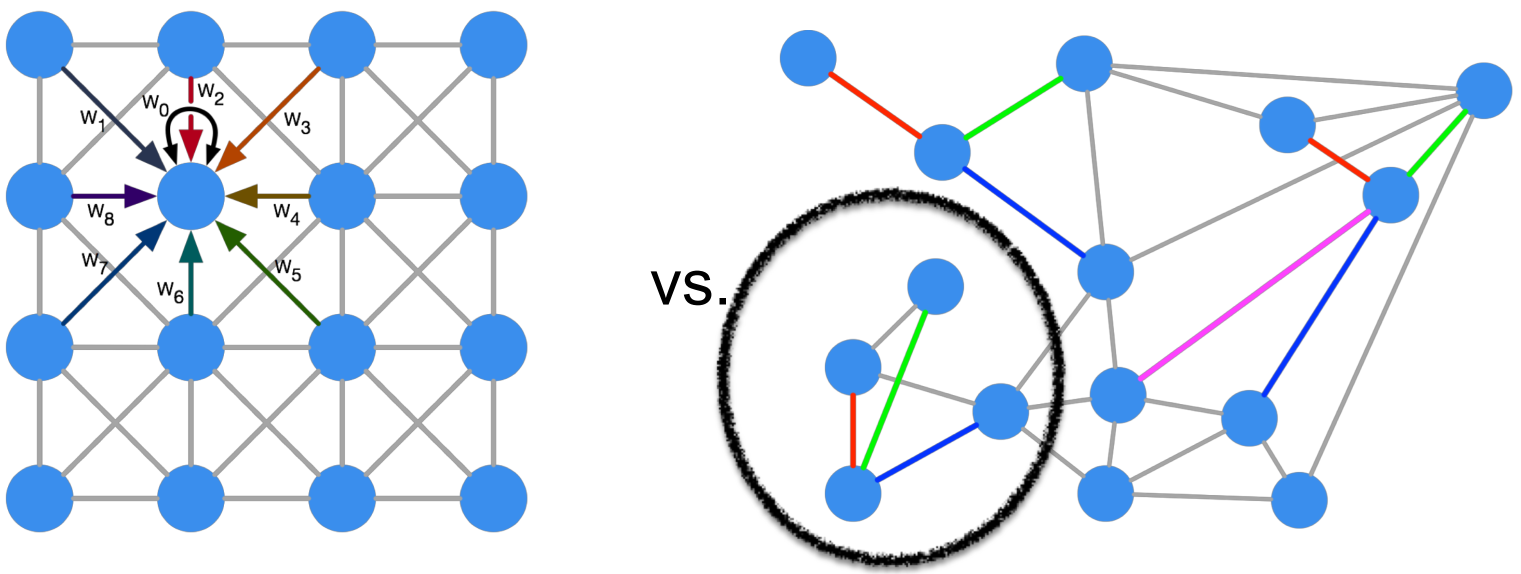
If $\boldsymbol{\theta}$ better* than $\boldsymbol{\theta}^{bl}$

Update $\boldsymbol{\theta}^{bl} \leftarrow \boldsymbol{\theta}$

* Paired *t*-test on solution of 10 000 instances with greedy rollout

# How do we represent the policy?

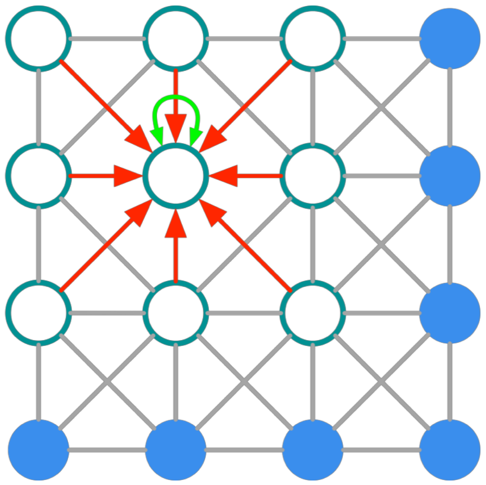
- We want to use the power of deep learning to embed the nodes (e.g. learn features).
- But, the embedding can not depend on the order of the input sequence.

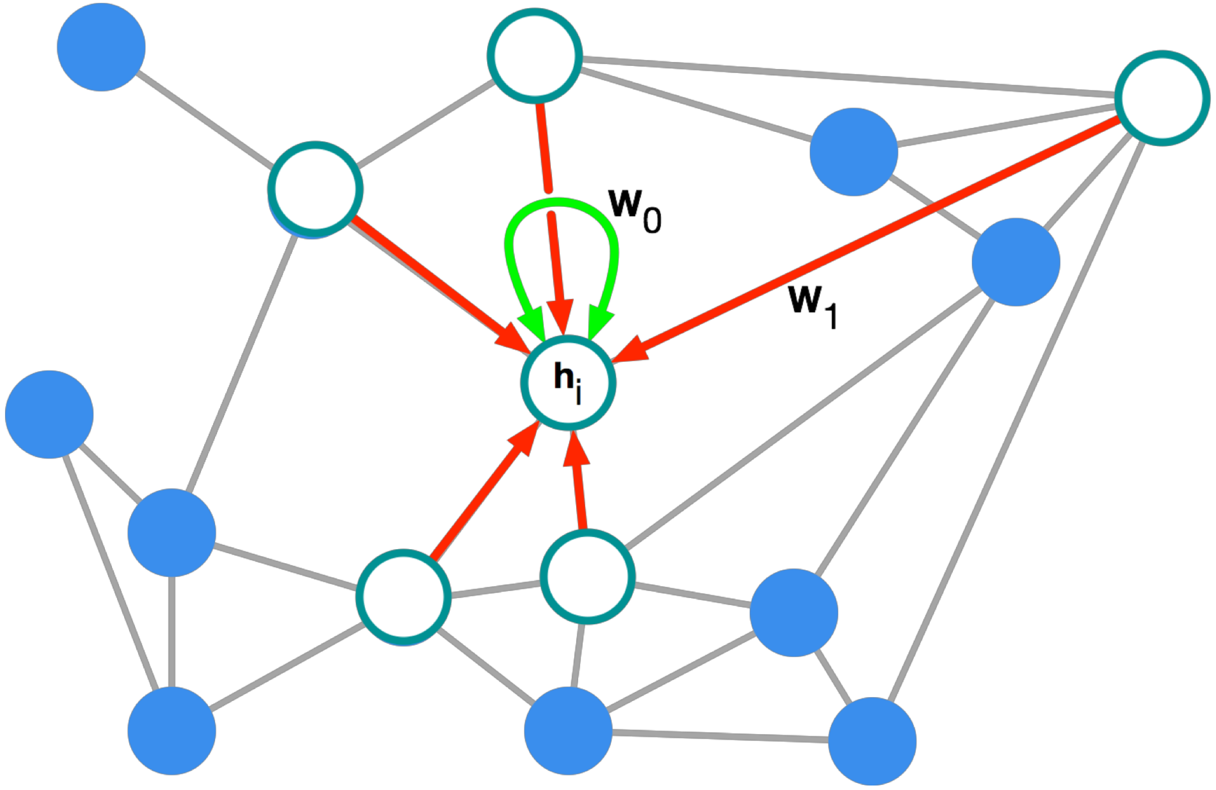



- Different number of neighbors
- No natural orientation/order of neighbors

# Graph Convolutions

Kipf & W. 2017

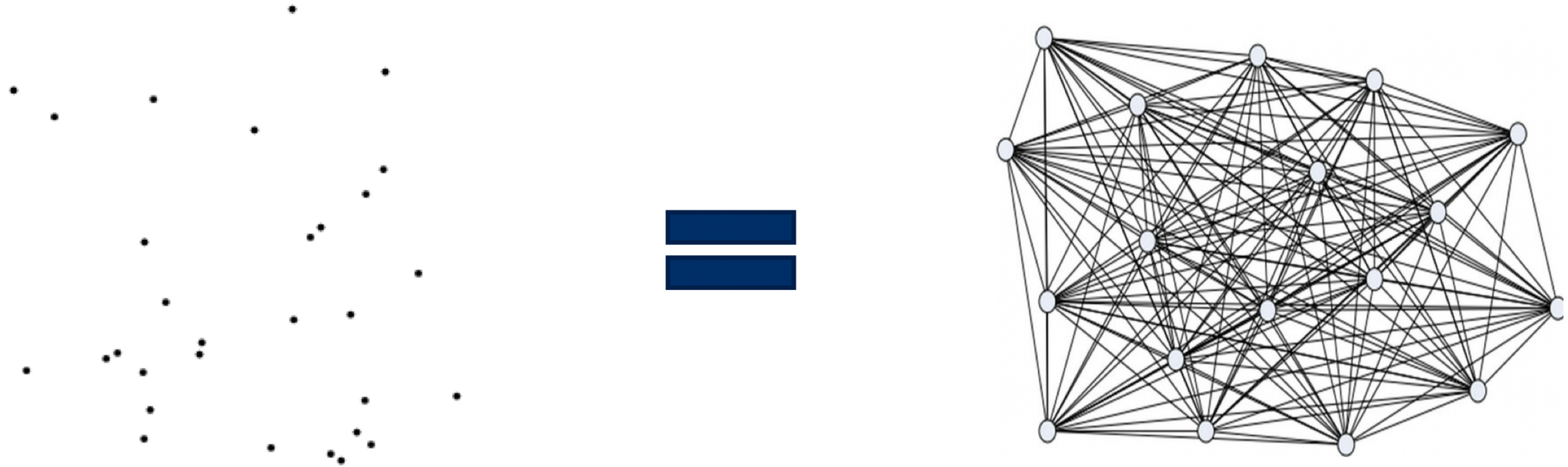




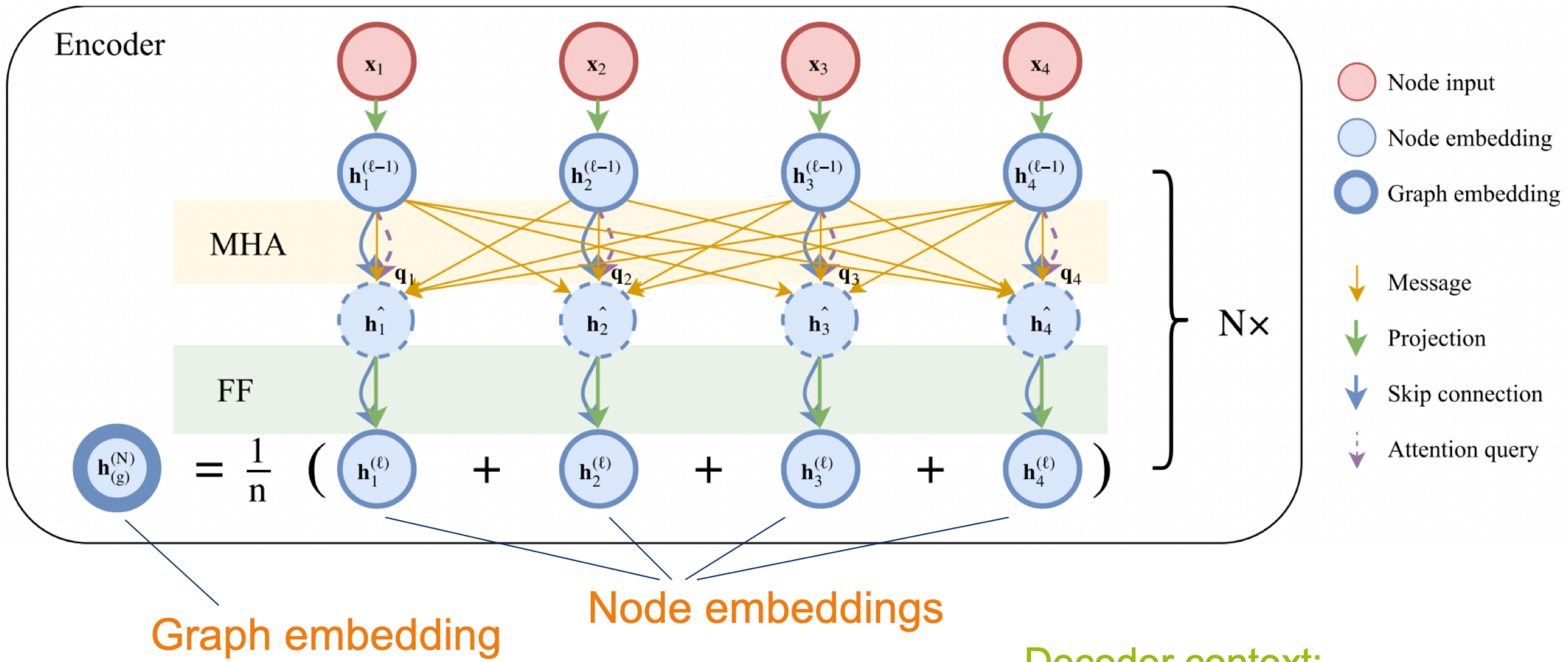

Propagation rule: $$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Use attention to compute weights

# Input as (fully connected) graph

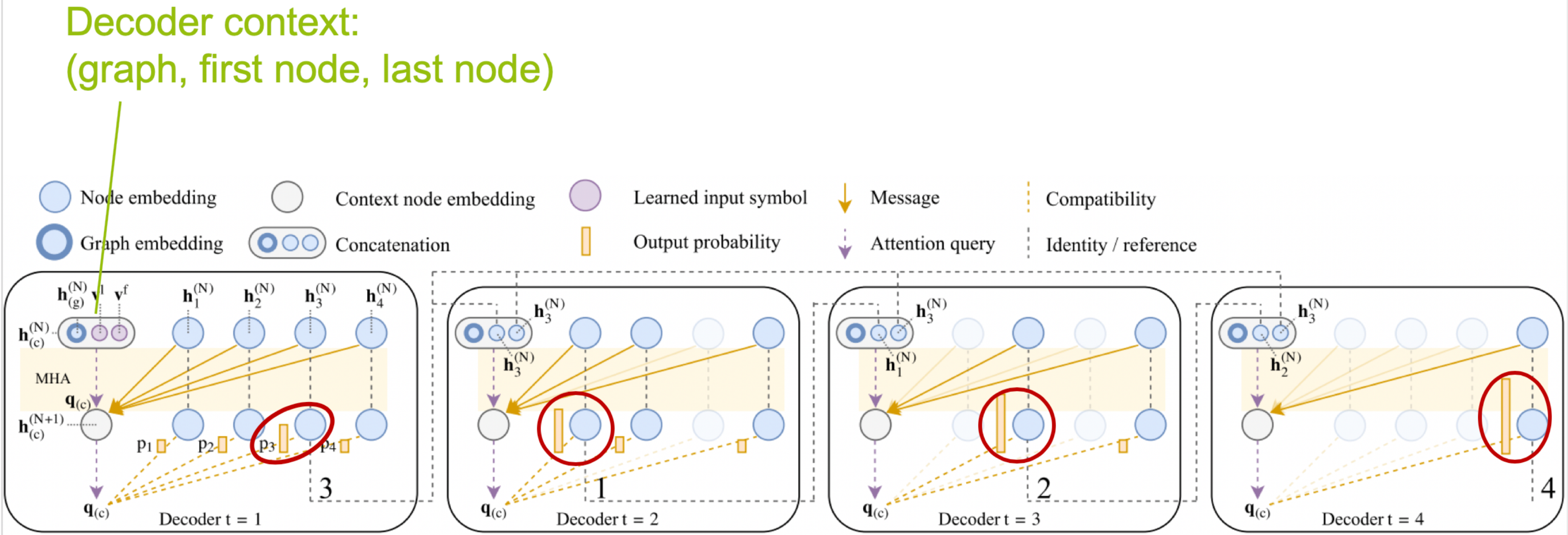


# We encode the input nodes using a graph CNN




Decoder context:
(graph, first node, last node)

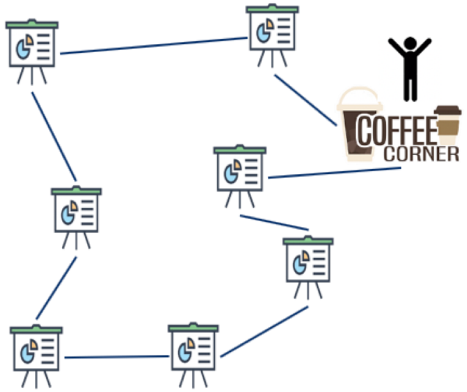
# We decode the sequence iteratively




$$\pi = (3,1,2,4)$$

# Experiments

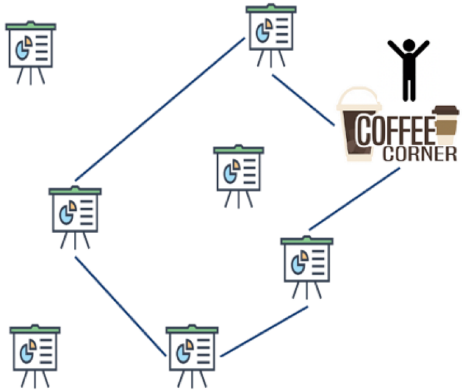
## Travelling Salesman Problem (TSP)




Minimize length

Visit all nodes

## Orienteering Problem (OP)




Maximize total prize

Max length constraint

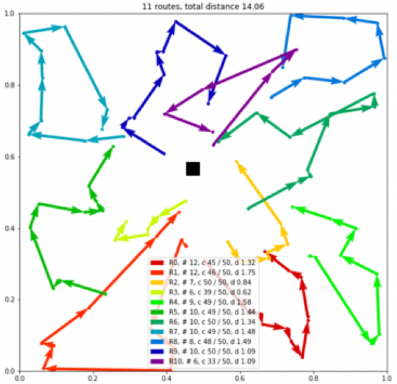
## (Stochastic) Prize Collecting TSP ((S)PCTSP)

Minimize length +

penalties of unvisited nodes

Collect minimum total prize

## Vehicle Routing Problem (VRP)




Minimize length

Visit all nodes

Total route demand must fit vehicle capacity

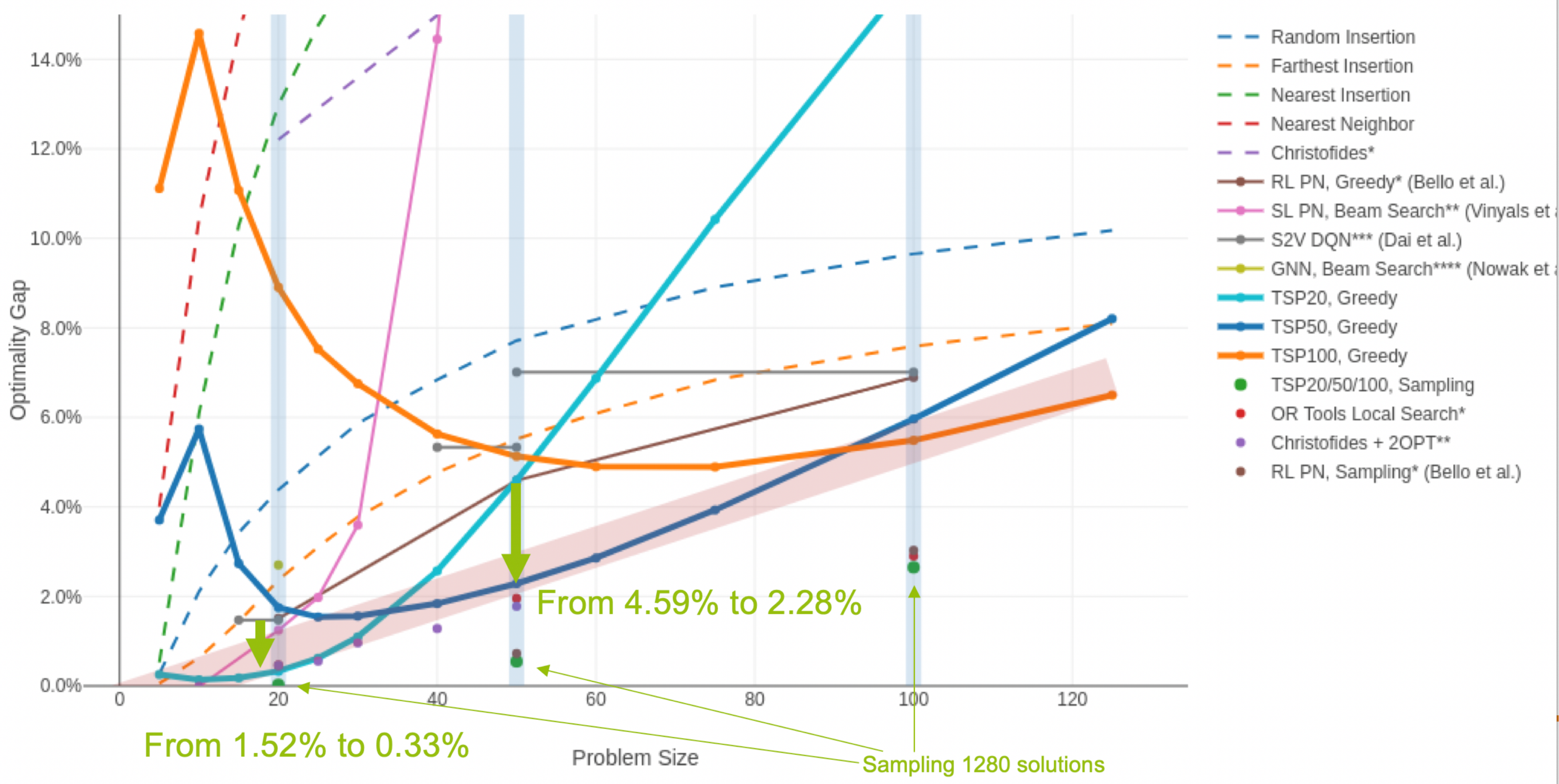
Train for each problem, *same hyperparameters*!

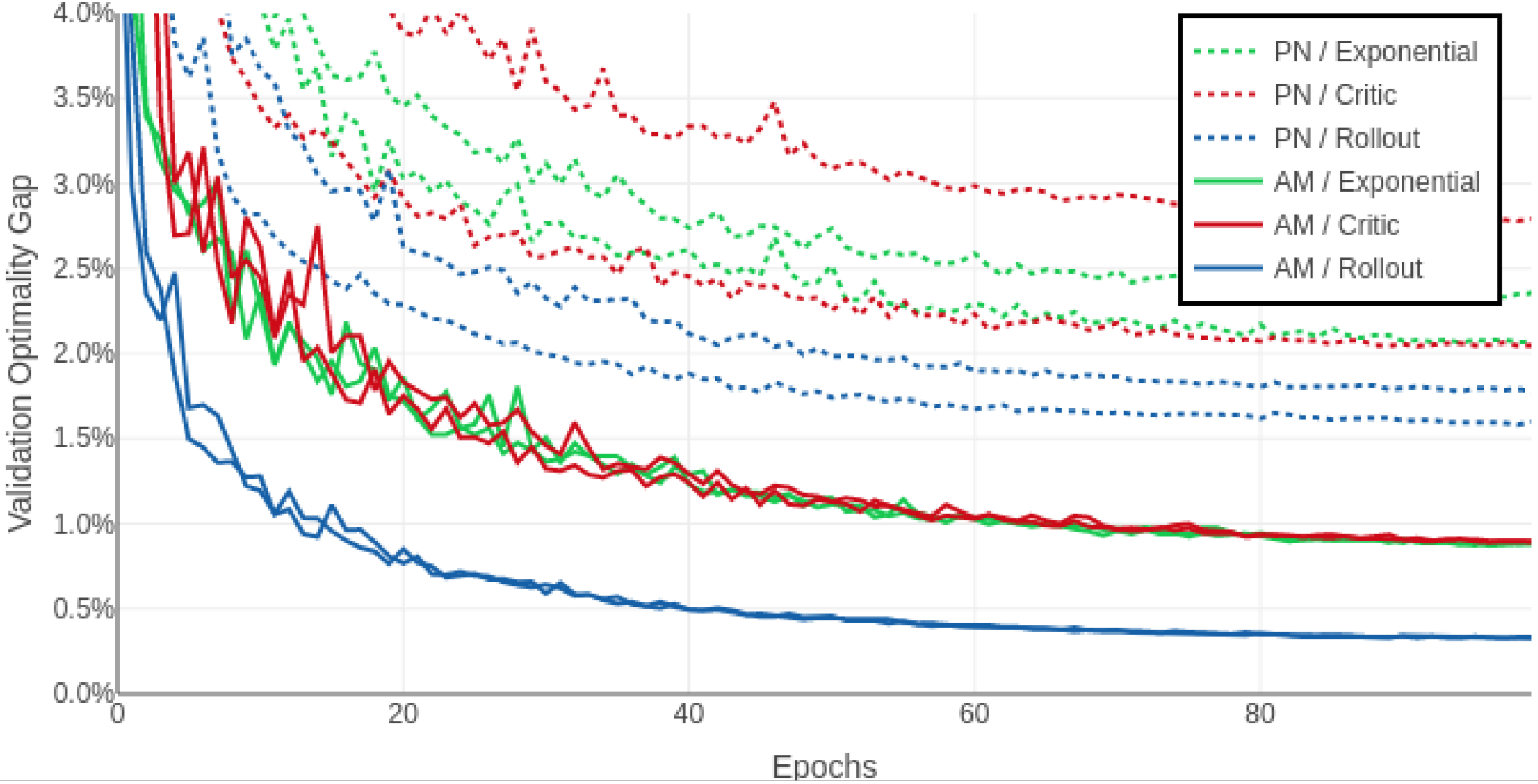
# Experimental setup

- Implementation in PYTORCH
- Use Adam optimizer, gradient clipping

- Train for $n = 20, 50, 100$
- Train $100$ epochs of $2500 \times 512 = 1\,280\,000$ instances
  - Takes 8 hours, 1 day (1GPU) and 2 days (2GPUs) respectively

- Test for various $n = 5, \ldots, 125$
- Test using greedy decoding or sampling (best of 1280)
- Compare against other approaches and heuristics


Optimality Gap
14.0%
12.0%
10.0%
8.0%
6.0%
4.0%
2.0%
0.0%
0
20
40
60
80
100
120
Problem Size
Random Insertion
Farthest Insertion
Nearest Insertion
Nearest Neighbor
Christofides*
RL PN, Greedy* (Bello et al.)
SL PN, Beam Search** (Vinyals et a
S2V DQN*** (Dai et al.)
GNN, Beam Search**** (Nowak et a
TSP20, Greedy
TSP50, Greedy
TSP100, Greedy
TSP20/50/100, Sampling
OR Tools Local Search*
Christofides + 2OPT**
RL PN, Sampling* (Bello et al.)
From 4.59% to 2.28%
From 1.52% to 0.33%
Sampling 1280 solutions


PN / Exponential
PN / Critic
PN / Rollout
AM / Exponential
AM / Critic
AM / Rollout
Validation Optimality Gap
4.0%
3.5%
3.0%
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
0
20
40
60
80
Epochs

Lots of problems can be tackled with this model!

Table 1: Attention Model (AM) vs baselines. The gap % is w.r.t. the best value across all methods.

| | Method | $n = 20$ Obj. | Gap | Time | $n = 50$ Obj. | Gap | Time | $n = 100$ Obj. | Gap | Time |
|---|---|---|---|---|---|---|---|---|---|---|
| TSP | Concorde | 3.84 | 0.00% | (1m) | 5.70 | 0.00% | (2m) | 7.76 | 0.00% | (3m) |
| | LKH3 | 3.84 | 0.00% | (18s) | 5.70 | 0.00% | (5m) | 7.76 | 0.00% | (21m) |
| | Gurobi | 3.84 | 0.00% | (7s) | 5.70 | 0.00% | (2m) | 7.76 | 0.00% | (17m) |
| | Gurobi (1s) | 3.84 | 0.00% | (8s) | 5.70 | 0.00% | (2m) | | - | |
| | Nearest Insertion | 4.33 | 12.91% | (1s) | 6.78 | 19.03% | (2s) | 9.46 | 21.82% | (6s) |
| | Random Insertion | 4.00 | 4.36% | (0s) | 6.13 | 7.65% | (1s) | 8.52 | 9.69% | (3s) |
| | Farthest Insertion | 3.93 | 2.36% | (1s) | 6.01 | 5.53% | (2s) | 8.35 | 7.59% | (7s) |
| | Nearest Neighbor | 4.50 | 17.23% | (0s) | 7.00 | 22.94% | (0s) | 9.68 | 24.73% | (0s) |
| | Vinyals et al. (gr.) | 3.88 | 1.15% | | 7.66 | 34.48% | | | - | |
| | Bello et al. (gr.) | 3.89 | 1.42% | | 5.95 | 4.46% | | 8.30 | 6.90% | |
| | Dai et al. | 3.89 | 1.42% | | 5.99 | 5.16% | | 8.31 | 7.03% | |
| | Nowak et al. | 3.93 | 2.46% | | | - | | | - | |
| | EAN (greedy) | 3.86 | 0.66% | (2m) | 5.92 | 3.98% | (5m) | 8.42 | 8.41% | (8m) |
| | AM (greedy) | **3.85** | **0.34%** | (0s) | **5.80** | **1.76%** | (2s) | **8.12** | **4.53%** | (6s) |
| | OR Tools | 3.85 | 0.37% | | 5.80 | 1.83% | | 7.99 | 2.90% | |
| | Chr.f. + 2OPT | 3.85 | 0.37% | | 5.79 | 1.65% | | | - | |
| | Bello et al. (s.) | | - | | 5.75 | 0.95% | | 8.00 | 3.03% | |
| | EAN (gr. + 2OPT) | 3.85 | 0.42% | (4m) | 5.85 | 2.77% | (26m) | 8.17 | 5.21% | (3h) |
| | EAN (sampling) | 3.84 | 0.11% | (5m) | 5.77 | 1.28% | (17m) | 8.75 | 12.70% | (56m) |
| | EAN (s. + 2OPT) | 3.84 | 0.09% | (6m) | 5.75 | 1.00% | (32m) | 8.12 | 4.64% | (5h) |
| | AM (sampling) | **3.84** | **0.08%** | (5m) | **5.73** | **0.52%** | (24m) | **7.94** | **2.26%** | (1h) |
| CVRP | Gurobi | 6.10 | 0.00% | | | - | | | - | |
| | LKH3 | 6.14 | 0.58% | (2h) | 10.38 | 0.00% | (7h) | 15.65 | 0.00% | (13h) |
| | RL (greedy) | 6.59 | 8.03% | | 11.39 | 9.78% | | 17.23 | 10.12% | |
| | AM (greedy) | **6.40** | **4.97%** | (1s) | **10.98** | **5.86%** | (3s) | **16.80** | **7.34%** | (8s) |
| | RL (beam 10) | 6.40 | 4.92% | | 11.15 | 7.46% | | 16.96 | 8.39% | |
| | Random CW | 6.81 | 11.64% | | 12.25 | 18.07% | | 18.96 | 21.18% | |
| | Random Sweep | 7.08 | 16.07% | | 12.96 | 24.91% | | 20.33 | 29.93% | |
| | OR Tools | 6.43 | 5.41% | | 11.31 | 9.01% | | 17.16 | 9.67% | |
| | AM (sampling) | **6.25** | **2.49%** | (6m) | **10.62** | **2.40%** | (28m) | **16.23** | **3.72%** | (2h) |
| SDVRP | RL (greedy) | 6.51 | 4.19% | | 11.32 | 6.88% | | 17.12 | 5.23% | |
| | AM (greedy) | **6.39** | **2.34%** | (1s) | **10.92** | **3.08%** | (4s) | **16.83** | **3.42%** | (11s) |
| | RL (beam 10) | 6.34 | 1.47% | | 11.08 | 4.61% | | 16.86 | 3.63% | |
| | AM (sampling) | **6.25** | **0.00%** | (9m) | **10.59** | **0.00%** | (42m) | **16.27** | **0.00%** | (3h) |
| OP (distance) | Gurobi | 5.39 | 0.00% | (16m) | | - | | | - | |
| | Gurobi (1s) | 4.62 | 14.22% | (4m) | 1.29 | 92.03% | (6m) | 0.58 | 98.25% | (7m) |
| | Gurobi (10s) | 5.37 | 0.33% | (12m) | 10.96 | 32.20% | (51m) | 1.34 | 95.97% | (53m) |
| | Gurobi (30s) | 5.38 | 0.05% | (14m) | 13.57 | 16.09% | (2h) | 3.23 | 90.28% | (3h) |
| | Compass | 5.37 | 0.36% | (2m) | 16.17 | 0.00% | (5m) | 33.19 | 0.00% | (15m) |
| | Tsili (greedy) | 4.08 | 24.25% | (4s) | 12.46 | 22.94% | (4s) | 25.69 | 22.59% | (5s) |
| | AM (greedy) | **5.19** | **3.64%** | (0s) | **15.64** | **3.23%** | (1s) | **31.62** | **4.75%** | (5s) |
| | GA (Python) | 5.12 | 4.88% | (10m) | 10.90 | 32.59% | (1h) | 14.91 | 55.08% | (5h) |
| | OR Tools (10s) | 4.09 | 24.05% | (52m) | | - | | | - | |
| | Tsili (sampling) | 5.30 | 1.62% | (28s) | 15.50 | 4.14% | (2m) | 30.52 | 8.05% | (6m) |
| | AM (sampling) | **5.30** | **1.56%** | (4m) | **16.07** | **0.60%** | (16m) | **32.68** | **1.55%** | (53m) |
| PCTSP | Gurobi | 3.13 | 0.00% | (2m) | | - | | | - | |
| | Gurobi (1s) | 3.14 | 0.07% | (1m) | | - | | | - | |
| | Gurobi (10s) | 3.13 | 0.00% | (2m) | 4.54 | 1.36% | (32m) | | - | |
| | Gurobi (30s) | 3.13 | 0.00% | (2m) | 4.48 | 0.03% | (54m) | | - | |
| | AM (greedy) | **3.18** | **1.62%** | (0s) | **4.60** | **2.66%** | (2s) | **6.25** | **4.46%** | (5s) |
| | ILS (C++) | 3.16 | 0.77% | (16m) | 4.50 | 0.36% | (2h) | **5.98** | **0.00%** | (12h) |
| | OR Tools (10s) | 3.14 | 0.05% | (52m) | 4.51 | 0.70% | (52m) | 6.35 | 6.21% | (52m) |
| | OR Tools (60s) | **3.13** | **0.01%** | (5h) | **4.48** | **0.00%** | (5h) | 6.07 | 1.56% | (5h) |
| | ILS (Python 10x) | 5.21 | 66.19% | (4m) | 12.51 | 179.05% | (3m) | 23.98 | 300.95% | (3m) |
| | AM (sampling) | 3.15 | 0.45% | (5m) | 4.52 | 0.74% | (19m) | 6.08 | 1.67% | (1h) |
| SPCTSP | REOPT (all) | 3.34 | 2.38% | (17m) | 4.68 | 1.04% | (2h) | 6.22 | 1.10% | (12h) |
| | REOPT (half) | 3.31 | 1.38% | (25m) | **4.64** | **0.00%** | (3h) | **6.16** | **0.00%** | (16h) |
| | REOPT (first) | 3.31 | 1.60% | (1h) | 4.66 | 0.44% | (22h) | | - | |
| | AM (greedy) | **3.26** | **0.00%** | (0s) | 4.65 | 0.33% | (2s) | 6.32 | 2.69% | (5s) |

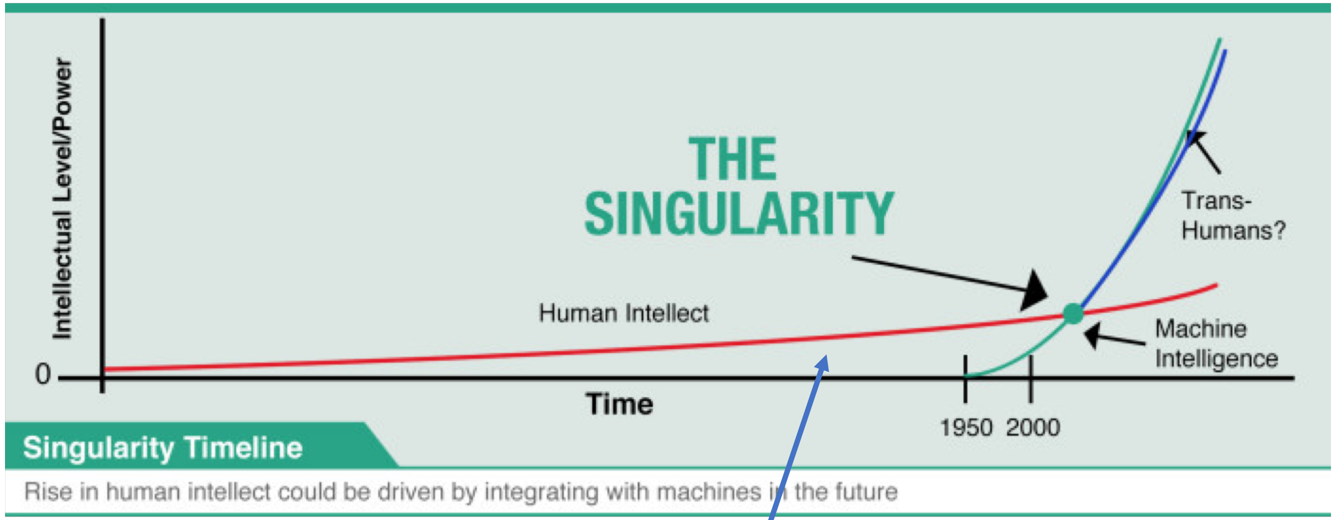
# Results Attention Model + Rollout Baseline

- Improves over classical heuristics!

- Improves over prior learned heuristics!
  - Attention Model improves
  - Rollout helps significantly

- Gets close to single-purpose SOTA (20 to 100 nodes)!
  - TSP 0.34% to 4.53% (greedy)
  - TSP 0.08% to 2.26% (best of 1280 samples)

# Afterthoughts




We are here for OR

- Learn to solve OR problems by simulating lots of examples and finding patterns.
- Right now not competitive with hand designed solvers for large problems.
- However, we can quickly generate new solutions for new problems in the same family.
- Perhaps hybrid methods will do better than either in isolation?
- In the long run, will ML overtake human designed methods (similar to computer vision)?