Utilizing Machine Learning to Enhance Anytime Tree Search Algorithms for Planning Problems

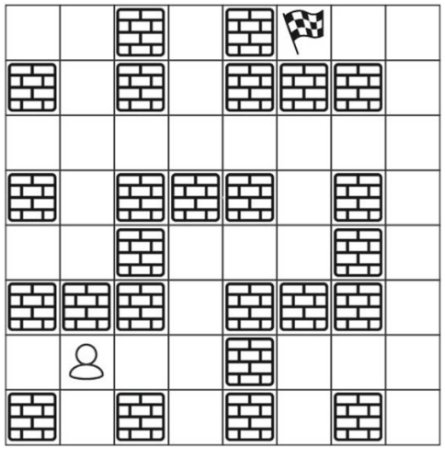
Marc Huber
mhuber@ac.tuwien.ac.at

Institute of Logic and Computation
TU Wien

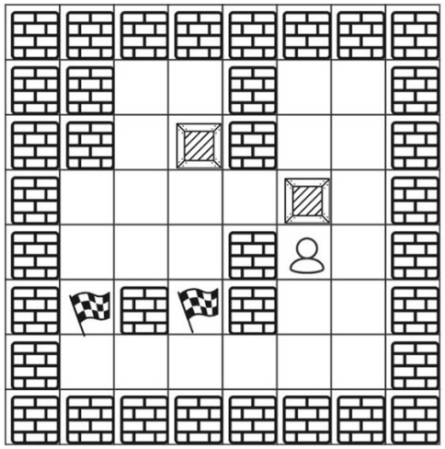
Open Problem Session
January 16, 2023

1. Motivation

Considered benchmark tasks:

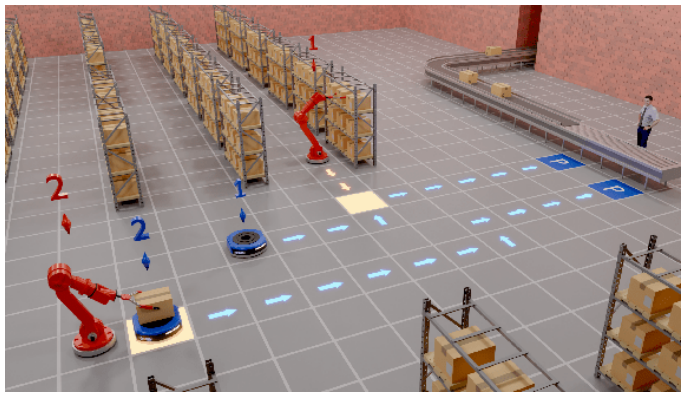


a) Maze of size 8x8



b) Sokoban with 2 boxes

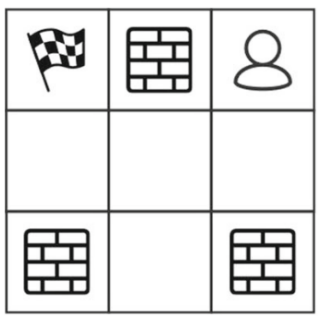
Applications:



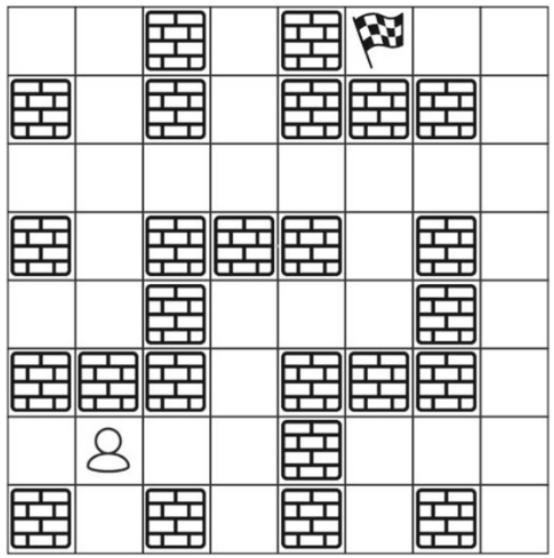
1. Motivation

Goals:

1. Learning guidance functions for anytime tree search algorithms on small-sized problem instances that scale well to different problem sizes.



a) Maze 3x3



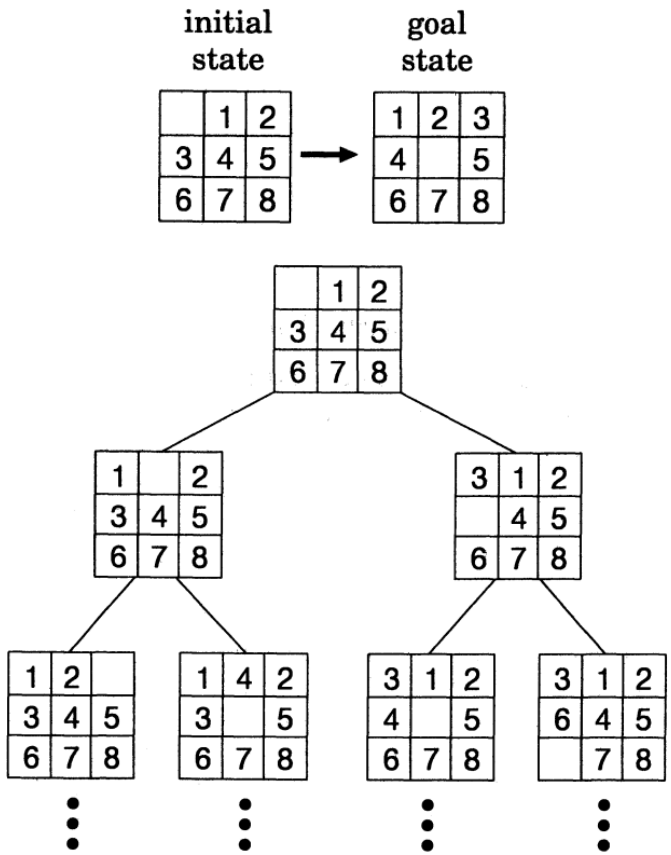
b) Maze 8x8

2. Learning admissible and consistent guidance functions for anytime tree search algorithms.

2. Introduction

Planning with state-space search:

- State-space search systematically enumerates the solutions of a problem through successive solution-space partitioning.



2. Introduction

- During the process of exploring a state space $\mathcal{S}$, a search algorithm
  - evaluates the value of the states to decide if a state needs to be examined,
  - determines the order in which the states are explored.

$\Rightarrow$ **A* Algorithm**

2. Introduction

A* Algorithm [Hart et al., 1968]:

- Popular best-first search algorithm.
- Maintains the set of generated but not expanded states $\mathcal{O}$ and the set of already expanded states $\mathcal{C}$.
- Determines the order in which states $s \in \mathcal{O}$ are expanded by

$$f(s) = g(s) + h(s),$$

  where
  - $g(s)$: exact cost from the starting point to state s.
  - $h(s)$: heuristic estimated cost from state s to the goal.

2. Introduction

$$f(s) = g(s) + h(s)$$

- If $h(s)$ is *admissible*, that is, if $h(s) \leq h^*(s)$, $\forall s \in \mathcal{S}$
  $\Rightarrow$ solution is guaranteed to be optimal.

- If $h(s)$ is *consistent*, that is, if $h(s) - h(s') \leq c(s, s')$, $\forall s, s'$, where $c(s, s')$ is the cost to transition from state s to s'
  $\Rightarrow$ a state is never expanded more than once.

2. Introduction

Weighted A* (WA*) [Pohl, 1969]:

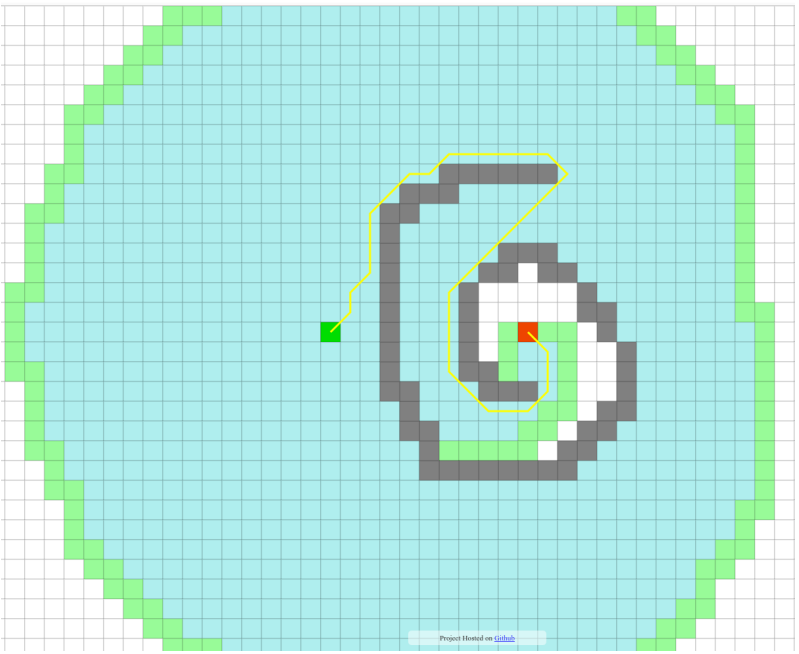
- Trades off optimality for speed.
- Expands states $s \in \mathcal{O}$ in the order of

$$f'(s) = g(s) + \epsilon \cdot h(s),$$

  where $\epsilon > 1 =$ bias towards states that are closer to goal.
- ϵ-suboptimal: $g(s^{\text{goal}}) \leq \epsilon \cdot g^*(s^{\text{goal}})$.
- In many domains, it has been shown to be orders of magnitude faster than A*.

2. Introduction

Exemplary behavior of A* and WA* algorithm:



a) A* euclidean dististance

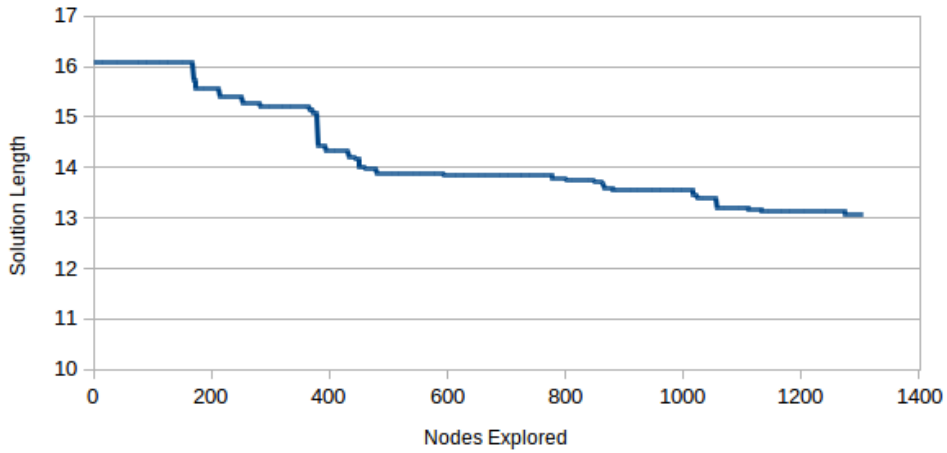
b) WA* euclidean distance, $\epsilon = 10$

Examples generated: `https://qiao.github.io/PathFinding.js/visual/`

2. Introduction

Anytime WA* (AWA*) [Hansen and Zhou, 2007]:

- Uses weighted function $f'(s) = g(s) + \epsilon \cdot h(s)$ to determine the order in which states $s \in \mathcal{O}$ are expanded.
- Continues the search after a (suboptimal) solution is found, in order to improve it.



- Uses admissible function $f(s) = g(s) + h(s)$ together with the cost of the best solution $g(s^{\text{goal}})$ found so far to
  - prune the search space: $g(s^{\text{goal}}) < f(s)$,
  - detect convergence: $\text{error} = g(s^{\text{goal}}) - \min_{s \in \mathcal{O}} f(s)$.

3. State of the Art

Solving the Rubik's cube with deep reinforcement learning and search [Agostinelli et al., 2019]:

- Proposed DeepCubeA, which
  - utilizes approximate value iteration to train a deep neural network as heuristic function.
  - trains on states obtained by starting from the goal state and randomly taking moves in reverse.
  - uses the learned heuristic function in WA* to solve combinatorial puzzles.
- Demonstrated that DeepCubeA
  - solves 100% of all test configurations.
  - finds in 60.3% of the time a shortest path to the goal state.

3. State of the Art

A Differentiable Loss Function for Learning Heuristics in A* [Chrestien et al., 2022]:

- Introduced a novel loss function L^* for learning heuristic functions.
- Heuristic function h_θ is realized by a neural network (NN).
- Proposed L^* loss requires states
  - on the optimal path $\mathcal{S}^o$,
  - off the optimal path $\mathcal{S}^n$,

  and attempts to shrink the size of the the A* search tree:

$$\mathrm{L}^* = \frac{1}{|\mathcal{S}^o||\mathcal{S}^n|} \sum_{s' \in \mathcal{S}^o} \sum_{s'' \in \mathcal{S}^n} [\![g(s') + h_\theta(s') \geq g(s'') + h_\theta(s'')]\!] +$$

$$\frac{1}{|\mathcal{S}^o|\,(|\mathcal{S}^o| - 1)} \sum_{i=2}^{|\mathcal{S}^o|} \sum_{j=1}^{i} [\![g(s_i) + h_\theta(s_i) > g(s_j) + h_\theta(s_j)]\!]$$

3. State of the Art

Training on exact solved instances:

- Training set = set of states generated during A* search with admissible heuristic function on random instances.
- Parameters of the NN are optimized the training set, where
  - each mini-batch contains all states from one problem instance.

Training in reinforcement learning (RL) manner:

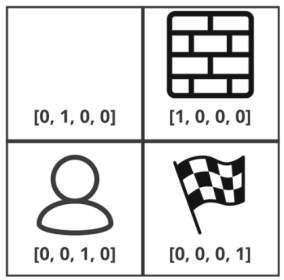
1. In each iteration, h_θ is first used in A* to try to solve mazes from an available set of mazes (time limit=10 min).
2. Parameters of the NN are optimized on a set of mazes it has solved.

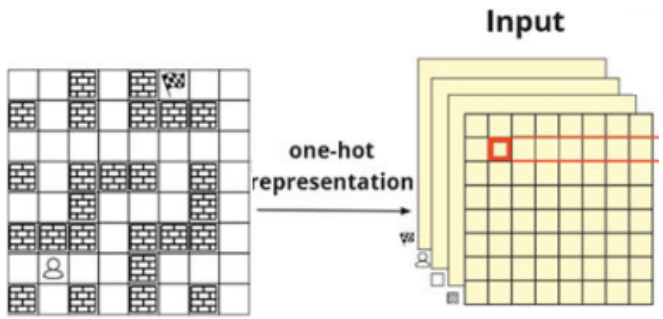
Note: Since h_θ is not guaranteed to be admissible, solutions returned by A* can be suboptimal.

3. State of the Art

Convolutional neural network (CNN):

- Input: one-hot bitvector encoding.





- Output: heuristic value h_θ.

Authors used two different CNNs in their evaluations:

1. CNN: composed of several convolutional layers.
2. CoAt: CNN with self-attention.

3. State of the Art

Results:

1. Models optimized on exact solved Sokoban mazes of size 10×10 with $n = 3$ boxes.

				CNN		CoAt	
n	SBA*	Merc	FD Stone Soup	L_2	L^*	L_2	L^*
3	21.40	29.32	27.89	30.56	28.67	22.90	22.02
4	34.00	41.00	37.00	43.42	41.33	35.11	35.03
5	38.82	45.76	42.37	45.34	44.83	40.12	40.12
6	41.11	-	-	49.82	46.32	42.11	41.65
7	-	-	-	58.23	56.33	53.33	53.19

Average plan length of Sokoban mazes in the test set.

2. Models trained in RL manner on mazes of size 40×40.

	1000	
epoch	L_2	L^*
0	25	25
1	42	45
2	45	68
3	69	83
4	84	90

Percentage of solved mazes of size 40×40.

4. Open Questions

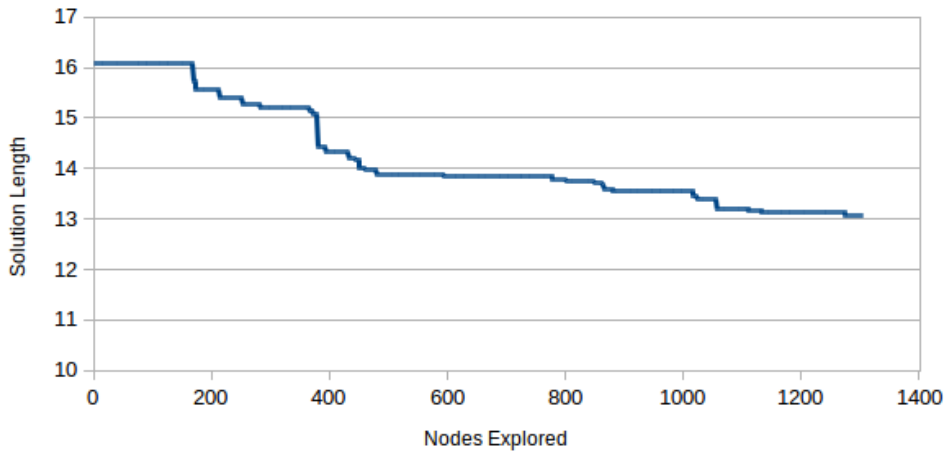
1. How to learn a guidance function for AWA* that is able to generalize well to different problem sizes?

2. Is it possible to learn admissible and consistent guidance functions for AWA*?

3. How to design a powerful NN for planning problems?

5. Idea

- Develop new loss function that attempts to shrink the size of the AWA* search tree by minimizing the area under the curve.



Thank you for your attention!

Questions?

References I


Agostinelli, F., McAleer, S., Shmakov, A., and Baldi, P. (2019).
Solving the rubik's cube with deep reinforcement learning and search.
Nature Machine Intelligence, 1(8):356–363.


Chrestien, L., Pevny, T., Komenda, A., and Edelkamp, S. (2022).
A differentiable loss function for learning heuristics in a*.


Hansen, E. A. and Zhou, R. (2007).
Anytime heuristic search.
Journal of Artificial Intelligence Research, 28:267–297.

References II

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968).
A formal basis for the heuristic determination of minimum cost paths.
IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107.

Pohl, I. (1969).
First Results on the Effect of Error in Heuristic Search.
MIP-R-. Edinburgh University, Department of Machine Intelligence and Perception.

Deep approximate value iteration (DAVI)

- Value iteration is a dynamic programming algorithm that iteratively improves a cost-to-go function J.
- Value iteration loops through each state s and updates $J(s)$ until convergence:

$$J \leftarrow \min_a \sum_{s' \in \mathcal{S}} P^a(s, s')(g^a(s, A(s, a)) + \gamma J(A(s, a))$$

where

  - $P^a(s, s') =$ probability of transitioning from state s to state s' by taking action a.
  - $A(s, a) =$ state obtained from taking action a in state s,
  - $g^a(s, s') =$ cost to transition from state s to state s' taking action a.

and

- DAVI: DNN$=J$ is trained to minimize the mean squared error between $J(s)$, and the updated cost-to-go estimation $J'(s)$:

$$J'(s) = \min_a (g^a(s, A(s, a)) + J(A(s, a)))$$

Deep approximate value iteration (DAVI)

Algorithm 1: DAVI.

Input:

B: Batch size

K: Maximum number of scrambles

M: Training iterations

C: How often to check for convergence

ϵ: Error threshold

Output:

θ: Trained neural network parameters

$\theta \leftarrow initialize\_parameters()$

$\theta_e \leftarrow \theta$

for $m = 1$ to M **do**

  $X \leftarrow get\_scrambled\_states(B, K)$

  for $x_i \in X$ **do**

    $y_i \leftarrow \min_a(g^a(x_i, A(x_i, a)) + j_{\theta_e}(A(x_i, a)))$

  $\theta, \text{loss} \leftarrow \text{train}(j_\theta, X, \mathbf{y})$

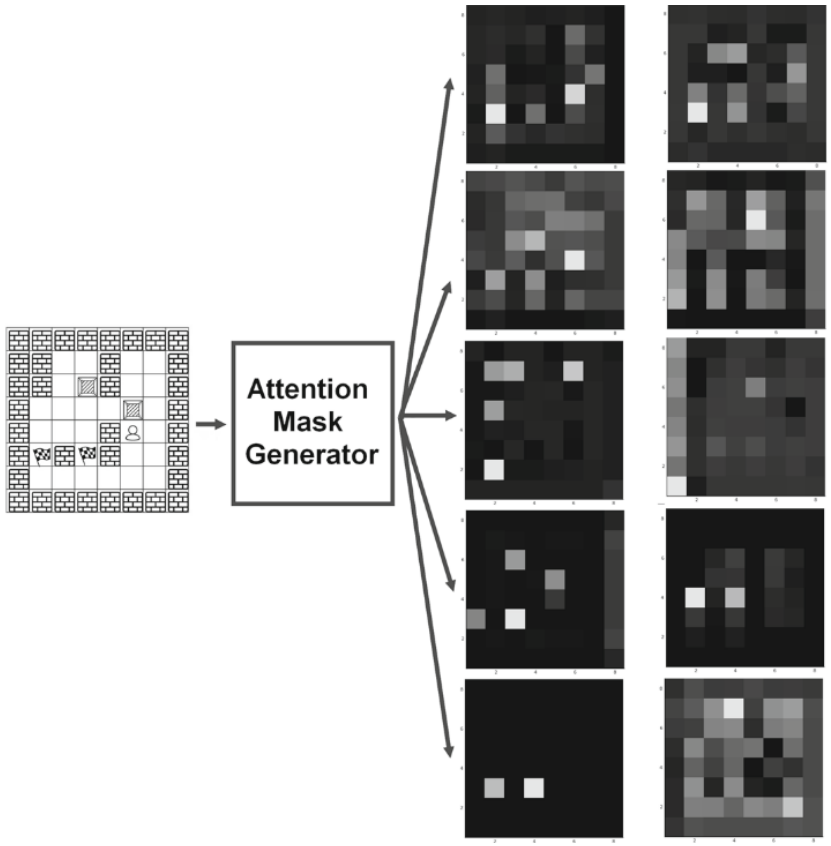
  if $(M \bmod C = 0)$ *and* $(\text{loss} < \epsilon)$ **then**

    $\theta_e \leftarrow \theta$

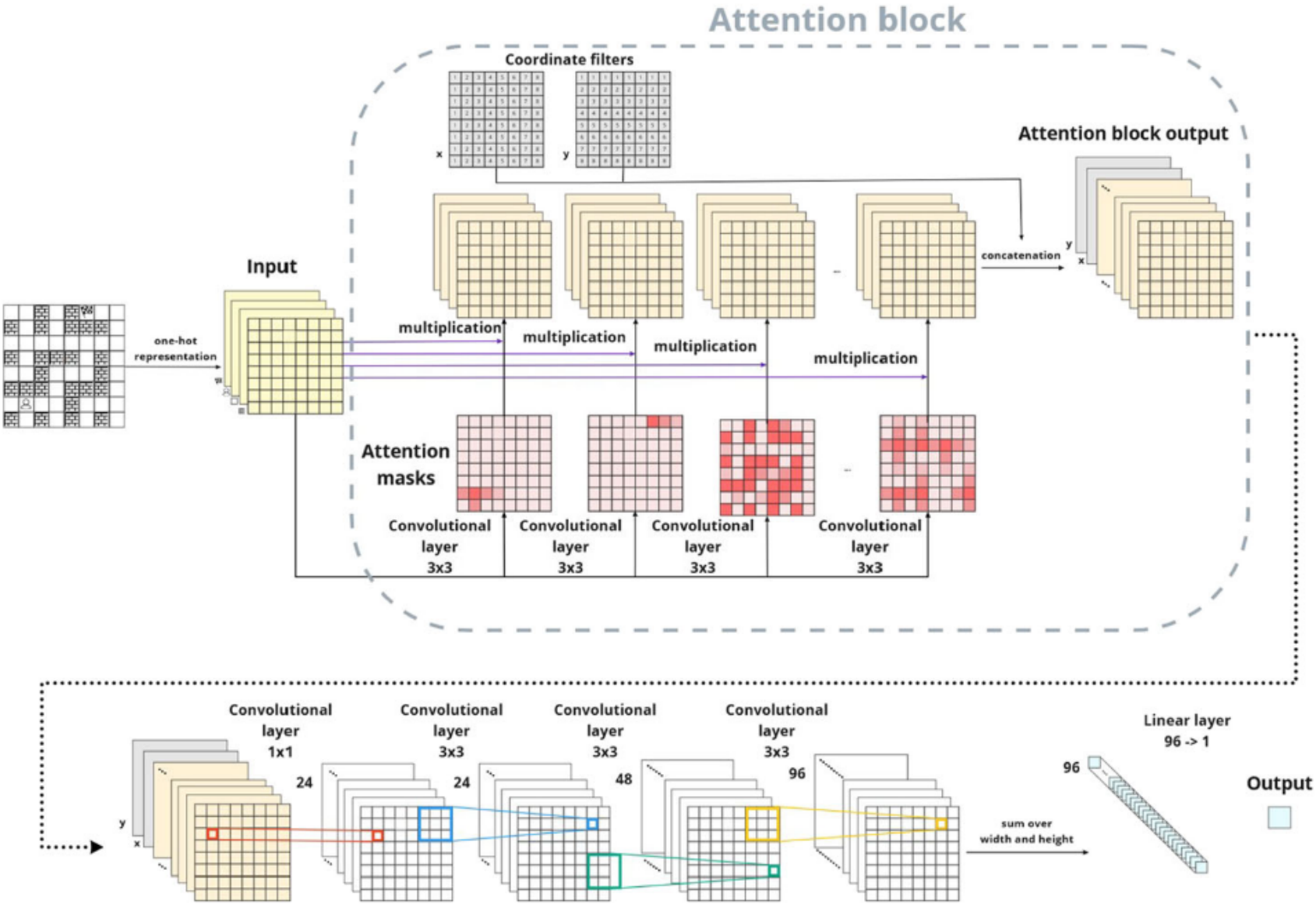
Return θ

Attention for neural networks

- Attention allows the network to focus only on subsets of inputs and requires the creation of attention masks.
- Masks are generated using convolutional layer and a softmax layer of the same width and height as the input, with all values summing up to one.



Attention for neural networks



Attention CNN architecture.