BSc Thesis Colloquium: Domain-Dependent Policy Learning using Neural Networks in Classical Planning

Lukas Schäfer

Foundations of Artificial Intelligence Research Group, Universität des Saarlandes

Reviewers: *Prof. Dr. Jörg Hoffmann, Dr. Mario Fritz*
Supervisors: *Prof. Dr. Jörg Hoffmann, Patrick Ferber, Maximilian Fickert*

August 20, 2018

Planning

Classical Automated Planning

- "big goal": Solve arbitrary tasks with a single planner
- find a sequence of actions that leads to a goal
- classical (automated) planning - limited to finite, deterministic, fully-observable tasks
- dominant approach: state-based heuristic search

STRIPS

Formalisation as STRIPS task $\Pi = (\mathcal{P}, \mathcal{A}, c, I, G)$:

STRIPS

Formalisation as STRIPS task $\Pi = (\mathcal{P}, \mathcal{A}, c, I, G)$:

- **propositions** $\mathcal{P}$

STRIPS

Formalisation as STRIPS task $\Pi = (\mathcal{P}, \mathcal{A}, c, I, G)$:

- **propositions** $\mathcal{P}$
- **actions** $\mathcal{A}$ as triples (pre_a, add_a, del_a) with

STRIPS

Formalisation as STRIPS task $\Pi = (\mathcal{P}, \mathcal{A}, c, I, G)$:

- **propositions** $\mathcal{P}$
- **actions** $\mathcal{A}$ as triples (pre_a, add_a, del_a) with
  - pre_a: preconditions of a
  - add_a: add-list of a
  - del_a: delete-list of a

STRIPS

Formalisation as STRIPS task $\Pi = (\mathcal{P}, \mathcal{A}, c, I, G)$:

- **propositions** $\mathcal{P}$
- **actions** $\mathcal{A}$ as triples (pre_a, add_a, del_a) with
  - pre_a: preconditions of a
  - add_a: add-list of a
  - del_a: delete-list of a
- **cost function** c
- **initial state** I
- **goal(s)** G

Learning in Automated Planning

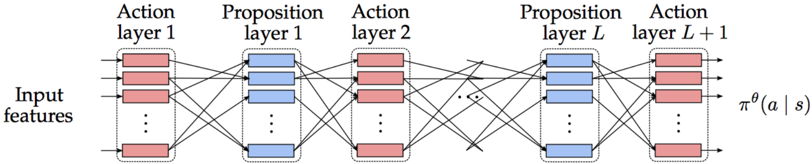
- planning is still dominated by domain-independent heuristics
- learning domain-specific knowledge can improve performance
- learning **generalized policies**
- computed by AlphaGo (Zero) with neural networks and combined with Monte-Carlo tree search

ASNets

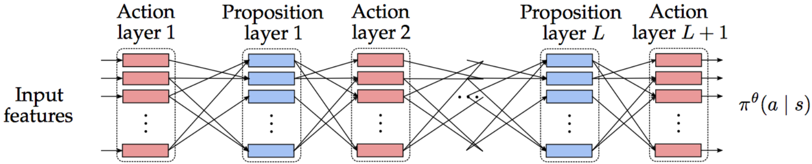
Idea

- neural network architecture suited for planning
- proposed by Toyer et al. in 2017
- learns domain-specific generalized policy
- can be learned from small problems and exploited on arbitrary ones of this domain

Architecture



Architecture



- alternating **proposition** and **action layers**
- modules correspond to ground actions/ propositions
- **policy** π^θ represents probability to choose action a in state s
- sparse connectivity based on **relatedness** $R(a, p)$ iff p appears in pre_a, add_a or del_a

Action Modules

compute **hidden representation** $\phi_a^l = f(W_a^l \cdot u_a^l + b_a^l) \in \mathbb{R}^{d_h}$

Action Modules

compute **hidden representation** $\phi_a^l = f(W_a^l \cdot u_a^l + b_a^l) \in \mathbb{R}^{d_h}$

- nonlinear activation function f
- learned weight matrix $W_a^l \in \mathbb{R}^{d_h \times d_a^l}$ and bias $b_a^l \in \mathbb{R}^{d_h}$
- input vector $u_a^l \in \mathbb{R}^{d_a^l}$: concatenation of hidden representations of related proposition modules

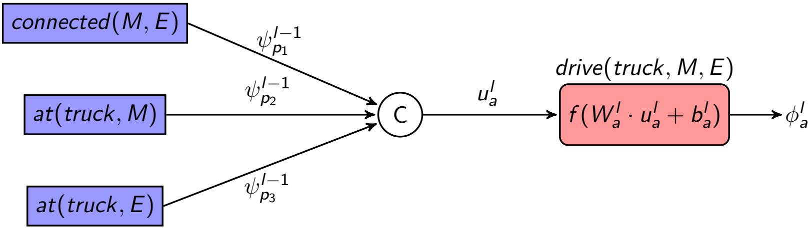
Action Modules

compute **hidden representation** $\phi_a^l = f(W_a^l \cdot u_a^l + b_a^l) \in \mathbb{R}^{d_h}$

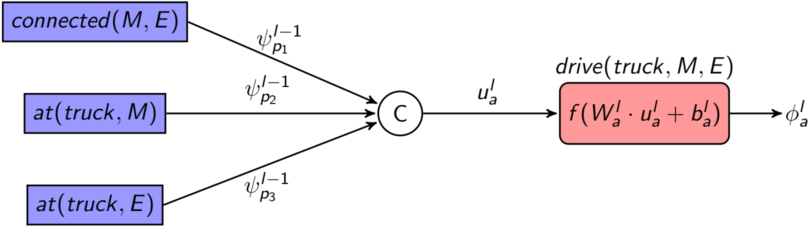
- nonlinear activation function f
- learned weight matrix $W_a^l \in \mathbb{R}^{d_h \times d_a^l}$ and bias $b_a^l \in \mathbb{R}^{d_h}$
- input vector $u_a^l \in \mathbb{R}^{d_a^l}$: concatenation of hidden representations of related proposition modules

$$R(a, p_i) \text{ for } i = 1, ..., M : u_a^l = \begin{bmatrix} \psi_1^{l-1} \\ \vdots \\ \psi_M^{l-1} \end{bmatrix}$$

Action Modules

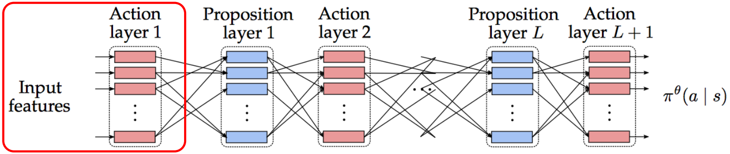


Action Modules

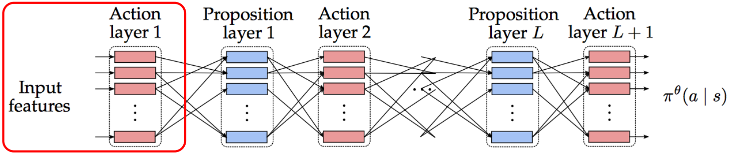


same structure for all actions of the '*drive*' schema
$\rightarrow$ W_a^l and b_a^l can be **shared** for all actions in layer l of the drive schema

Input Layer


Input
features
Action
layer 1
Proposition
layer 1
Action
layer 2
Proposition
layer L
Action
layer $L+1$
$\pi^\theta(a \mid s)$

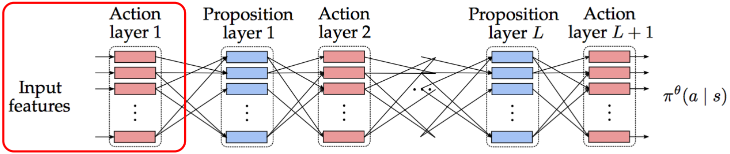
Input Layer



first input vectors u_a^1 for state $s \in S$ include

- binary values indicating iff $p_i \in s$
- binary values indicating iff $p_i \in G$
- value indicating iff $pre_a \subseteq s$

Input Layer

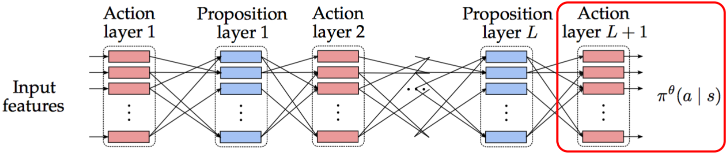


first input vectors u_a^1 for state $s \in S$ include

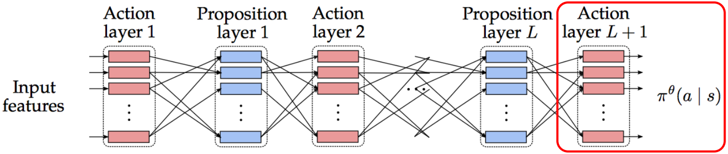
- binary values indicating iff $p_i \in s$
- binary values indicating iff $p_i \in G$
- value indicating iff $pre_a \subseteq s$
- additional heuristic features

Toyer et al. experimented with disjunctive action landmark features

Output Layer



Output Layer



- has to compute policy π^θ as probability distribution
- **masked softmax** activation function

Proposition Modules

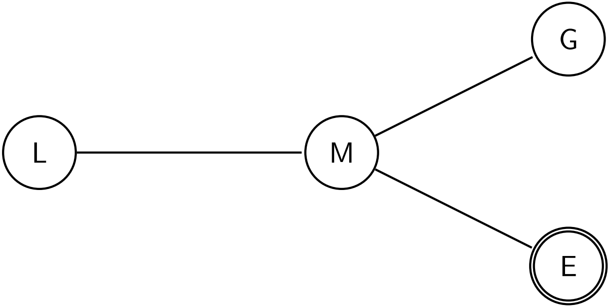
- compute **hidden representation** $\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \in \mathbb{R}^{d_h}$

Proposition Modules

- compute **hidden representation** $\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \in \mathbb{R}^{d_h}$
- **difficulty**: number of related actions can vary

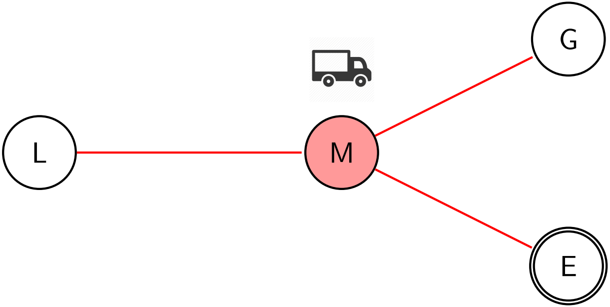
Proposition Modules

- compute **hidden representation** $\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \in \mathbb{R}^{d_h}$
- **difficulty**: number of related actions can vary
- example:



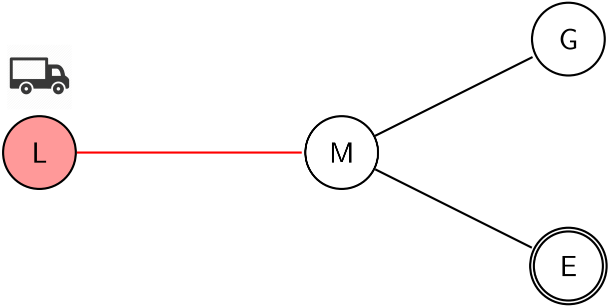
Proposition Modules

- compute **hidden representation** $\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \in \mathbb{R}^{d_h}$
- **difficulty**: number of related actions can vary
- example:



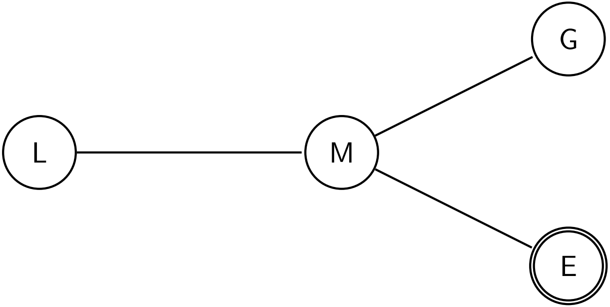
Proposition Modules

- compute **hidden representation** $\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \in \mathbb{R}^{d_h}$
- **difficulty**: number of related actions can vary
- example:



Proposition Modules

- compute **hidden representation** $\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) \in \mathbb{R}^{d_h}$
- **difficulty**: number of related actions can vary
- example:



- solution: **Pooling**

Empirical Experiment

- experiment for **probabilistic** and **classical planning**
  - comparison with multiple state-of-the-art baseline planners
  - ASNets with h^{LM-cut} and h^{add} teacher policy

Empirical Experiment

- experiment for **probabilistic** and **classical planning**
  - comparison with multiple state-of-the-art baseline planners
  - ASNets with h^{LM-cut} and h^{add} teacher policy
- **results**
  - prob. planning: ASNets mostly outperformed baseline planners
  - class. planning: ASNets were outperformed by LAMA planners

Thesis Objective

Thesis Objective

- focus on application in deterministic, classical planning

Thesis Objective

- focus on application in deterministic, classical planning
- implementation of ASNets in Fast-Downward planning system

Thesis Objective

- focus on application in deterministic, classical planning
- implementation of ASNets in Fast-Downward planning system
  - network definition
  - extension and integration in Fast-Downward
  - training algorithm

Thesis Objective

- focus on application in deterministic, classical planning
- implementation of ASNets in Fast-Downward planning system
  - network definition
  - extension and integration in Fast-Downward
  - training algorithm
- extensive experiment to evaluate suitability of ASNets for classical automated planning

Implementation

Network Definition

- network architecture based on planning task

Network Definition

- network architecture based on planning task
- Fast-Downward translation builds PDDL representation

Network Definition

- network architecture based on planning task
- Fast-Downward translation builds PDDL representation
- computation of relations between groundings

Network Definition

- network architecture based on planning task
- Fast-Downward translation builds PDDL representation
- computation of relations between groundings
    1. compute relations between action schemas and predicates

Network Definition

- network architecture based on planning task
- Fast-Downward translation builds PDDL representation
- computation of relations between groundings
  1. compute relations between action schemas and predicates
  2. instantiate abstracts to groundings

ASNet Network

- ASNets architecture implemented with *Keras* on *Tensorflow* backend

ASNet Network

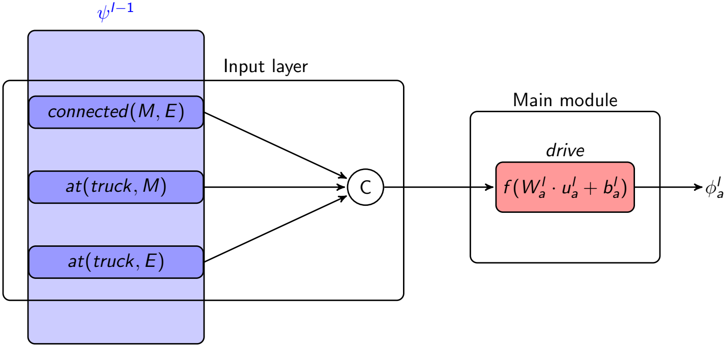
- ASNets architecture implemented with *Keras* on *Tensorflow* backend
- modules implemented as custom *Keras* layers

ASNet Network

- ASNets architecture implemented with *Keras* on *Tensorflow* backend
- modules implemented as custom *Keras* layers
- distinguished between input extraction and shared main module

ASNet Network

- ASNets architecture implemented with *Keras* on *Tensorflow* backend
- modules implemented as custom *Keras* layers
- distinguished between input extraction and shared main module



Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, \ldots, T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
 $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
 $n_{epoch} \leftarrow 0$ ▷ Epoch counter
 while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
  for all $p \in P_{train}$ **do**
   $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
   for $n_{p-epoch} = 1, \ldots, T_{prob-epochs}$ **do**   ▷ Run problem epochs
    $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
    $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
   weights $\leftarrow asnet_p.save\_weights()$
   $\mathcal{M} \leftarrow \emptyset$
  $n_{epoch} \leftarrow n_{epoch} + 1$

- $T_{max-epochs}$: maximum number of training cycle epochs

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure TRAIN(P_{train})
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow$ BUILD-MODEL(p, weights) ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow$ SAMPLE(p) ▷ Sample on p
                OPT_TRAIN($asnet_p, \mathcal{M}, T_{train-epochs}$) ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
      for $n_{p-epoch} = 1, \ldots, T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
        $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

- terminate the training when network is performing very well

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
      for $n_{p-epoch} = 1, \ldots, T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
        $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

- terminate the training when network is performing very well
- less restrictive than Sam Toyer's early stopping

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
      for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow \text{SAMPLE}(p)$ ▷ Sample on p
        $\text{OPT\_TRAIN}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

- iterate over all training problems in P_{train}

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure TRAIN(P_{train})
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow$ BUILD-MODEL(p, weights) ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow$ SAMPLE(p) ▷ Sample on p
                OPT_TRAIN($asnet_p, \mathcal{M}, T_{train-epochs}$) ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

function BUILD-MODEL(p, weights) ▷ Build ASNet model and load weights
    task_meta $\leftarrow$ COMPUTE_META(p) ▷ Compute task meta information
    $asnet_p \leftarrow$ CREATE_MODEL(task_meta, p)
    if weights exist **then**
        $asnet_p.load\_weights$(weights) ▷ Load weights if existing
    return $asnet_p$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
$\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
$n_{epoch} \leftarrow 0$ ▷ Epoch counter
while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
for all $p \in P_{train}$ **do**
$asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
for $n_{p-epoch} = 1, \ldots, T_{prob-epochs}$ **do** ▷ Run problem epochs
$\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
$\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
weights $\leftarrow asnet_p.save\_weights()$
$\mathcal{M} \leftarrow \emptyset$
$n_{epoch} \leftarrow n_{epoch} + 1$

function $\textsc{Build-Model}(p, \text{weights})$ ▷ Build ASNet model and load weights
task_meta $\leftarrow \textsc{compute\_meta}(p)$ ▷ Compute task meta information
$asnet_p \leftarrow \textsc{create\_model}(\text{task\_meta}, p)$
if weights exist **then**
$asnet_p.load\_weights(\text{weights})$ ▷ Load weights if existing
return $asnet_p$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

function $\textsc{Build-Model}(p, \text{weights})$ ▷ Build ASNet model and load weights
    task_meta $\leftarrow \textsc{compute\_meta}(p)$ ▷ Compute task meta information
    $asnet_p \leftarrow \textsc{create\_model}(\text{task\_meta}, p)$
    if weights exist **then**
        $asnet_p.load\_weights(\text{weights})$ ▷ Load weights if existing
    return $asnet_p$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure TRAIN(P_{train})
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow$ BUILD-MODEL(p, weights) ▷ ASNet model
      for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow$ SAMPLE(p) ▷ Sample on p
        OPT_TRAIN($asnet_p, \mathcal{M}, T_{train-epochs}$) ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

function BUILD-MODEL(p, weights) ▷ Build ASNet model and load weights
  task_meta $\leftarrow$ COMPUTE_META(p) ▷ Compute task meta information
  $asnet_p \leftarrow$ CREATE_MODEL(task_meta, p)
  if weights exist **then**
    $asnet_p.load\_weights$(weights) ▷ Load weights if existing
  return $asnet_p$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure TRAIN(P_{train})
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow$ BUILD-MODEL(p, weights) ▷ ASNet model
      for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow$ SAMPLE(p) ▷ Sample on p
        OPT_TRAIN($asnet_p, \mathcal{M}, T_{train-epochs}$) ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

function BUILD-MODEL(p, weights) ▷ Build ASNet model and load weights
  task_meta $\leftarrow$ COMPUTE_META(p) ▷ Compute task meta information
  $asnet_p \leftarrow$ CREATE_MODEL(task_meta, p)
  if weights exist **then**
    $asnet_p.load\_weights$(weights) ▷ Load weights if existing
  return $asnet_p$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, \ldots, T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            $\text{weights} \leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

- execute $T_{prob-epochs}$ problem epochs

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
      for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
        $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

- sample states using network search $\mathcal{S}^{\theta}$ and teacher search $\mathcal{S}^{*}$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

- sample states using network search $\mathcal{S}^{\theta}$ and teacher search $\mathcal{S}^{*}$
- more about the sampling process after this cycle

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

- update θ using $T_{train-epochs}$ gradient descent steps

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, \dots, T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

- update θ using $T_{train-epochs}$ gradient descent steps
- minimize crossentropy loss:

$$\mathcal{L}_\theta(\mathcal{M}) = \sum_{s \in \mathcal{M}} \sum_{a \in \mathcal{A}} -(1 - y_{s,a}) \cdot log(1 - \pi^\theta(a \mid s)) - y_{s,a} \cdot log\ \pi^\theta(a \mid s)$$

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure TRAIN(P_{train})
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow$ BUILD-MODEL(p, weights) ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow$ SAMPLE(p) ▷ Sample on p
                OPT_TRAIN($asnet_p, \mathcal{M}, T_{train-epochs}$) ▷ Train network
            weights $\leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

- save network weights after training for problem p

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
    $n_{epoch} \leftarrow 0$ ▷ Epoch counter
    while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
        for all $p \in P_{train}$ **do**
            $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
            for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
                $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
                $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
            $\text{weights} \leftarrow asnet_p.save\_weights()$
            $\mathcal{M} \leftarrow \emptyset$
        $n_{epoch} \leftarrow n_{epoch} + 1$

- save network weights after training for problem p
- reset sampled states

Training Cycle

Algorithm Training Cycle on set of training problems P_{train}

procedure $\textsc{Train}(P_{train})$
  $\mathcal{M} \leftarrow \emptyset$ ▷ Sampled states
  $n_{epoch} \leftarrow 0$ ▷ Epoch counter
  while $n_{epoch} < T_{max-epochs}$ and not early stopping **do**
    for all $p \in P_{train}$ **do**
      $asnet_p \leftarrow \textsc{Build-Model}(p, \text{weights})$ ▷ ASNet model
      for $n_{p-epoch} = 1, ..., T_{prob-epochs}$ **do** ▷ Run problem epochs
        $\mathcal{M} \leftarrow \textsc{sample}(p)$ ▷ Sample on p
        $\textsc{opt\_train}(asnet_p, \mathcal{M}, T_{train-epochs})$ ▷ Train network
      weights $\leftarrow asnet_p.save\_weights()$
      $\mathcal{M} \leftarrow \emptyset$
    $n_{epoch} \leftarrow n_{epoch} + 1$

Sampling

Sampling for problem p:

Sampling

Sampling for problem p:

- extract $s_0^\theta, \ldots, s_N^\theta$ with network exploration $\mathcal{S}^\theta$

Sampling

Sampling for problem p:

- extract $s_0^\theta, \ldots, s_N^\theta$ with network exploration $\mathcal{S}^\theta$
  $\rightarrow$ improve upon previous performance

Sampling

Sampling for problem p:

- extract $s_0^\theta, \ldots, s_N^\theta$ with network exploration $\mathcal{S}^\theta$
  $\rightarrow$ improve upon previous performance
- apply $\mathcal{S}^*$ from explored states to extract $s_0^*, \ldots, s_N^*$

Sampling

Sampling for problem p:

- extract $s_0^\theta, \dots, s_N^\theta$ with network exploration $\mathcal{S}^\theta$
  $\rightarrow$ improve upon previous performance
- apply $\mathcal{S}^*$ from explored states to extract $s_0^*, \dots, s_N^*$
  $\rightarrow$ learn "good" policy states

Policies in Fast-Downward

Policies:

- addition of policies in Fast-Downward

Policies in Fast-Downward

Policies:

- addition of policies in Fast-Downward
- based on preferred operators $\rightarrow$ added preferences

Policies in Fast-Downward

Policies:

- addition of policies in Fast-Downward
- based on preferred operators $\rightarrow$ added preferences
- network policy communicates with a network representation in Fast-Downward based on a *Protobuf* network model

Policies in Fast-Downward

Policies:

- addition of policies in Fast-Downward
- based on preferred operators $\rightarrow$ added preferences
- network policy communicates with a network representation in Fast-Downward based on a *Protobuf* network model

Policy search:

- new search-engine for policies in Fast-Downward

Policies in Fast-Downward

Policies:

- addition of policies in Fast-Downward
- based on preferred operators $\rightarrow$ added preferences
- network policy communicates with a network representation in Fast-Downward based on a *Protobuf* network model

Policy search:

- new search-engine for policies in Fast-Downward
- naive search following the most probable action

Policies in Fast-Downward

Policies:

- addition of policies in Fast-Downward
- based on preferred operators $\rightarrow$ added preferences
- network policy communicates with a network representation in Fast-Downward based on a *Protobuf* network model

Policy search:

- new search-engine for policies in Fast-Downward
- naive search following the most probable action
- allows solely evaluating the network policy quality

Evaluation

Evaluation Objective

evaluate the suitability of ASNets for classical automated planning

Evaluation Objective

evaluate the suitability of ASNets for classical automated planning

1. can ASNets learn good (or even optimal) policies

Evaluation Objective

evaluate the suitability of ASNets for classical automated planning

1. can ASNets learn good (or even optimal) policies
2. on which domains do ASNets perform well

Evaluation Objective

evaluate the suitability of ASNets for classical automated planning

1. can ASNets learn good (or even optimal) policies
2. on which domains do ASNets perform well
3. how long do we need to train ASNets

ASNet Configurations

- two layers and hidden representation size $d_h = 16$
- ELU activation function
- L_2 regularization and dropout
- $T_{max-epochs} = 10$, $T_{prob-epochs} = 3$ and $T_{train-epochs} = 100$
- Adam optimizer with $\alpha = 0.001$

ASNet Configurations

- two layers and hidden representation size $d_h = 16$
- ELU activation function
- L_2 regularization and dropout
- $T_{max-epochs} = 10$, $T_{prob-epochs} = 3$ and $T_{train-epochs} = 100$
- Adam optimizer with $\alpha = 0.001$
- three teacher searches
  1. A^* with h^{LM-cut} (optimal)
  2. A^* with h^{add}
  3. $GBFS$ with h^{FF} using preferred operators in dual-queue

ASNet Configurations

- two layers and hidden representation size $d_h = 16$
- ELU activation function
- L_2 regularization and dropout
- $T_{max-epochs} = 10$, $T_{prob-epochs} = 3$ and $T_{train-epochs} = 100$
- Adam optimizer with $\alpha = 0.001$
- three teacher searches
  1. A^* with h^{LM-cut} (optimal)
  2. A^* with h^{add}
  3. $GBFS$ with h^{FF} using preferred operators in dual-queue
- time limit for training of two hours

Baseline Planners

1. A^* with h^{LM-cut}
2. A^* with h^{add}
3. $GBFS$ with h^{FF} using preferred operators in dual-queue

Baseline Planners

1. A^* with h^{LM-cut}
2. A^* with h^{add}
3. *GBFS* with h^{FF} using preferred operators in dual-queue
4. LAMA-2011
   - winner of 2011 IPC
   - iterative *GBFS* and $(W)A^*$
   - h^{LM}s and h^{FF}

Baseline Planners

1. A^* with h^{LM-cut}
2. A^* with h^{add}
3. $GBFS$ with h^{FF} using preferred operators in dual-queue
4. LAMA-2011
   - winner of 2011 IPC
   - iterative $GBFS$ and $(W)A^*$
   - h^{LM}s and h^{FF}

Evaluation time limit of 30 minutes

Evaluation Domains

- eight domains from previous IPC iterations and the FF-Domain collection

Domain	number of evaluation problems	number of training problems	expected difficulty
Tyreworld	20	2	simple
TurnAndOpen	19	3	simple
Sokoban	30	2	simple - mediocre
Hanoi	20	3	mediocre
Floortile	20	1	mediocre - hard
Blocksworld	35	3	hard
Elevator	30	1	hard
ParcPrinter	10	4	hard

Evaluation Domains

- eight domains from previous IPC iterations and the FF-Domain collection

Domain	number of evaluation problems	number of training problems	expected difficulty
Tyreworld	20	2	simple
TurnAndOpen	19	3	simple
Sokoban	30	2	simple - mediocre
Hanoi	20	3	mediocre
Floortile	20	1	mediocre - hard
Blocksworld	35	3	hard
Elevator	30	1	hard
ParcPrinter	10	4	hard

Evaluation Domains

- eight domains from previous IPC iterations and the FF-Domain collection

Domain	number of evaluation problems	number of training problems	expected difficulty
Tyreworld	20	2	simple
TurnAndOpen	19	3	simple
Sokoban	30	2	simple - mediocre
Hanoi	20	3	mediocre
Floortile	20	1	mediocre - hard
Blocksworld	35	3	hard
Elevator	30	1	hard
ParcPrinter	10	4	hard

Evaluation Domains

- eight domains from previous IPC iterations and the FF-Domain collection

Domain	number of evaluation problems	number of training problems	expected difficulty
Tyreworld	20	2	simple
TurnAndOpen	19	3	simple
Sokoban	30	2	simple - mediocre
Hanoi	20	3	mediocre
Floortile	20	1	mediocre - hard
Blocksworld	35	3	hard
Elevator	30	1	hard
ParcPrinter	10	4	hard

Evaluation Domains

- eight domains from previous IPC iterations and the FF-Domain collection

Domain	number of evaluation problems	number of training problems	expected difficulty
Tyreworld	20	2	simple
TurnAndOpen	19	3	simple
Sokoban	30	2	simple - mediocre
Hanoi	20	3	mediocre
Floortile	20	1	mediocre - hard
Blocksworld	35	3	hard
Elevator	30	1	hard
ParcPrinter	10	4	hard

Evaluation Domains

- eight domains from previous IPC iterations and the FF-Domain collection

Domain	number of evaluation problems	number of training problems	expected difficulty
Tyreworld	20	2	simple
TurnAndOpen	19	3	simple
Sokoban	30	2	simple - mediocre
Hanoi	20	3	mediocre
Floortile	20	1	mediocre - hard
Blocksworld	35	3	hard
Elevator	30	1	hard
ParcPrinter	10	4	hard

Evaluation Results

Domain	A^* LM	A^* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Tyreworld	3/20	6/20	20/20	20/20	20/20	0/20	20/20
Turnandopen	0/19	17/19	15/19	19/19	0/19	0/19	0/19
Sokoban	28/30	29/30	29/30	29/30	0/30	0/30	0/30
Hanoi	13/20	15/20	16/20	15/20	3/20	2/20	2/20
Floortile	6/20	20/20	9/20	9/20	0/20	1/20	0/20
Blocksworld	28/35	35/35	35/35	35/35	7/35	7/35	4/35
Elevator	2/30	15/30	30/30	30/30	0/29	0/30	0/30
Parcprinter	6/10	8/10	10/10	10/10	1/10	1/10	4/10

Evaluation Results

Domain	A* LM	A* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Tyreworld	3/20	6/20	20/20	20/20	20/20	0/20	20/20
Turnandopen	0/19	17/19	15/19	19/19	0/19	0/19	0/19
Sokoban	28/30	29/30	29/30	29/30	0/30	0/30	0/30
Hanoi	13/20	15/20	16/20	15/20	3/20	2/20	2/20
Floortile	6/20	20/20	9/20	9/20	0/20	1/20	0/20
Blocksworld	28/35	35/35	35/35	35/35	7/35	7/35	4/35
Elevator	2/30	15/30	30/30	30/30	0/29	0/30	0/30
Parcprinter	6/10	8/10	10/10	10/10	1/10	1/10	4/10

That looks quite disappointing

Evaluation Results

Domain	A^* LM	A^* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Tyreworld	3/20	6/20	20/20	20/20	20/20	0/20	20/20
Turnandopen	0/19	17/19	15/19	19/19	0/19	0/19	0/19
Sokoban	28/30	29/30	29/30	29/30	0/30	0/30	0/30
Hanoi	13/20	15/20	16/20	15/20	3/20	2/20	2/20
Floortile	6/20	20/20	9/20	9/20	0/20	1/20	0/20
Blocksworld	28/35	35/35	35/35	35/35	7/35	7/35	4/35
Elevator	2/30	15/30	30/30	30/30	0/29	0/30	0/30
Parcprinter	6/10	8/10	10/10	10/10	1/10	1/10	4/10

Some learning is visible

Evaluation Results

Domain	A^* LM	A^* add	GBFS	LAMA	ASNet LM	ASNet add	ASNet FF
Tyreworld	3/20	6/20	20/20	20/20	20/20	0/20	20/20
Turnandopen	0/19	17/19	15/19	19/19	0/19	0/19	0/19
Sokoban	28/30	29/30	29/30	29/30	0/30	0/30	0/30
Hanoi	13/20	15/20	16/20	15/20	3/20	2/20	2/20
Floortile	6/20	20/20	9/20	9/20	0/20	1/20	0/20
Blocksworld	28/35	35/35	35/35	35/35	7/35	7/35	4/35
Elevator	2/30	15/30	30/30	30/30	0/29	0/30	0/30
Parcprinter	6/10	8/10	10/10	10/10	1/10	1/10	4/10

Overall still not the desired performance

Detailed Evaluation

- **Floortile & TurnAndOpen:**

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search $\rightarrow$ sampling did not terminate

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search $\rightarrow$ sampling did not terminate
  - Floortile: dead-ends were successfully prevented

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search → sampling did not terminate
  - Floortile: dead-ends were successfully prevented
- **Floortile, Sokoban & TurnAndOpen:**

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search → sampling did not terminate
  - Floortile: dead-ends were successfully prevented
- **Floortile, Sokoban & TurnAndOpen:**
  - indecisive movement → inverting actions are applied

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search → sampling did not terminate
  - Floortile: dead-ends were successfully prevented
- **Floortile, Sokoban & TurnAndOpen:**
  - indecisive movement → inverting actions are applied
    - limited receptive field:
    - interchangeable paths:

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search $\rightarrow$ sampling did not terminate
  - Floortile: dead-ends were successfully prevented
- **Floortile, Sokoban & TurnAndOpen:**
  - indecisive movement $\rightarrow$ inverting actions are applied
    - limited receptive field: additional input features
    - interchangeable paths:

Detailed Evaluation

- **Floortile & TurnAndOpen:**
  - too expensive teacher search → sampling did not terminate
  - Floortile: dead-ends were successfully prevented
- **Floortile, Sokoban & TurnAndOpen:**
  - indecisive movement → inverting actions are applied
    - limited receptive field: additional input features
    - interchangeable paths: symmetry pruning

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
    - convincing training data

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
  - convincing training data
    - training terminated before one hour for Blocksworld and Hanoi
    - stable, considerable success rate above 70%

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
  - convincing training data
    - training terminated before one hour for Blocksworld and Hanoi
    - stable, considerable success rate above 70%
  - but: limited generalisation

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
  - convincing training data
    - training terminated before one hour for Blocksworld and Hanoi
    - stable, considerable success rate above 70%
  - but: limited generalisation
  - **Blocksworld & Hanoi:** problem solutions are too diverse

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
  - convincing training data
    - training terminated before one hour for Blocksworld and Hanoi
    - stable, considerable success rate above 70%
  - but: limited generalisation
  - **Blocksworld & Hanoi:** problem solutions are too diverse
  - **ParcPrinter:**

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
  - convincing training data
    - training terminated before one hour for Blocksworld and Hanoi
    - stable, considerable success rate above 70%
  - but: limited generalisation
  - **Blocksworld & Hanoi:** problem solutions are too diverse
  - **ParcPrinter:**
    - complex task with various components
    - scheduling is learned almost perfectly

Detailed Evaluation

- **Blocksworld, Hanoi & ParcPrinter:**
  - convincing training data
    - training terminated before one hour for Blocksworld and Hanoi
    - stable, considerable success rate above 70%
  - but: limited generalisation
  - **Blocksworld & Hanoi:** problem solutions are too diverse
  - **ParcPrinter:**
    - complex task with various components
    - scheduling is learned almost perfectly
    - only images are frequently printed on the wrong sheets

Detailed Evaluation

- **Tyreworld:**

Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent

Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance

Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance
  - **but:** even for Tyreworld ASNets are not perfect

Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance
  - **but:** even for Tyreworld ASNets are not perfect
- **Model creation:**

Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance
  - **but:** even for Tyreworld ASNets are not perfect
- **Model creation:**
  - ASNets contain one module for each grounding in every layer

Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance
  - **but:** even for Tyreworld ASNets are not perfect
- **Model creation:**
  - ASNets contain one module for each grounding in every layer
    $\rightarrow$ potentially very large networks

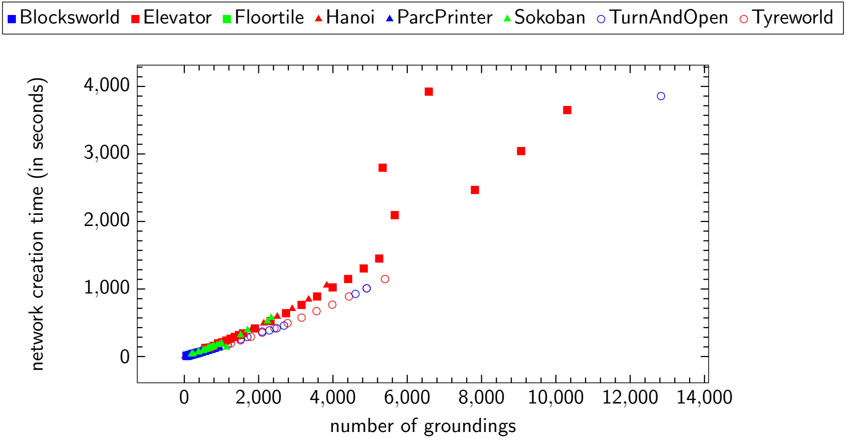
Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance
  - **but:** even for Tyreworld ASNets are not perfect
- **Model creation:**
  - ASNets contain one module for each grounding in every layer
    $\rightarrow$ potentially very large networks
    - considerable memory consumption
    - long network creation time

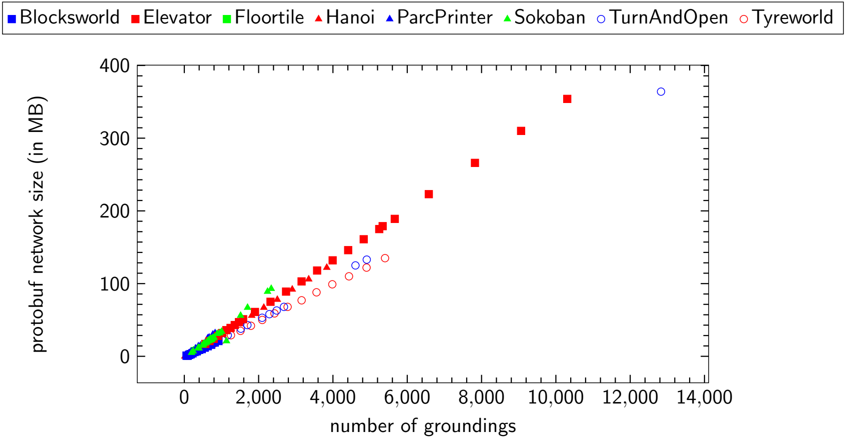
Detailed Evaluation

- **Tyreworld:**
  - only domain ASNets generalise well on
  - entirely repetitious pattern for each tyre can be learned
  - subproblems of replacing each tyre are independent
    $\rightarrow$ indecisiveness is not harming the performance
  - **but:** even for Tyreworld ASNets are not perfect
- **Model creation:**
  - ASNets contain one module for each grounding in every layer
    $\rightarrow$ potentially very large networks
    - considerable memory consumption
    - long network creation time
  - for each evaluated problem a network is necessary

Model Creation



Model Creation



Loss development

- similarly loss values do not further decrease

Loss development

- similarly loss values do not further decrease

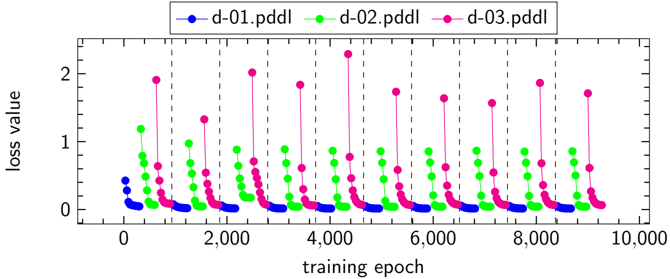


Figure: Loss development for the Hanoi domain with A* h^{add} teacher

Loss development

- similarly loss values do not further decrease

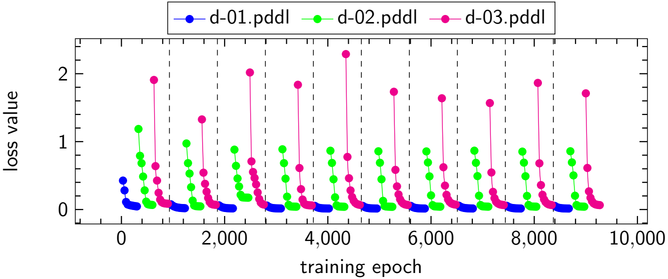


Figure: Loss development for the Hanoi domain with A* h^{add} teacher

- loss values are very volatile → seem to overfit for problems

Conclusion

Future work

- **Additional input features**

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy
- **Improved network search:**

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy
- **Improved network search:**
  - search with backtracking

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy
- **Improved network search:**
  - search with backtracking
  - combine ASNets with heuristics

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy
- **Improved network search:**
  - search with backtracking
  - combine ASNets with heuristics
    1. use network policy probabilities for tiebreaking

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy
- **Improved network search:**
  - search with backtracking
  - combine ASNets with heuristics
    1. use network policy probabilities for tiebreaking
    2. for $s \xrightarrow{a} s'$: combine $\pi^{\theta}(a \mid s), h(s')$ and potentially $g(s')$

Future work

- **Additional input features**
  - heuristic landmark features as proposed by Sam Toyer
  - action costs
- **Sampling:**
  - most sampled states are taken from teacher search trajectories
    $\rightarrow$ network policy is trained to imitate the teacher search
  - uniform sampling strategy
- **Improved network search:**
  - search with backtracking
  - combine ASNets with heuristics
    1. use network policy probabilities for tiebreaking
    2. for $s \xrightarrow{a} s'$: combine $\pi^{\theta}(a \mid s), h(s')$ and potentially $g(s')$
  - add pruning techniques to remove symmetric, interchangeable states

Conclusion

- contributions:
  - ASNet definition using *Keras*
  - Fast-Downward extension for ASNet relations and policies
  - training cycle with novel sampling search
  - extensive evaluation of ASNets for classical planning

Conclusion

- contributions:
  - ASNet definition using *Keras*
  - Fast-Downward extension for ASNet relations and policies
  - training cycle with novel sampling search
  - extensive evaluation of ASNets for classical planning
- only impressive performance on Tyreworld domain

Conclusion

- contributions:
  - ASNet definition using *Keras*
  - Fast-Downward extension for ASNet relations and policies
  - training cycle with novel sampling search
  - extensive evaluation of ASNets for classical planning
- only impressive performance on Tyreworld domain
- considerable learning could be found for further domains

Conclusion

- contributions:
  - ASNet definition using *Keras*
  - Fast-Downward extension for ASNet relations and policies
  - training cycle with novel sampling search
  - extensive evaluation of ASNets for classical planning
- only impressive performance on Tyreworld domain
- considerable learning could be found for further domains
  → shortcomings of the current approach were identified

Conclusion

- contributions:
  - ASNet definition using *Keras*
  - Fast-Downward extension for ASNet relations and policies
  - training cycle with novel sampling search
  - extensive evaluation of ASNets for classical planning
- only impressive performance on Tyreworld domain
- considerable learning could be found for further domains
  $\rightarrow$ shortcomings of the current approach were identified
- modifications for further research were proposed

Conclusion

- contributions:
  - ASNet definition using *Keras*
  - Fast-Downward extension for ASNet relations and policies
  - training cycle with novel sampling search
  - extensive evaluation of ASNets for classical planning
- only impressive performance on Tyreworld domain
- considerable learning could be found for further domains
  $\rightarrow$ shortcomings of the current approach were identified
- modifications for further research were proposed

The final verdict regarding the suitability of ASNets for classical planning is still outstanding