Neural Networks for Predicting Algorithm Runtime Distributions

Katharina Eggensperger, Marius Lindauer & Frank Hutter
Motivation

Algorithm portfolios yield state-of-the-art performance for SAT, ASP, Planning, …

→ to build these we can make use of runtime predictions

Other applications:

- Optimal restarts
- Algorithm selection
- Algorithm configurations
Describing the Runtime of an Algorithm?

```python
solve(instance, seed):
    # do something
    return solution, runtime
```
Describing the Runtime of an Algorithm?

```python
solve(instance, seed):
    # do something
    return solution, runtime
```
Contributions

1. Study how to predict parametric RTDs

2. Propose DistNet, a practical neural network for predicting RTDs

3. Evaluate DistNet and show that it can learn from only a few samples per instance
Typical Pipeline for Runtime prediction

Algorithm

Training instances

Run algorithm multiple times on each instance

Compute instance features
Typical Pipeline for Runtime prediction

Algorithm
Training instances

Run algorithm multiple times on each instance
Compute instance features

Data

Estimate RTD family
Typical Pipeline for Runtime prediction

1. Algorithm
2. Training instances
3. Run algorithm multiple times on each instance
4. Compute instance features
5. Data
6. Estimate RTD family
7. Fit RTD model
Typical Pipeline for Runtime prediction

1. Algorithm
2. Training instances
3. Run algorithm multiple times on each instance
4. Compute instance features
5. Data
6. Estimate RTD family
7. Fit RTD model
8. New instance
9. Compute features
10. Use model to predict parameters
Typical Pipeline for Runtime prediction

1. **Algorithm**
2. **Training instances**
3. **Run algorithm multiple times on each instance**
4. **Compute instance features**
5. **Data**
6. **Estimate RTD family**
7. **Fit RTD model**
8. **Use model to predict parameters**
9. **New instance**
10. **Compute features**
Empirical RTDs

Clasp-factoring

LPG-Zenotravel

SAPS-CV-VAR

$p(\text{solved by this time})$

runtime [sec]

runtime [sec]

runtime [sec]

$p(solved \ by \ this \ time)$
Typical Pipeline for Runtime prediction

1. Algorithm
2. Training instances
   - Run algorithm multiple times on each instance
3. Compute instance features
4. Data
5. Estimate RTD family
6. New instance
   - Compute features
7. Fit RTD model
8. Use model to predict parameters
Considered Parametric Distribution

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Param.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal (N)</td>
<td>$\mu, \sigma$</td>
</tr>
<tr>
<td>Lognormal (LOG)</td>
<td>$s, \sigma$</td>
</tr>
<tr>
<td>Exponential (EXP)</td>
<td>$\beta$</td>
</tr>
<tr>
<td>Inverse Normal (INV)</td>
<td>$\mu, \lambda$</td>
</tr>
</tbody>
</table>
Quantifying the Quality of Runtime Distributions

\[
\mathcal{L}_D(\theta \mid t(\pi)_1, \ldots, t(\pi)_k) = \prod_{i=1}^{k} p_D(t(\pi)_i \mid \theta) \tag{1}
\]

observed runtimes

distribution parameter
Quantifying the Quality of Runtime Distributions

\[ \mathcal{L}_D(\theta \mid t(\pi)_1, \ldots, t(\pi)_k) = \prod_{i=1}^{k} p_D(t(\pi)_i \mid \theta) \]  

(1)

\[ -\log \mathcal{L}_D(\theta \mid t(\pi)_1, \ldots, t(\pi)_k) = -\sum_{i=1}^{k} \log p_D(t(\pi)_i \mid \theta) \]  

(2)
Typical Pipeline for Runtime prediction

1. Algorithm
2. Training instances
3. Run algorithm multiple times on each instance
4. Compute instance features
5. Data
6. Estimate RTD family
7. Fit RTD model
8. Use model to predict parameters
9. New instance
10. Compute features
Predicting multiple Runtime Distributions

Option 1

For each training instance
→ fit the parametric distribution’s parameter on observed runtimes.

Then for all training instances, for each distribution parameter:
  fit a model
Predicting multiple Runtime Distributions

Option 1

For each training instance
→ fit the parametric distribution’s parameter on observed runtimes.

Then for all training instances, for each distribution parameter:

**fit a model**

Problematic, because models
- can only be as good as each fitted distribution
- do not know about interaction between their outputs
- typically minimize loss in the parameter space
Predicting multiple Runtime Distributions

Option 2

For each training instance → fit the parametric distribution’s parameter on observed runtimes.

Then for all training instances, for each distribution parameter:

**fit a model with multiple outputs**

Problematic, because model

- can only be as good as each fitted distribution
- does not know about interaction between their outputs
- typically minimizes loss in the parameter space
Predicting multiple Runtime Distributions

DistNet

For each training instance

→ fit the parametric distribution’s parameter on observed runtimes.

Then for all training instances, for each distribution parameter:

fit a neural network using negative log-likelihood as a loss function
Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>dist</th>
<th>iRF</th>
<th>mRF</th>
<th>DistNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saps-CV-VAR</td>
<td>LOG</td>
<td>0.99</td>
<td>-0.29</td>
<td>-0.52</td>
</tr>
</tbody>
</table>

We compared

- DistNet
- independent Random Forests (iRF)
- multi-output Random Forests (mRF)

on 7 scenarios from SAT solving and AI planning.

Figure: Averaged negative log-likelihood. Smaller values are better.
Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>dist</th>
<th>iRF</th>
<th>mRF</th>
<th>DistNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saps-CV-VAR</td>
<td>LOG</td>
<td>0.99</td>
<td>-0.29</td>
<td>-0.52</td>
</tr>
<tr>
<td></td>
<td>INV</td>
<td>0.22</td>
<td>-0.09</td>
<td>-0.54</td>
</tr>
</tbody>
</table>

Figure: Averaged negative log-likelihood. Smaller values are better.

We compared

- DistNet
- independent Random Forests (iRF)
- multi-output Random Forests (mRF)

on 7 scenarios from SAT solving and AI planning.
Results

We compared

- DistNet
- independent Random Forests (iRF)
- multi-output Random Forests (mRF)

on 7 scenarios from SAT solving and AI planning.

→ Predicting parameters for RTDs is possible
→ Joint predictions work better
→ DistNet provides more robust predictions which are often better than those of competitors

<table>
<thead>
<tr>
<th>Scenario</th>
<th>dist</th>
<th>iRF</th>
<th>mRF</th>
<th>DistNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saps-CV-VAR</td>
<td>LOG</td>
<td>0.99</td>
<td>-0.29</td>
<td>-0.52</td>
</tr>
<tr>
<td></td>
<td>INV</td>
<td>0.22</td>
<td>-0.09</td>
<td>-0.54</td>
</tr>
<tr>
<td>Clasp-factoring</td>
<td>INV</td>
<td>-0.04</td>
<td>-0.09</td>
<td>-0.16</td>
</tr>
<tr>
<td></td>
<td>LOG</td>
<td>-0.14</td>
<td>-0.13</td>
<td>-0.14</td>
</tr>
<tr>
<td>LPG-Zenotravel</td>
<td>LOG</td>
<td>-0.85</td>
<td>-0.84</td>
<td>-0.85</td>
</tr>
<tr>
<td></td>
<td>INV</td>
<td>-0.72</td>
<td>-0.80</td>
<td>-0.84</td>
</tr>
</tbody>
</table>

Figure: Averaged negative log-likelihood. Smaller values are better.
DistNet on a Low Number of Observations

DistNet: Runtime Distribution Prediction #2772

Eggenşperger, Lindauer and Hutter

IJCAI’2018

YalSAT-QCP

multi-output Random Forest

Distribution fitted on all samples

averaged NLLH

#samples per train instance
We have proposed DistNet, which

+ jointly learns distribution parameters
+ directly optimizes the loss function of interest
+ performs well even if only few observations per instance are available
Wrap-Up

We have proposed DistNet, which

+ **jointly learns** distribution parameters
+ directly optimizes the **loss function of interest**
+ performs well even if **only few observations per instance** are available

Open Questions:

- How to automatically determine a well fitting distribution family?
- How to handle heterogeneous datasets?
We have proposed DistNet, which

+ **jointly learns** distribution parameters
+ directly optimizes the **loss function of interest**
+ performs well even if **only few observations per instance** are available

Open Questions:

- How to automatically determine a well fitting distribution family?
- How to handle heterogeneous datasets?

Code and data: [https://www.automl.org/distnet/](https://www.automl.org/distnet/)