Learning From Interpretation Transition using Feed-Forward Neural Networks (NN-LFIT)

Enguerrand Gentet\textsuperscript{1,2} Sophie Tourret\textsuperscript{2} Katsumi Inoue\textsuperscript{2,3}

\textsuperscript{1}ENS (Cachan)/Paris-Sud University (Orsay) France,
\textsuperscript{2}NII (Tokyo) Japan,
\textsuperscript{3}Tokyo Institute of Technology (Tokyo), Japan

5\textsuperscript{th} September, ILP 2016
Table of Contents

1 Problem description
   - Context

2 NN-LFIT algorithm
   - Neural Network specification
   - Model training
   - Rules extraction

3 Experiments and results
   - Data
   - Model performance
   - Rule analysis
1. Problem description
   - Context

2. NN-LFIT algorithm
   - Neural Network specification
   - Model training
   - Rules extraction

3. Experiments and results
   - Data
   - Model performance
   - Rule analysis
Learning From Interpretation Transition

Vector-Representation of dynamical systems

A system is represented by a finite-state vector whose components evolve through time: $\forall t, \mathbf{x}(t) = (x_1(t), x_2(t), ..., x_{n_{var}}(t))$.

Goal of NN-LFIT

**Input:** $\{\mathbf{x}(t) \rightarrow \mathbf{x}(t + 1)\}$

**Output:** logic program describing the system dynamics

LFIT [Inoue, Ribeiro, and Sakama (2014)] vs NN-LFIT

- **LFIT:** the logic program is built directly from the observation of the system transitions in a purely logical framework.
- **NN-LFIT:** the learning of the model is separated from the extraction of the logic program.
Working assumptions

Assumptions

Boolean values: \( x \in \{0, 1\}^{n_{\text{var}}} \)

No delays: \( \forall t, x(t + 1) \) only depends on \( x(t) \)

Example

From a set of transitions...

\[ \{x(t) \rightarrow x(t + 1)\} \text{ with } \forall t, x(t) = (p(t), q(t), r(t)):\]

\[ (1, 1, 1) \rightarrow (1, 1, 0), (1, 1, 0) \rightarrow (1, 0, 0), (1, 0, 0) \rightarrow (0, 0, 0), \]

\[ (0, 0, 0) \rightarrow (0, 0, 1), (0, 0, 1) \rightarrow (0, 0, 1) \]
Assumptions

Boolean values: $\mathbf{x} \in \{0, 1\}^{n_{\text{var}}}$

No delays: $\forall t, \mathbf{x}(t + 1)$ only depends on $\mathbf{x}(t)$

Example

From a set of transitions...

$\{ \mathbf{x}(t) \rightarrow \mathbf{x}(t + 1) \}$ with $\forall t, \mathbf{x}(t) = (p(t), q(t), r(t))$:

$\{(1, 1, 1) \rightarrow (1, 1, 0), (1, 1, 0) \rightarrow (1, 0, 0), (1, 0, 0) \rightarrow (0, 0, 0),$

$(0, 0, 0) \rightarrow (0, 0, 1), (0, 0, 1) \rightarrow (0, 0, 1)\}$

...to a logic program.

$p \leftarrow q \quad q \leftarrow p \land r \quad r \leftarrow \neg p$
1 Problem description
   • Context

2 NN-LFIT algorithm
   • Neural Network specification
   • Model training
   • Rules extraction

3 Experiments and results
   • Data
   • Model performance
   • Rule analysis
Neural Network specification

**Neural Network**

**Type:** Feed-Forward Neural Network  
**Training:** back-propagation algorithm on MSE  
**Number of hidden layer:** 0, 1, 2, 3...  
**Number of hidden neurons:** T.B.D.

![Neural Network Diagram]

**Input layer**

\[ x_1(t) \rightarrow i_1 \]

\[ x_2(t) \rightarrow i_2 \]

\[ x_{n_{var}}(t) \rightarrow i_{n_{var}} \]

**Hidden layer**

\[ h_1 \]

\[ h_{n_{hid}} \]

\[ \vdots \]

**Output layer**

\[ o_1 \rightarrow x_1(t + 1) \]

\[ o_2 \rightarrow x_2(t + 1) \]

\[ o_{n_{var}} \rightarrow x_{n}(t + 1) \]
Neural Network specification

**Neural Network**

**Type:** Feed-Forward Neural Network  
**Training:** back-propagation algorithm on MSE  
**Number of hidden layers:** 0, 1, 2, 3...  
**Number of hidden neurons:** T.B.D.

![Neural Network Diagram]

\[x_1(t) \rightarrow i_1\]  
\[x_2(t) \rightarrow i_2\]  
\[x_{n\text{var}}(t) \rightarrow i_{n\text{var}}\]  

\[\rightarrow h_1\]  
\[\rightarrow h_2\]  
\[\rightarrow h_{n\text{hid}}\]  

\[\rightarrow o_1\]  
\[\rightarrow o_2\]  
\[\rightarrow o_{n\text{var}}\]  

\[\rightarrow x_1(t+1)\]  
\[\rightarrow x_2(t+1)\]  
\[\rightarrow x_{n}(t+1)\]
Constructive algorithm

Start with a small number of hidden neurons and add new ones as long as the performance are getting better.

\[ \text{err}_{\text{validation}} = 0.7 \]
**Number of hidden neurons**

**Constructive algorithm**

Start with a small number of hidden neurons and add new ones as long as the performance are getting better.

\[
\begin{align*}
\text{epoch 1: } & \quad p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \\
& \quad q(t) \rightarrow i_2 \rightarrow h_1 \rightarrow o_2 \rightarrow q(t+1) \\
& \quad r(t) \rightarrow i_3 \rightarrow h_1 \rightarrow o_3 \rightarrow r(t+1)
\end{align*}
\]

\[
\text{err}_{\text{validation}} = 0.7
\]

\[
\begin{align*}
\text{epoch 2: } & \quad p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \\
& \quad q(t) \rightarrow i_2 \rightarrow h_1 \rightarrow o_2 \rightarrow q(t+1) \\
& \quad r(t) \rightarrow i_3 \rightarrow h_1 \rightarrow o_3 \rightarrow r(t+1)
\end{align*}
\]

\[
\text{construction step}
\]

\[
\begin{align*}
\text{epoch 3: } & \quad p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \\
& \quad q(t) \rightarrow i_2 \rightarrow h_1 \rightarrow o_2 \rightarrow q(t+1) \\
& \quad r(t) \rightarrow i_3 \rightarrow h_1 \rightarrow o_3 \rightarrow r(t+1)
\end{align*}
\]

\[
\text{err}_{\text{validation}} = 0.6
\]

\[
\begin{align*}
\text{epoch 4: } & \quad p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \\
& \quad q(t) \rightarrow i_2 \rightarrow h_2 \rightarrow o_2 \rightarrow q(t+1) \\
& \quad r(t) \rightarrow i_3 \rightarrow h_4 \rightarrow o_3 \rightarrow r(t+1)
\end{align*}
\]

\[
\text{err}_{\text{validation}} = 0.4
\]
Constructive algorithm

Start with a small number of hidden neurons and add new ones as long as the performance are getting better.

\[ \text{err}_{\text{validation}} = 0.6 \]

\[ \text{err}_{\text{validation}} = 0.4 \]
Constructive algorithm

Start with a small number of hidden neurons and add new ones as long as the performance are getting better.
Constructive algorithm

Start with a small number of hidden neurons and add new ones as long as the performance are getting better.

\[ \text{err}_{\text{validation}} = 0.2 \]
Pruning step

Pruning algorithm

Simplify the NN by pruning useless links using a dichotomous search.

\[
\begin{align*}
p(t) &\rightarrow i_1 \\
q(t) &\rightarrow i_2 \\
r(t) &\rightarrow i_3 \\
h_1 &\rightarrow o_1 \\
h_2 &\rightarrow o_1 \\
h_3 &\rightarrow o_2 \\
h_4 &\rightarrow o_2 \\
o_1 &\rightarrow p(t+1) \\
o_2 &\rightarrow q(t+1) \\
o_3 &\rightarrow r(t+1) \\
err_{validation} &= 0.2
\end{align*}
\]
Pruning step

Pruning algorithm

Simplify the NN by pruning useless links using a dichotomous search.
Pruning step

Pruning algorithm

Simplify the NN by pruning useless links using a dichotomous search.

\[ \text{err}_{\text{validation}} = 0.2 \]

\[ \text{err}_{\text{validation}} = 0.3 \]
Pruning step

**Pruning algorithm**

Simplify the NN by pruning useless links using a dichotomous search.

\[ \text{err}_{\text{validation}} = 0.2 \]
Rule extraction: black box approach

For each output neuron...

\[ p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \]
\[ q(t) \rightarrow i_2 \rightarrow h_2 \rightarrow o_2 \rightarrow q(t+1) \]
\[ r(t) \rightarrow i_3 \rightarrow h_3 \rightarrow o_3 \rightarrow r(t+1) \]
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.

\[
\begin{align*}
p(t) &\rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \\
q(t) &\rightarrow i_2 \rightarrow h_2 \rightarrow o_2 \rightarrow q(t+1) \\
r(t) &\rightarrow i_3 \rightarrow h_3 \rightarrow o_3 \rightarrow r(t+1)
\end{align*}
\]
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.

\[ \begin{array}{c|c|c}
 p & q \\
 0 & 1 \\
 1 & 1 \\
\end{array} \]

\[
p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1)
\]

\[
q(t) \rightarrow i_2 \rightarrow h_2 \rightarrow o_2 \rightarrow q(t+1)
\]

\[
r(t) \rightarrow i_3 \rightarrow h_3 \rightarrow o_3 \rightarrow r(t+1)
\]
For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.

\[
\begin{array}{c|c|c}
  p & q & DNF \\
  0 & 1 & p \leftarrow (\neg p \land q) \lor (p \land q) \\
  1 & 1 & \\
\end{array}
\]

\[
p \overset{\text{DNF}}{\rightarrow} \]

Enguerrand Gentet, Sophie Tourret, Katsumi Inoue
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]

\[ DNF \]

\[ p \leftarrow (\neg p \land q) \lor (p \land q) \]

\[ p \leftarrow q \]

Enguerrand Gentet, Sophie Tourret, Katsumi Inoue

NN-LFIT
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]

\[
\begin{array}{c|c}
p & q \
\hline
0 & 1 \\
1 & 1 \\
\end{array}
\]

\[
p \leftarrow (\neg p \land q) \lor (p \land q)
\]

\[
p \leftarrow q
\]
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]

\[ p(t) \rightarrow i_1 \rightarrow h_1 \rightarrow o_1 \rightarrow p(t+1) \]
\[ q(t) \rightarrow i_2 \rightarrow h_2 \rightarrow o_2 \rightarrow q(t+1) \]
\[ r(t) \rightarrow i_3 \rightarrow h_3 \rightarrow o_3 \rightarrow r(t+1) \]

\[ \begin{array}{|c|c|} \hline p & q \\ \hline 0 & 1 \\ 1 & 1 \\ \hline \end{array} \]

\[ DNF \rightarrow p \leftarrow (\neg p \land q) \lor (p \land q) \]

\[ \begin{array}{|c|c|} \hline p & r \\ \hline 0 & 1 \\ 1 & 1 \\ \hline \end{array} \]

\[ DNF \rightarrow q \leftarrow p \land r \]

\[ p \leftarrow q \]

Enguerrand Gentet, Sophie Tourret, Katsumi Inoue
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]
Rule extraction: black box approach

For each output neuron...

1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]
Rule extraction: black box approach

For each output neuron...
1. Extract sub-NN.
2. Feed the sub-NN with every possible combination.
3. Recover the combinations that activate the neuron.
4. Generate DNF formula.
5. Use primer to simplify the rule. [Previti et al. 2015]
Sommaire

1 Problem description
   - Context

2 NN-LFIT algorithm
   - Neural Network specification
   - Model training
   - Rules extraction

3 Experiments and results
   - Data
   - Model performance
   - Rule analysis
Benchmarks

Boolean networks
(Dubrova and Teslenko 2011)

Mammalian cell-cycle regulation
\( n_{\text{var}} = 10 \)
Fission yeast cell-cycle regulation
\( n_{\text{var}} = 10 \)
Budding yeast cell-cycle regulation
\( n_{\text{var}} = 12 \)

Training and test sets

all transitions \( \rightarrow \) \{ training set, test set \}

Figure: fission yeast cell-cycle regulation network
(Ruz and Goles 2014)
Size of the training set

Influence of the size of the training set on the test error.

Figure: Mammalian cell-cycle regulation benchmark

Figure: Fission yeast cell-cycle regulation benchmark

Enguerrand Gentet, Sophie Tourret, Katsumi Inoue
Pruning step influence

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Mammalian, $n_{var} = 10$</th>
<th>Fission, $n_{var} = 10$</th>
<th>Budding, $n_{var} = 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Neurons</td>
<td>Links</td>
<td>Error (%)</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>13.5</td>
<td>270</td>
<td>1.92</td>
</tr>
<tr>
<td>Pruned</td>
<td>11.2</td>
<td>98.6</td>
<td>1.37</td>
</tr>
</tbody>
</table>

Table: Architecture and test error changes during the pruning.
size of the training set: 15%, results averaged on 40 runs
## Pruning step influence

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Mammalian, $n_{var} = 10$</th>
<th>Fission, $n_{var} = 10$</th>
<th>Budding, $n_{var} = 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Neurons</td>
<td>Links</td>
<td>Error (%)</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>13.5</td>
<td>270</td>
<td>1.92</td>
</tr>
<tr>
<td>Pruned</td>
<td>11.2</td>
<td>98.6</td>
<td>1.37</td>
</tr>
</tbody>
</table>

**Table:** Architecture and test error changes during the pruning. Size of the training set: 15%, results averaged on 40 runs

$n_{neurons} \downarrow : 23\%$
### Pruning step influence

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Neurons</th>
<th>Links</th>
<th>Error (%)</th>
<th>Neurons</th>
<th>Links</th>
<th>Error (%)</th>
<th>Neurons</th>
<th>Links</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Connected</td>
<td>13.5</td>
<td>270</td>
<td>1.92</td>
<td>13.73</td>
<td>275</td>
<td>1.61</td>
<td>14.4</td>
<td>346</td>
<td>0.237</td>
</tr>
<tr>
<td>Pruned</td>
<td>11.2</td>
<td>98.6</td>
<td>1.37</td>
<td>11.7</td>
<td>97.8</td>
<td>1.21</td>
<td>12.2</td>
<td>91</td>
<td>0.156</td>
</tr>
</tbody>
</table>

**Table:** Architecture and test error changes during the pruning. size of the training set: 15%, results averaged on 40 runs

\[ \text{n_{neurons}} \downarrow : 23\% \quad \text{n_{links}} \downarrow : 74\% \]
## Pruning step influence

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Mammalian, ( n_{\text{var}} = 10 )</th>
<th>Fission, ( n_{\text{var}} = 10 )</th>
<th>Budding, ( n_{\text{var}} = 12 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Neurons</td>
<td>Links</td>
<td>Error (%)</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>13.5</td>
<td>270</td>
<td>1.92</td>
</tr>
<tr>
<td>Pruned</td>
<td>11.2</td>
<td>98.6</td>
<td>1.37</td>
</tr>
</tbody>
</table>

**Table:** Architecture and test error changes during the pruning. Size of the training set: 15%, results averaged on 40 runs

- \( n_{\text{neurons}} \downarrow : 23\% \)
- \( n_{\text{links}} \downarrow : 74\% \)
- \( \text{error}_{\text{test}} \downarrow : 45\% \)
Pruning step influence

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Mammalian, ( n_{\text{var}} = 10 )</th>
<th>Fission, ( n_{\text{var}} = 10 )</th>
<th>Budding, ( n_{\text{var}} = 12 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neurons</td>
<td>13.5</td>
<td>13.73</td>
<td>14.4</td>
</tr>
<tr>
<td>Links</td>
<td>270</td>
<td>275</td>
<td>346</td>
</tr>
<tr>
<td>Error (%)</td>
<td>1.92</td>
<td>1.61</td>
<td>0.237</td>
</tr>
<tr>
<td>Neurons</td>
<td>11.2</td>
<td>11.7</td>
<td>12.2</td>
</tr>
<tr>
<td>Links</td>
<td>98.6</td>
<td>97.8</td>
<td>91</td>
</tr>
<tr>
<td>Error (%)</td>
<td>1.37</td>
<td>1.21</td>
<td>0.156</td>
</tr>
</tbody>
</table>

**Table:** Architecture and test error changes during the pruning. size of the training set: 15%, results averaged on 40 runs

\[ n_{\text{neurons}} \downarrow : 23\% \quad n_{\text{links}} \downarrow : 74\% \quad error_{\text{test}} \downarrow : 45\% \]

**Consequences of the pruning step**

- Complexity of the NN is reduced.
- Better model performance.
Simplicity of the rules

Reference rule:  $a \leftarrow (\neg a \land b) \lor (\neg b \land c)$
Inferred rule:  $a \leftarrow (\neg a \land b) \lor (b \land c)$
Simplicity of the rules

Reference rule: \( a \leftarrow (\neg a \land b) \lor (\neg b \land c) \)

Inferred rule: \( a \leftarrow (\neg a \land b) \lor (b \land c) \)  
valid terms
Simplicity of the rules

Reference rule: \( a \leftarrow (\neg a \land b) \lor (\neg b \land c) \)

Inferred rule: \( a \leftarrow (\neg a \land b) \lor (b \land c) \)  \hspace{1cm} \text{wrong terms}
Simplicity of the rules

Reference rule: \( a \leftarrow (\neg a \land b) \lor (\neg b \land c) \)

Inferred rule: \( a \leftarrow (\neg a \land b) \lor (b \land c) \)  
missing terms
Simplicity of the rules

Reference rule: $a \leftarrow (\neg a \land b) \lor (\neg b \land c)$
Inferred rule: $a \leftarrow (\neg a \land b) \lor (b \land c)$

Distribution of terms before and after the pruning step.

**Figure:** Mammalian cell-cycle regulation benchmark

**Figure:** Fission yeast cell-cycle regulation benchmark
In conclusion:

- Incomplete data are well handled by NN-LFIT.
- Pruning step improves accuracy and simplicity of the rules.
Ongoing and future work

**Ongoing work**

- Improvement: change the rule extraction algorithm to a decompositional approach [Garcez, Broda, and Gabbay 2001].
- Extension: train the model on real-valued data, encouraging results on DREAM4 challenge [Greenfield et al. 2010].

**Future work**

- Extension: rule extraction on real-valued data problems.
- Extension: manage delays, use of Recurrent Neural Networks.
Thank you for your attention. Do you have any question?
References


Alex Greenfield et al. “DREAM4: Combining genetic and dynamic information to identify biological networks and dynamical models”. In: PloS one 5.10 (2010), e13397.


Rule simplification

Simplify a DNF formula: compute a prime implicant cover.

**Cover**

\[ F_1 \text{ covers } F_2 \iff \begin{cases} \forall D_2 \in F_2, \exists D_1 \in F_1, D_1 \subseteq D_2 \\ F_1 \leftrightarrow F_2 \end{cases} \]

**Prime implicant**

\( D \) is a prime implicant of \( F \)

\( \iff \begin{cases} D \models F \\ \forall D', D' \models F, D \models D' \Rightarrow D' \models D \end{cases} \)

Consequence: \( \forall D'', D'' \subset D \Rightarrow D'' \not\models F \).

**Example**

\[ F = (x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2 \land \neg x_3) \lor (x_1 \land x_2 \land x_3) \]

\[ F' = (x_1 \land \neg x_3) \lor (x_1 \land x_2) \]
Use of primer

**primer [Previti et al. 2015]**

**Input:** CNF formula  
**Output:** prime implicate cover of the input formula

**Solving dual problem**

The notion of a prime implicate is dual to a prime implicant. To simplify a rule we solve the dual problem using primer.

**Example**

\[ F = (x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2 \land \neg x_3) \lor (x_1 \land x_2 \land x_3) \]
Use of primer [Previti et al. 2015]

**Input:** CNF formula

**Output:** prime implicate cover of the input formula

Solving dual problem

The notion of a prime implicate is dual to a prime implicant. To simplify a rule we solve the dual problem using primer.

**Example**

\[
\begin{align*}
F &= (x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2 \land \neg x_3) \lor (x_1 \land x_2 \land x_3) \\
\tilde{F} &= (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_3)
\end{align*}
\]
Use of primer

primer [Previti et al. 2015]

**Input:** CNF formula

**Output:** prime implicate cover of the input formula

Solving dual problem

The notion of a prime implicate is dual to a prime implicant. To simplify a rule we solve the dual problem using primer.

Example

\[
F = (x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2 \land \neg x_3) \lor (x_1 \land x_2 \land x_3)
\]

\[
\tilde{F} = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_3)
\]

\[
\tilde{F}' = (x_1 \lor \neg x_3) \land (x_1 \lor x_2)
\]
Use of primer

**primer [Previti et al. 2015]**

**Input:** CNF formula  
**Output:** prime implicate cover of the input formula

**Solving dual problem**

The notion of a prime implicate is dual to a prime implicant. To simplify a rule we solve the dual problem using primer.

**Example**

\[
F = (x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2 \land \neg x_3) \lor (x_1 \land x_2 \land x_3) \\
\tilde{F} = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_3) \\
\tilde{F}' = (x_1 \lor \neg x_3) \land (x_1 \lor x_2) \\
F' = (x_1 \land \neg x_3) \lor (x_1 \land x_2) \\
\]
Train the model on real-valued data

Figure: Example of DREAM4 time-series data on six genes of a network.

<table>
<thead>
<tr>
<th></th>
<th>1\textsuperscript{st} part.</th>
<th>NN-LFIT</th>
<th>2\textsuperscript{nd} part.</th>
<th>3\textsuperscript{rd} part.</th>
<th>4\textsuperscript{th} part.</th>
<th>5\textsuperscript{th} part.</th>
<th>...</th>
<th>LFkT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0228</td>
<td>0.0243</td>
<td>0.0272</td>
<td>0.0273</td>
<td>0.0277</td>
<td>0.0482</td>
<td>...</td>
<td>0.0543</td>
</tr>
</tbody>
</table>

Table: Global MSE obtained on the dual knockouts bonus round of the DREAM4 in-silico challenge.

"\textit{n}\textsuperscript{th} part.\textit{"} : participant at the \textit{n}\textsuperscript{th} place on the leaderboard.