Belief propagation on strings

Boris Yangel

Last modified: November 27, 2013

Abstract

This document describes a way of performing belief propagation in models with string variables, with factors corresponding to widely used operations on strings such as Substring, Contains, Replace, Concat etc.

Contents

1 Notation 1

2 Automata 2
  2.1 Atomic automata 2
  2.2 Operations on automata 2
  2.3 Examples of compound automata 2

3 Transducers 2
  3.1 Atomic transducers 3
  3.2 Operations on transducers 3
  3.3 Projection of an automaton onto a transducer 3
  3.4 Some properties of operations on transducers 3
  3.5 Examples of compound transducers 4

4 Message operators 4
  4.1 Substring 4
  4.2 Concat 5
  4.3 ContainsAlt 5

1 Notation

We denote automata by capital letters A, B etc or, in case of a specific automaton, by a string starting from a capital letter, such as Unif. However, when an automaton represents a BP message, we denote it by $\mu$ with a subscript describing the source and the destination of the message, like $\mu_f \rightarrow s$. Notation for transducers is the same, except that we use a different font: $T$, $U$, $Copy$. Lowercase letters and sequences of lowercase letters denote variables: $s$, $t$, $str$.

The notation $s \cdot t$ stands for the concatenation of strings $s$ and $t$. 
2 Automata

We represent marginals and messages from and to string variables by weighted
finite state automata. Weighted finite state automaton, which we’ll refer to simply
as automaton from now on, is a function mapping strings to real values. Not every
such function is a finite state automaton though. For an explanation of what a finite
state automaton really is and what are the limitations of this class of functions we
refer the reader to [?], while this paper takes a different approach. We instead define
a set of atomic automata, a number of operations w.r.t. which automata are closed,
and then simply use all the automata that can be obtained from the atomic ones
using the proposed operations.

2.1 Atomic automata

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform of characters</td>
<td>Unif_C</td>
<td>Maps every string consisting entirely of characters from the set ( C ) to 1 and all other strings to 0.</td>
</tr>
<tr>
<td>Single character</td>
<td>Char_C</td>
<td>Maps every string consisting of a single character from the set ( C ) to 1 and all other strings to 0.</td>
</tr>
</tbody>
</table>

If the set \( C \) consists of all possible characters, we denote Unif\_C simply by Unif, and Char\_C by Char.

2.2 Operations on automata

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>( \alpha A )</td>
<td>((\alpha A)(s) := \alpha A(s))</td>
</tr>
<tr>
<td>Sum</td>
<td>( A + B )</td>
<td>((A + B)(s) := A(s) + B(s))</td>
</tr>
<tr>
<td>Product</td>
<td>( AB )</td>
<td>((AB)(s) := A(s)B(s))</td>
</tr>
<tr>
<td>Concatenation</td>
<td>( A \cdot B )</td>
<td>((A \cdot B)(s) := \sum_{s_1 \cdot s_2 = s} A(s_1)B(s_2)).</td>
</tr>
<tr>
<td>Normalizer</td>
<td>( \mathcal{Z}(A) )</td>
<td>( \mathcal{Z}(A) := \sum_s A(s) ) (defined only when the corresponding sum converges).</td>
</tr>
</tbody>
</table>

We denote \( A + (\neg 1)B \) simply by \( A - B \).

2.3 Examples of compound automata

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OfLen_N</td>
<td>( \underbrace{\text{Char} \cdot \text{Char} \cdot \ldots \cdot \text{Char}}_{N \text{ times}} )</td>
<td>1 on any string of length ( N ), zero on everything else.</td>
</tr>
</tbody>
</table>

3 Transducers

In order to compute messages from a factor to a variable, we also need to employ
weighted finite state transducers, which are, essentially, functions mapping pairs
of sequences to real values. Following the section 2 we define the set of atomic
transducers and a number of operations that allows us to produce all the transducers
of interest. For a more general introduction, we refer the reader to [?].
3.1 Atomic transducers

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Copy</td>
<td>Copy((s, t) := \begin{cases} 1, &amp; s = t \ 0, &amp; \text{otherwise} \end{cases} )</td>
</tr>
<tr>
<td>Copy char</td>
<td>CopyChar</td>
<td>CopyChar((s, t) := \begin{cases} 1, &amp; s = t,</td>
</tr>
<tr>
<td>Consume an automaton</td>
<td>ConsumeA</td>
<td>ConsumeA((s, t) := \begin{cases} A(s), &amp; t = \varepsilon \ 0, &amp; \text{otherwise} \end{cases} )</td>
</tr>
<tr>
<td>Produce an automaton</td>
<td>ProduceA</td>
<td>ProduceA((s, t) := \begin{cases} A(t), &amp; s = \varepsilon \ 0, &amp; \text{otherwise} \end{cases} )</td>
</tr>
</tbody>
</table>

3.2 Operations on transducers

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td>( \mathcal{A} \cdot \mathcal{B} )</td>
<td>((\mathcal{A} \cdot \mathcal{B})(s, t) := \sum_{s_1, t_1, s_2, t_2} A(s_1, t_1)B(s_2, t_2) )</td>
</tr>
<tr>
<td>Transpose</td>
<td>( \mathcal{A}^T )</td>
<td>((\mathcal{A}^T)(s, t) := \mathcal{A}(t, s) )</td>
</tr>
<tr>
<td>Superposition</td>
<td>( \mathcal{B}(\mathcal{A}) )</td>
<td>((\mathcal{B}(\mathcal{A})(s, t) := \sum_u A(s, u)B(u, t) )</td>
</tr>
</tbody>
</table>

3.3 Projection of an automaton onto a transducer

Additionaly, an automaton can be projected onto a transducer. The result of the projection is another automaton, which is defined as

\[
\text{proj}[A, \mathcal{T}](t) := \sum_s A(s)\mathcal{T}(s, t). \tag{1}
\]

This operation can be very handy when computing BP messages.

3.4 Some properties of operations on transducers

**Property 1.**

\[
\text{proj}[A, \mathcal{U}(\mathcal{T})] = \text{proj}[\text{proj}[A, \mathcal{T}], \mathcal{U}]. \tag{2}
\]

**Proof.**

\[
\text{proj}[A, \mathcal{U}(\mathcal{T})](s, t) = \sum_s A(s)\sum_u \mathcal{T}(s, u)\mathcal{U}(u, t) = \sum_u \mathcal{U}(u, t)\sum_s A(s)\mathcal{T}(s, u) = \sum_u \mathcal{U}(u, t)\text{proj}[A, \mathcal{T}](u) = \text{proj}[\text{proj}[A, \mathcal{T}], \mathcal{U}]. \tag{3}
\]

**Property 2.**

\[
\mathcal{Z}(A) = \text{proj}[\text{Unif}, \text{Consume}_A \cdot \text{Produce}_\text{Unif}](t) \quad \forall t. \tag{4}
\]
Proof. \[ \forall t : \mathbb{Z}(\Lambda) = \sum_s \Lambda(s) = \sum_s \text{Unif}(s)\tilde{\Lambda}(s,t), \quad (5) \]
where \[ \tilde{\Lambda}(s,t) = \Lambda(s) = \text{Consume}_\Lambda \cdot \text{Produce}_\text{Unif}(s,t). \quad (6) \]

3.5 Examples of compound transducers
Here we present a number of compound transducers that are useful for message computation.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occurrences((s, str))</td>
<td>( \text{Produce}<em>\text{Unit} \cdot \text{Copy} \cdot \text{Produce}</em>\text{Unif} )</td>
<td>The number of occurrences of ( s ) in ( str ).</td>
</tr>
<tr>
<td>OccursAt(_p)((s, str))</td>
<td>( \text{Produce}_\text{OfLen}<em>p \cdot \text{Copy} \cdot \text{Produce}</em>\text{Unif} )</td>
<td>1 if ( s ) occurs in ( str ) at the position ( p ), 0 otherwise.</td>
</tr>
<tr>
<td>CopyOfLen(_N)((s_1, s_2))</td>
<td>( \underbrace{\text{CopyChar} \cdot \text{CopyChar} \cdot \ldots \cdot \text{CopyChar}}_{N \text{ times}} )</td>
<td>1 if ( s_1 = s_2 ) and both strings are of length ( N ).</td>
</tr>
<tr>
<td>IsSubstring(_p,l)((s, str))</td>
<td>( \underbrace{\text{Produce}_\text{OfLen}<em>p \cdot \text{CopyOfLen}<em>l \cdot \text{Produce}</em>\text{Unif}}</em>{N \text{ times}} )</td>
<td>1 if ( s ) occurs in ( str ) from the position ( p ) to ( p+l ), 0 otherwise.</td>
</tr>
</tbody>
</table>

4 Message operators

4.1 Substring

4.1.1 Factor

\[ f(str, s, p, l) = \mathbb{1}[str \text{ contains } s \text{ from } p \text{ to } p+l]. \quad (7) \]
We assume that the position \( p \) and the length \( l \) are observed.

4.1.2 Message to \( str \)

\[ \mu_{f \rightarrow str}(str) = \sum_s \mu_{s \rightarrow f}(s)\mathbb{1}[str \text{ contains } s \text{ from } p \text{ to } p+l] = \]
\[ = \text{proj}[\mu_{s \rightarrow f}, \text{IsSubstring}\_p,l]. \quad (8) \]

4.1.3 Message to \( s \)

\[ \mu_{f \rightarrow s}(s) = \sum_{str} \mu_{str \rightarrow f}(s)\mathbb{1}[str \text{ contains } s \text{ from } p \text{ to } p+l] = \]
\[ = \text{proj}[\mu_{str \rightarrow f}, \text{IsSubstring}\_p,l]. \quad (9) \]
4.2 Concat

4.2.1 Factor

\[ f(s, s_1, s_2) = \mathbb{1}[s = s_1 \cdot s_2]. \]  

(10)

4.2.2 Message to \( s \)

\[ \mu_{f \rightarrow s}(s) = \sum_{s_1} \mu_{s_1 \rightarrow f}(s_1) \sum_{s_2} \mu_{s_2 \rightarrow f}(s_2) \mathbb{1}[[s = s_1 \cdot s_2] = \mu_{s_1 \rightarrow f} \cdot \mu_{s_2 \rightarrow f}. \]  

(11)

4.2.3 Message to \( s_1 \)

\[ \mu_{f \rightarrow s_1}(s_1) = \sum_{s} \mu_{s \rightarrow f}(s) \sum_{s_2} \mu_{s_2 \rightarrow f}(s_2) \mathbb{1}[s = s_1 \cdot s_2] = \]  

\[ = \text{proj}[\mu_{s \rightarrow f}, \text{Copy} \cdot \text{Consume}_{\mu_{s_2 \rightarrow f}}]. \]  

(12)

4.2.4 Message to \( s_2 \)

\[ \mu_{f \rightarrow s_2}(s_2) = \sum_{s} \mu_{s \rightarrow f}(s) \sum_{s_1} \mu_{s_1 \rightarrow f}(s_1) \mathbb{1}[s = s_1 \cdot s_2] = \]  

\[ = \text{proj}[\mu_{s \rightarrow f}, \text{Consume}_{\mu_{s_1 \rightarrow f}} \cdot \text{Copy}]. \]  

(13)

4.3 ContainsAt

4.3.1 Factor

\[ f(\text{str}, s, p, c) = \begin{cases} \mathbb{1}[\text{str contains } s \text{ at pos } p] & \text{if } c = 1 \\ (1 - \mathbb{1}[\text{str contains } s \text{ at pos } p]) & \text{if } c = 0 \end{cases} \]  

\[ + (1 - c)(1 - \mathbb{1}[\text{str contains } s \text{ at pos } p]). \]  

(14)

We assume that the position \( p \) is always observed.

4.3.2 Message to \( c \)

\[ \mu_{f \rightarrow c}(c = 1) = \sum_{\text{str}} \mu_{\text{str} \rightarrow f}(\text{str}) \sum_{s} \mu_{s \rightarrow f}(s) \mathbb{1}[\text{str contains } s \text{ at pos } p] = \]  

\[ = \mathcal{Z}(\mu_{\text{str} \rightarrow f} \text{proj}[\mu_{s \rightarrow f}, \text{OccursAt}_p]). \]  

(15)

\[ \mu_{f \rightarrow c}(c = 0) = \sum_{\text{str}} \mu_{\text{str} \rightarrow f}(\text{str}) \sum_{s} \mu_{s \rightarrow f}(s)(1 - \mathbb{1}[\text{str contains } s \text{ at pos } p]) = \]  

\[ = \mathcal{Z}(\mu_{\text{str} \rightarrow f}) \mathcal{Z}(\mu_{s \rightarrow f}) - \mathcal{Z}(\mu_{\text{str} \rightarrow f} \text{proj}[\mu_{s \rightarrow f}, \text{OccursAt}_p]). \]  

(16)
4.3.3 Message to \textit{str}

\[
\mu_{f\rightarrow\text{str}}(\text{str}) = \mu_{c\rightarrow f}(c = 1) \sum_s \mu_{s\rightarrow f}(s)\mathbb{I}[\text{str contains } s \text{ at pos } p] + \\
+ \mu_{c\rightarrow f}(c = 0) \sum_s \mu_{s\rightarrow f}(s)(1 - \mathbb{I}[\text{str contains } s \text{ at pos } p]) = \\
= (\mu_{c\rightarrow f}(c = 1) - \mu_{c\rightarrow f}(c = 0)) \text{ proj}[\mu_{s\rightarrow f}, \text{OccursAt}_p] + \mu_{c\rightarrow f}(c = 0)\text{Unif}. \quad (17)
\]

4.3.4 Message to \textit{s}

\[
\mu_{f\rightarrow s}(s) = \mu_{c\rightarrow f}(c = 1) \sum_{\text{str}} \mu_{\text{str}\rightarrow f}(\text{str})\mathbb{I}[\text{str contains } s \text{ at pos } p] + \\
+ \mu_{c\rightarrow f}(c = 0) \sum_{\text{str}} \mu_{\text{str}\rightarrow f}(\text{str})(1 - \mathbb{I}[\text{str contains } s \text{ at pos } p]) = \\
= (\mu_{c\rightarrow f}(c = 1) - \mu_{c\rightarrow f}(c = 0)) \text{ proj}[\mu_{\text{str}\rightarrow f}, \text{OccursAt}_p] + \mu_{c\rightarrow f}(c = 0)\text{Unif}. \quad (18)
\]