RNNLIB: Connectionist Temporal Classification and Transcription Layer

Wantee Wang

2015-02-08 16:56:40 +0800

Contents

CTC is the core concept make it possible to transcribe unsegmented sequence data. RNNLIB implements it in a single layer called Transcription Layer. We go into this particular layer in this post, the main reference is the Graves' original paper.

The key point for CTC is to use a simple map transforming the RNN output to unsegmented labelling, and construct a new objective function based on the map. This map do not need a precise alignment, thus greatly simplify the task and reduce human expert involvement.

1 The Name

"Connectionist" is the adjective form of "connectionism", Connectionism is a terminology in cognitive science, which models mental or behavioural phenomena as the emergent processes of interconnected networks of simple units. The most common forms use neural network models.

In the traditional neural network recipe, we independently model the input sequence in each time-step or frame. This can be referred as *framewise classification*. Kadous extends the classification paradigm to multivariate time series, and names it as *temporal classification*. Mathematically, framewise classification models the distribution over output sequences of the *same* length as the input sequence, nevertheless, temporal classification models the distribution over output sequences of *all* lengths. With this, we do not have to label every time step in training data set.

Combining RNN and temporal classification, Graves proposes the *connectionist temporal classification*.

To distinguish from classification, RNNLIB implements the CTC as *Transcription Layer*, indicating that with CTC we can directly transcribe input sequence(e.g. acoustic signal) into output sequence(e.g. words).

2 The Theory

2.1 List of Symbols

Following the notations in the paper, we first list the symbols.

Symbol	Meaning
L	(finite) alphabet of labels
L'	$L \cup \{blank\}$
$\mathcal{X}$	$(\mathbb{R}^m)^*$, m dimensional input space
$\mathcal{Z}$	L^*, output space, set of all sequences over the L
$\mathcal{D}_{\mathcal{X}\times\mathcal{Z}}$	underlying distribution of data
S	set of training examples supposed to be drawn from $\mathcal{D}_{\mathcal{X}\times\mathcal{Z}}$
$(\mathbf{x}, \mathbf{z})$	example in S, $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, $\mathbf{z} = (z_1, z_2, \ldots, z_U)$ and $U \leq T$
$h : \mathcal{X} \mapsto \mathcal{Z}$	temporal classifier to be trained
$\mathcal{N}_w : (R^m)^T \mapsto (R^n)^T$	RNN, with m inputs, n outputs and weight vector w, as a continuous map
$\mathbf{y} = \mathcal{N}_w$	sequence of RNN output
y_k^t	the activation of output unit k at time t
π	*path*, element of L'^T
$\mathbf{l} \in L^{\leq T}$	label sequence or *labelling*
$\mathcal{B} : L'^T \mapsto L^{\leq T}$	map from path to labelling
$\mathbf{l}_{a:b}$	sub-sequence of $\mathbf{l}$ from ath to bth labels

Symbol	Meaning
$\mathbf{l}'$	modified label sequence, with blanks added to the beginning and the end and inserte
$\alpha_t(s)$	forward variable, the total probability of $\mathbf{l}_{1:s}$ at time t
$\beta_t(s)$	backward variable, the total probability of $\mathbf{l}_{s:\|\mathbf{l}'\|}$ at time t
$\tilde{\beta}_t(s)$	backward variable, the total probability of $\mathbf{l}_{s:\|\mathbf{l}'\|}$ start at time $t+1$
$O^{ML}(S, \mathcal{N}_w)$	maximum likelihood objective function
$\delta_{kk'}$	Kronecker delta

2.2 Training Procedure

The goal is to use S to train a temporal classifier h to classify previously unseen input sequences in a way that minimises the ML objective function:

$$O^{ML}(S, \mathcal{N}_w) = -\sum_{(\mathbf{x},\mathbf{z})\in S} \ln(p(\mathbf{z}|\mathbf{x})) \tag{1}$$

To train the network with gradient descent, we need to differentiate (1) with respect to the network outputs. Since the training examples are independent we can consider them separately:

$$\frac{\partial O^{ML}(\{(\mathbf{x},\mathbf{z}\}, \mathcal{N}_w)}{\partial y_k^t} = -\frac{\partial \ln(p(\mathbf{z}|\mathbf{x}))}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})}\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} \tag{2}$$

Another thing we have to consider is how to map from network outputs to labellings. Use $\mathcal{B}$ to denote such a map. Given a path, we simply removing all blanks and repeated labels and the remaining labels form a labelling(e.g. $\mathcal{B}(a-ab-) = \mathcal{B}(-aa--abb) = aab$).

Then we can define the conditional probability of a labelling,

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi\in\mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}) \tag{3}$$

where, $p(\pi|\mathbf{x})$ is the conditional probability of a path given $\mathbf{x}$, and is defined as:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^t, \forall\pi \in L'^T \tag{4}$$

To calculate (2), we first define the forward and backward variable,

$$\alpha_t(s) = \sum_{\pi\in L^T:\mathcal{B}(\pi_{1:t})=\mathbf{l}_{1:s}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'} \tag{5}$$

$$\beta_t(s) = \sum_{\pi \in L^T : \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}'|}} \prod_{t'=t}^{T} y^{t'}_{\pi_{t'}} \tag{6}$$

Note that the product of the forward and backward variables at a given s and t is the probability of all the paths corresponding to $\mathbf{l}$ that go through the symbol s at time t, i.e.,

$$\alpha_t(s)\beta_t(s) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l}) : \pi_t = \mathbf{l}'_s} y^t_{\mathbf{l}'_s} \prod_{t=1}^{T} y^t_{\pi_t} \tag{7}$$

Rearranging and substituting in from (4) gives,

$$\frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}} = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l}) : \pi_t = \mathbf{l}'_s} p(\pi|\mathbf{x}) \tag{8}$$

For any t, we can therefore sum over all s to get $p(\mathbf{l}|\mathbf{x})$:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}} \tag{9}$$

On the other hand, combining (3) and (4),

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} \prod_{t=1}^{T} y^t_{\pi_t} \tag{10}$$

Thus to differentiate this with respect to y^t_k , we need only consider those paths going through label k at time t (derivatives of other paths is zero). Noting that the same label (or blank) may be repeated several times for a single labelling $\mathbf{l}$, we define the set of positions where label k occurs as $lab(\mathbf{l}, k) = s : \mathbf{l}'_s = k$, which may be empty. We then differentiate (10) to get,

$$
\begin{aligned}
\frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y_k^t} &= \sum_{s \in lab(\mathbf{l},k)} \frac{\partial p(\pi|\mathbf{x})}{\partial y_k^t} \\
&= \sum_{s \in lab(\mathbf{l},k)} \frac{\partial \prod_{t'=1}^{T} y_{\pi_{t'}}^{t'}}{\partial y_k^t} \\
&= \sum_{s \in lab(\mathbf{l},k)} \frac{\partial y_k^t \prod_{t' \neq t} y_{\pi_{t'}}^{t'}}{\partial y_k^t} \\
&= \sum_{s \in lab(\mathbf{l},k)} \prod_{t' \neq t} y_{\pi_{t'}}^{t'} \\
&= \frac{1}{{y_k^t}^2} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s)
\end{aligned}
\tag{11}
$$

At this point, we can set $\mathbf{l} = \mathbf{z}$ and substituting into (2), then get the final gradient,

$$
\frac{\partial O^{ML}(\{(\mathbf{x}, \mathbf{z}\}, \mathcal{N}_w)}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{1}{{y_k^t}^2} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s) \tag{12}
$$

where, $p(\mathbf{z}|\mathbf{x})$ can be calculated from (9).

Next, we can give the gradient for the unnormalised output u_k^t. Recall that derivative of softmax function is,

$$
\frac{\partial y_{k'}^t}{\partial u_k^t} = y_{k'}^t \delta_{kk'} - y_{k'}^t y_k^t \tag{13}
$$

Then we get,

$$
\begin{aligned}
\frac{\partial O}{\partial u_k^t} &= \sum_{k'} \frac{\partial O}{\partial y_{k'}^t} \frac{\partial y_{k'}^t}{\partial u_k^t} \\
&= \sum_{k'} ((y_{k'}^t \delta_{kk'} - y_{k'}^t y_k^t)(-\frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{1}{{y_{k'}^t}^2} \sum_{s \in lab(\mathbf{l},k')} \alpha_t(s)\beta_t(s))) \\
&= \sum_{k'} (\frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{y_k^t}{y_{k'}^t} \sum_{s \in lab(\mathbf{l},k')} \alpha_t(s)\beta_t(s)) - \frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{1}{y_k^t} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s) \\
&= y_k^t - \frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{1}{y_k^t} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s)
\end{aligned}
\tag{14}
$$

we write the last step by noting that $\sum_{k'} \sum_{s \in lab(\mathbf{l},k')} (\cdot) \equiv \sum_{s=1}^{|\mathbf{l}'|} (\cdot)$, then, using (9), the $p(\mathbf{z}|\mathbf{x})$ is canceled out.

2.3 The CTC Forward-Backward Algorithm

The last thing we have to do is calculating the forward and backward variables. We now show that by define a recursive from, these variables can be calculated efficiently.

Given a labelling $\mathbf{l}$, we first extend it to $\mathbf{l}'$ with blanks added to the beginning and the end and inserted between every pair of labels. The length of $\mathbf{l}'$ is therefore $2|\mathbf{l}| + 1$. In calculating the probabilities of prefixes of $\mathbf{l}'$ we allow all transitions between blank and non-blank labels, and also those between any pair of distinct non-blank labels(because of the map $\mathcal{B}$, the repeated labels will be merged). We allow all prefixes to start with either a blank (b) or the first symbol in $\mathbf{l}$ ($\mathbf{l}_1$).

This gives us the following rules for initialisation

$$\begin{aligned} \alpha_1(1) &= y_b^1 \\ \alpha_1(2) &= y_{\mathbf{l}_1}^1 \\ \alpha_1(s) &= 0, \forall s > 2 \end{aligned}$$

and recursion

$$\alpha_t(s) = \begin{cases} y_{\mathbf{l}'_s}^t (\alpha_{t-1}(s) + \alpha_{t-1}(s-1)) & \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ y_{\mathbf{l}'_s}^t (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2)) & otherwise \end{cases} \tag{15}$$

Note that $\alpha_t(s) = 0, \forall s < |\mathbf{l}'| - 2(T-t) - 1$, because these variables correspond to states for which there are not enough time-steps left to complete the sequence.

Here we can get another method to calculate $p(\mathbf{l}|\mathbf{x})$, by adding up all forward variables at time T, i.e.,

$$p(\mathbf{l}|\mathbf{x}) = \alpha_T(|\mathbf{l}'|) + \alpha_T(|\mathbf{l}'| - 1) \tag{16}$$

Similarly, the backward variables can be initalisd as,

$$\begin{aligned} \beta_T(|\mathbf{l}'|) &= y_b^T \\ \beta_T(|\mathbf{l}'| - 1) &= y_{\mathbf{l}_{|\mathbf{l}|}}^T \\ \beta_T(s) &= 0, \forall s < |\mathbf{l}'| - 1 \end{aligned}$$

and recursion

$$\beta_t(s) = \begin{cases} y^t_{\mathbf{l}'_s}(\beta_{t+1}(s) + \beta_{t+1}(s+1)) & \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s+2} = \mathbf{l}'_s \\ y^t_{\mathbf{l}'_s}(\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2)) & otherwise \end{cases} \tag{17}$$

Note that $\beta_t(s) = 0, \forall s > 2t$.

Following figure (Figure 1) illustrate the forward backward algorithm applied to the labelling 'CAT'(from the paper).

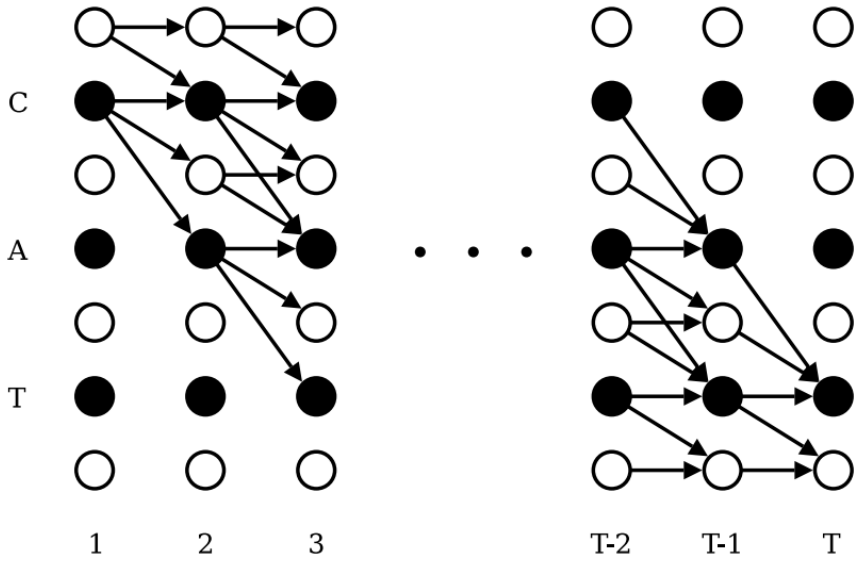

Figure 1: Alpha-Beta Algorithm

3 The Implementation

The `TranscriptionLayer` class inherits the `SoftmaxLayer` class(see this post). The `feed_forward()` and `feed_back()` methods are the general softmax function, so only need to implement the `calculate_errors()` method to calculate the $\frac{\partial O}{\partial y^t_k}$. In order to use (12) to get output error, first need to calculate the αs and βs. Forward variables are got using (15).

But backward variables are in another form, given in Graves' Dissertation. Consider backward variable started from time $t+1$,

$$\tilde{\beta}_t(s) = \sum_{\pi \in L^T : \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}'|}} \prod_{t'=t+1}^{T} y^{t'}_{\pi_{t'}} \tag{18}$$

Noting that, β and $\tilde{\beta}$ has a simple relationship:

$$\beta_t(s) = y^t_{\pi_t} \tilde{\beta}_t(s) \tag{19}$$

Thus, we can get recursion formula for $\tilde{\beta}$ by substituting (19) into (17),

$$\begin{aligned} \tilde{\beta}_T(|\mathbf{l}'|) &= 1 \\ \tilde{\beta}_T(|\mathbf{l}'| - 1) &= 1 \\ \tilde{\beta}_T(s) &= 0, \forall s < |\mathbf{l}'| - 1 \end{aligned}$$

$$\tilde{\beta}_t(s) = \begin{cases} y^{t+1}_{\mathbf{l}'_s} \tilde{\beta}_{t+1}(s) + y^{t+1}_{\mathbf{l}'_{s+1}} \tilde{\beta}_{t+1}(s+1) & \mathbf{l}'_s = b \,\text{or}\, \mathbf{l}'_{s+2} = \mathbf{l}'_s \\ y^{t+1}_{\mathbf{l}'_s} \tilde{\beta}_{t+1}(s) + y^{t+1}_{\mathbf{l}'_{s+1}} \tilde{\beta}_{t+1}(s+1) + y^{t+1}_{\mathbf{l}'_{s+2}} \tilde{\beta}_{t+1}(s+2) & otherwise \end{cases} \tag{20}$$

Noting that, if $\mathbf{l}'_s \neq blank$, then $\mathbf{l}'_{s+1}$ must be $blank$.

And the gradient for output (12) becomes,

$$\frac{\partial O^{ML}(\{(\mathbf{x}, \mathbf{z}\}, \mathcal{N}_w)}{\partial y^t_k} = -\frac{1}{p(\mathbf{z}|\mathbf{x})} \frac{1}{y^t_k} \sum_{s \in lab(\mathbf{l}, k)} \alpha_t(s) \tilde{\beta}_t(s) \tag{21}$$

where,

$$p(\mathbf{z}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{z}'|} \alpha_t(s) \tilde{\beta}_t(s) \tag{22}$$

Actually, the RNNLIB code computes $p(\mathbf{z}|\mathbf{x})$ using (16).

To wrap up, CTC using a forward-backward algorithm to efficiently compute the RNN output errors, corresponding to a new ML objective function. With these errors, we can use any traditional gradient methods to train the network.

4 Decoding

Once the network is trained, we would use it to transcribe some unknown input sequence $\mathbf{x}$. *Decoding* is referred to the task of finding the best labelling $\mathbf{l}^*$,

$$\mathbf{l}^* = \underset{\mathbf{l}}{\operatorname{argmax}}\ p(\mathbf{l}|\mathbf{x}) \tag{23}$$

There are two approximate algorithms.

4.1 Best Path Decoding

This method assumes that the most probable path corresponding to the most probable labelling,

$$\mathbf{l}^* \approx \mathcal{B}(\pi^*) \tag{24}$$

where $\pi^* = \arg\,!\max_{\pi}\ p(\pi|\mathbf{x})$.

This is trivial to compute, simply by concatenating the most active outputs at every time step. But it can lead to errors, because that the map $\mathcal{B}$ is a many-to-one map.

4.2 Prefix Search Decoding

By modifying the forward variables, this method can efficiently calculate the probabilities of successive extensions of labelling prefixes.

Prefix search decoding is a best-first search through the tree of labellings, where the children of a given labelling are those that share it as a prefix. At each step the search extends the labelling whose children have the largest cumulative probability (see below figure(Figure 2)).

Each node either ends (e) or extends the prefix at its parent node. The number above an extending node is the total probability of all labellings beginning with that prefix. The number above an end node is the probability of the single labelling ending at its parent. At every iteration the extensions of the most probable remaining prefix are explored. Search ends when a single labelling (here XY) is more probable than any remaining prefix.

To extend the tree, we need to compute extended path probability, which can be computed in a recursive way. Let $\gamma_t(\mathbf{p}_n)$ be the probability of the network outputting prefix $\mathbf{p}$ by time t such that a non-blank label is output at t. Similarly, let $\gamma_t(\mathbf{p}_b)$ be the probability of the network outputting prefix $\mathbf{p}$ by time t such that the blank label is output at t. i.e.

$$\gamma_t(\mathbf{p}_n) = p(\pi_{1:t} : \mathcal{B}(\pi_{1:t}) = \mathbf{p}, \pi_t = \mathbf{p}_{|\mathbf{p}|} \mid \mathbf{x}) \tag{25}$$

$$\gamma_t(\mathbf{p}_b) = p(\pi_{1:t} : \mathcal{B}(\pi_{1:t}) = \mathbf{p}, \pi_t = blank \mid \mathbf{x}) \tag{26}$$

Then for a length T input sequence $\mathbf{x}$, $p(\mathbf{p}|\mathbf{x}) = \gamma_T(\mathbf{p}_n) + \gamma_T(\mathbf{p}_b)$. Also let $p(\mathbf{p}\ldots|\mathbf{x})$ be the cumulative probability of all labelling not equal to $\mathbf{p}$ of which

$$\mathbf{p}$$

is a prefix

$$p(\mathbf{p}\ldots \mid \mathbf{x}) = \sum_{\mathbf{l}\neq\emptyset} p(\mathbf{p}+\mathbf{l} \mid \mathbf{x}) \tag{27}$$

where $\emptyset$ is the empty sequence. $p(\mathbf{p}\ldots \mid \mathbf{x})$ is the value for extending node in the prefix tree, and $p(\mathbf{p} \mid \mathbf{x})$ is the value for end node.

In fact, by definition, relation between γ and α is,

$$\gamma_t(\mathbf{p}_n) = \alpha_t(2|\mathbf{p}|) \tag{28}$$

$$\gamma_t(\mathbf{p}_b) = \alpha_t(2|\mathbf{p}| + 1) \tag{29}$$

Using (15), we get the recursion for $\gamma_t(\mathbf{p}_n)$ given $\gamma_{t-1}(\mathbf{p}_n)$, extending $\mathbf{p}^*$ to $\mathbf{p} = \mathbf{p}^* + k$ with label $k \in L$,

$$\gamma_1(\mathbf{p}_n) = \begin{cases} y_k^1 & \mathbf{p}^* = \emptyset \\ 0 & \text{otherwise} \end{cases}$$
$$\gamma_1(\mathbf{p}_b) = 0$$

$$\gamma_t(\mathbf{p}_n) = \begin{cases} y_k^t(\gamma_{t-1}(\mathbf{p}_b^*) + \gamma_{t-1}(\mathbf{p}_n)) & \mathbf{p}^* \text{ ends in } k \\ y_k^t(\gamma_{t-1}(\mathbf{p}_b^*) + \gamma_{t-1}(\mathbf{p}_n^*) + \gamma_{t-1}(\mathbf{p}_n)) & otherwise \end{cases} \tag{30}$$

$$\gamma_t(\mathbf{p}_b) = y_b^t(\gamma_{t-1}(\mathbf{p}_b) + \gamma_{t-1}(\mathbf{p}_n)) \tag{31}$$

And calculating the path probabilities,

$$p(\mathbf{p} \mid \mathbf{x}) = \gamma_T(\mathbf{p}_b) + \gamma_T(\mathbf{p}_n) \tag{32}$$

$$p(\mathbf{p}\ldots \mid \mathbf{x}) = \gamma_1(\mathbf{p}_n) + \sum_{t=2}^{T} (\gamma_t(\mathbf{p}_n) - y_k^t \gamma_{t-1}(\mathbf{p}_n)) - p(\mathbf{p} \mid \mathbf{x}) \tag{33}$$

The extension procedure start from $\mathbf{p}^* = \emptyset$, with initialisation,

$$\begin{aligned} 1 \leq t \leq T \begin{cases} \gamma_t(\emptyset_n) & = 0 \\ \gamma_t(\emptyset_b) & = \prod_{t'=1}^{t} y_b^{t'} \end{cases} \\ p(\emptyset \mid \mathbf{x}) &= \gamma_T(\emptyset_b) \\ p(\emptyset \ldots \mid \mathbf{x}) &= 1 - p(\emptyset \mid \mathbf{x}) \end{aligned}$$

and iterate util $\max_p p(\mathbf{p}\ldots \mid \mathbf{x}) < \max_{p'} p(\mathbf{p}' \mid \mathbf{x})$.

Given enough time, prefix search decoding always finds the most probable labelling. However, the maximum number of prefixes it must expand grows exponentially with the input sequence length. We need further heuristic.

Observing that the outputs of a trained CTC network tend to form a series of spikes separated by strongly predicted blanks, we can divide the output sequence into sections that are very likely to begin and end with a blank. We can do this by choosing boundary points where the probability of observing a blank label is above a certain threshold, then apply the above algorithm to each section individually and concatenate these to get the final transcription.

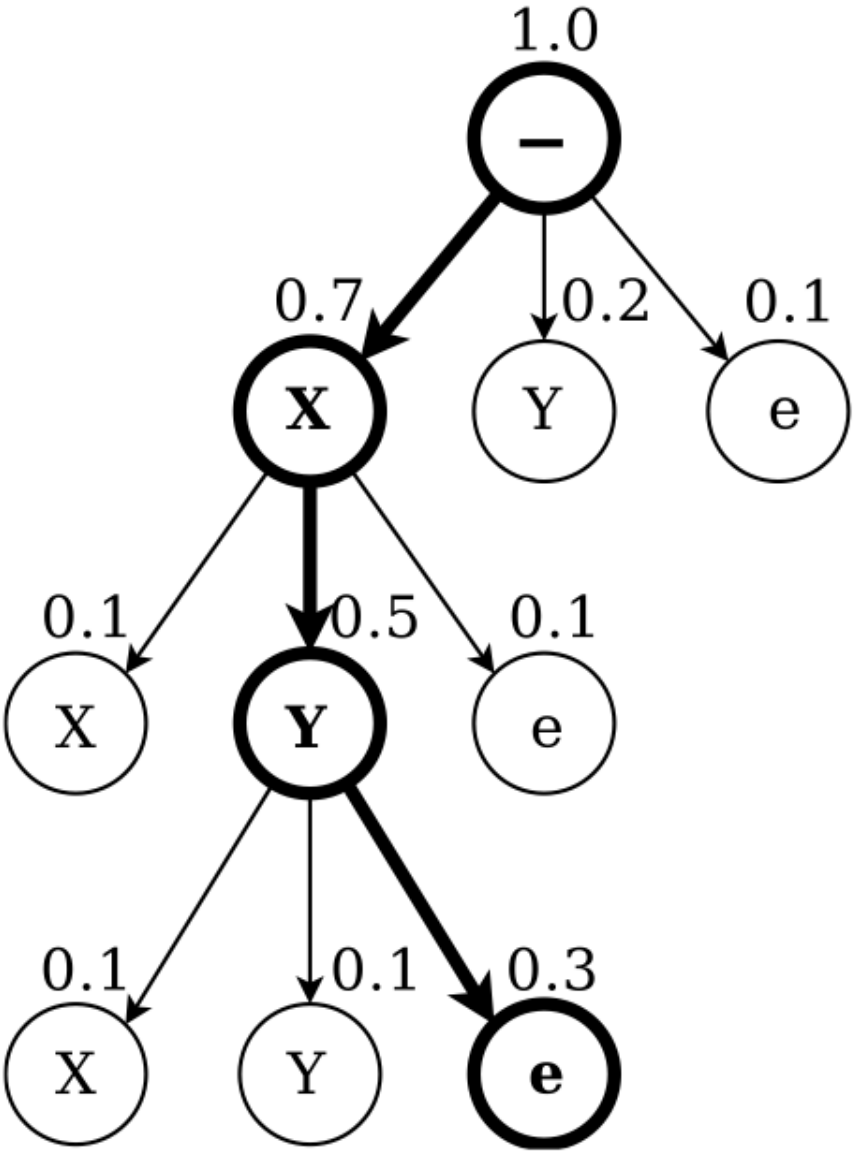

Figure 2: Prefix Search Decoding