Notes on Sequence Transduction with Recurrent Neural Networks

Mingkun Huang

May 17, 2018

1 Recurrent Neural Network Transducer

Let $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ be a length T *input sequence* of arbitrary length belonging to the set $\mathcal{X}^*$ of all sequences over some input space $\mathcal{X}$. Let $\mathbf{y} = (y_1, y_2, \ldots, y_U)$ be a length U *output sequence* belonging to the set $\mathcal{Y}^*$ of all sequences over some output space $\mathcal{Y}$. Both the inputs vectors x_t and the output vectors y_u are represented by fixed-length real-valued vectors.

Define the extended output space $\bar{\mathcal{Y}}$ as $\mathcal{Y} \cup \varnothing$, where $\varnothing$ denotes the *null* output. We refer to the elements $\mathbf{a} \in \bar{\mathcal{Y}}^*$ as *alignments*, since the location of the null symbols determines an alignment between the input and output sequences. Given $\mathbf{x}$, the RNN transducer defines a conditional distribution $\Pr\big(\mathbf{a} \in \bar{\mathcal{Y}}^* | \mathbf{x}\big)$. This distribution is then collapsed onto the following distribution over $\mathcal{Y}^*$

$$\Pr(\mathbf{y} \in \mathcal{Y}^* | \mathbf{x}) = \sum_{\mathbf{a} \in \mathcal{B}^{-1}(\mathbf{y})} \Pr(\mathbf{a} | \mathbf{x}) \tag{1}$$

where $\mathcal{B} : \bar{\mathcal{Y}}^* \to \mathcal{Y}^*$ is a function that removes the null symbols from the alignments in $\bar{\mathcal{Y}}^*$.

Two recurrent neural networks are used to determine $\Pr\big(\mathbf{a} \in \bar{\mathcal{Y}}^* | \mathbf{x}\big)$. One network, referred to as the *transcription network* $\mathcal{F}$, scans the input sequence $\mathbf{x}$ and outputs the sequence $\mathbf{f} = (f_1, \ldots, f_T)$ of transcription vectors[1]. The other network, referred to as the *prediction network* $\mathcal{G}$, scans the output sequence $\mathbf{y}$ and outputs the prediction vector sequence $\mathbf{g} = (g_0, g_1, \ldots, g_U)$.

1.1 Prediction Network

The prediction network $\mathcal{G}$ is a recurrent neural network consisting of an input layer, an output layer and a single hidden layer. The length $U + 1$ input sequence $\mathbf{y}^* = (\varnothing, y_1, \ldots, y_U)$ to $\mathcal{G}$ output sequence $\mathbf{y}$ with $\varnothing$ prepended. If $\mathcal{Y}$ consists of K labels, the input layer is therefore size K. $\varnothing$ is encoded as a length K vector of zeros. The output layer is size $K + 1$ (one unit for each element of $\bar{\mathcal{Y}}$) and hence the prediction vectors g_u are also size $K + 1$.

The prediction network attempts to model each element of $\mathbf{y}$ given the previous ones; it is therefore similar to a standard next-step-prediction RNN, only with the added option of makeing *null* predictions. Since the input and output lengths are equal, we can apply attention onto it with the output of transcription network.

1.2 Transcription Network

The transcription network $\mathcal{F}$ is an arbitrary representation learning network. Given a length T input sequence $(x_1, \ldots, x_T)$, it outputs the same length hidden vectors $(f_1, \ldots, f_T)$ with each vector size $K + 1$.

The transcription network is similar to a CTC RNN, which also uses a null output to define a distribution over input-output alignments.

[1] For simplicity we assume the transcription sequence to be the same length as the input sequence; however this may not be true, for example if the transcription network uses a pooling architecture to reduce the sequence length.

1.3 Output Distribution

Given the transcription vector f_t, where $1 \leq t \leq T$, the prediction vector g_u, where $0 \leq u \leq U$, and label $k \in \bar{\mathcal{Y}}$, define the *log output density function*

$$h(k,t,u) = f_t^k + g_u^k \tag{2}$$

where superscript k denotes the k^{th} element of the vectors. The density can be softmaxed to yield the conditional *output distribution*

$$\Pr\big(k \in \bar{\mathcal{Y}}|t,u\big) = \frac{e^{h(k,t,u)}}{\sum_{k' \in \bar{\mathcal{Y}}} e^{h(k',t,u)}} \tag{3}$$

To simplify notation, define

$$\begin{aligned} y(t,u) &:= \Pr(y_{u+1}|t,u) \\ \varnothing(t,u) &:= \Pr(\varnothing|t,u) \end{aligned} \tag{4}$$

$\Pr(k|t,u)$ is used to determine the transition probabilities in the lattice shown in [1]. For any specific node, there are at most two transitions.

1.4 Forward-Backward Algorithm

Define the *forward variable* $\alpha(t,u)$ as the probability of outputting $\mathbf{y}_{[1:u]}$ during $\mathbf{f}_{[1:t]}$. The forward variables for all $1 \leq t \leq T$ and $0 \leq u \leq U$ can be calculated recursively using

$$\alpha(t,u) = \alpha(t-1,u) \cdot \varnothing(t-1,u) + \alpha(t,u-1) \cdot y(t,u-1) \tag{5}$$

with initial condition $\alpha(1,0) = 1$. The total output sequence probability is equal to the forward variable at the terminal node:

$$\Pr(\mathbf{y}|\mathbf{x}) = \alpha(T,U) \cdot \varnothing(T,U) \tag{6}$$

Define the *backward variable* $\beta(t,u)$ as the probability of outputting $\mathbf{y}_{[u+1:U]}$ during $\mathbf{f}_{[t:T]}$. Then

$$\beta(t,u) = \beta(t+1,u) \cdot \varnothing(t,u) + \beta(t,u+1) \cdot y(t,u) \tag{7}$$

with initial condition $\beta(T,U) = \varnothing(T,U)$. From the definition of the forward and backward variables it follows that their product $\alpha(t,u) \cdot \beta(t,u)$ at any point (t,u) in the output lattice is equal to the probability of emitting the complete output sequence *if y_u is emitted during transcription step t.*

1.5 Training

Given an input sequence $\mathbf{x}$ and a *target sequence* $\mathbf{y}^*$, the natural way to train the model is to minimize the negtive log likelihood $\mathcal{L} = -\ln \Pr(\mathbf{y}^*|\mathbf{x})$ of the target sequence. Analysing the diffusion of probability through the output lattice shows that $\Pr(\mathbf{y}^*|\mathbf{x})$ is equal to the sum of $\alpha(t,u) \cdot \beta(t,u)$ over any top-left to bottom-right diagonal through the nodes. That is, $\forall n : 1 \leq n \leq U + T$

$$\Pr(\mathbf{y}^*|\mathbf{x}) = \sum_{(t,u):t+u=n} \alpha(t,u) \cdot \beta(t,u) \tag{8}$$

From Eqs. (5), (7) and (8) and the definition of $\mathcal{L}$ it follows that

$$\frac{\partial \mathcal{L}}{\partial \Pr(k|t,u)} = -\frac{\alpha(t,u)}{\Pr(\mathbf{y}^*|\mathbf{x})} \begin{cases} \beta(t,u+1) & \text{if } k = y_{u+1} \\ \beta(t+1,u) & \text{if } k = \varnothing \\ 1 & \text{if } k = \varnothing, t = T, u = U \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

And therefore

$$\frac{\partial \mathcal{L}}{\partial h(k,t,u)} = \sum_{k' \in \bar{\mathcal{Y}}} \frac{\partial \mathcal{L}}{\Pr(k'|t,u)} \frac{\partial \Pr(k'|t,u)}{\partial h(k,t,u)} \tag{10}$$

where, from Eq. (3)

$$\frac{\partial \Pr(k'|t,u)}{\partial h(k,t,u)} = \Pr(k'|t,u)[\delta_{k,k'} - \Pr(k|t,u)] \tag{11}$$

Then we have

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial h(k,t,u)} &= -\frac{\alpha(t,u)}{\Pr(\mathbf{y}^*|\mathbf{x})} \left\{ \beta(t,u+1) \cdot y(t,u) \cdot [\delta_{k,y_{u+1}} - \Pr(k|t,u)] + \beta(t+1,u) \cdot \varnothing(t,u) \cdot [\delta_{k,\varnothing} - \Pr(k|t,u)] \right\} \\
&= -\frac{\alpha(t,u)}{\Pr(\mathbf{y}^*|\mathbf{x})} \begin{cases}
\beta(t,u+1) \cdot y(t,u) \cdot [1 - y(t,u)] + \beta(t+1,u) \cdot \varnothing(t,u) \cdot (-y(t,u)), & k = y_{u+1} \\
\beta(t,u+1) \cdot y(t,u) \cdot (-\varnothing(t,u)) + \beta(t+1,u) \cdot \varnothing(t,u) \cdot (1 - \varnothing(t,u)), & k = \varnothing \\
\varnothing(t,u) \cdot (1 - \beta(t,u)), & k = \varnothing, t = T, u = U \\
\beta(t,u+1) \cdot y(t,u) \cdot (-\Pr(k|t,u)) + \beta(t+1,u) \cdot \varnothing(t,u) \cdot (-\Pr(k|t,u)), & \text{otherwise}
\end{cases} \\
&= -\frac{\alpha(t,u)}{\Pr(\mathbf{y}^*|\mathbf{x})} \begin{cases}
\beta(t,u+1) \cdot y(t,u) - \beta(t,u) \cdot y(t,u), & k = y_{u+1}, u < U \\
\beta(t+1,u) \cdot \varnothing(t,u) - \beta(t,u) \cdot \varnothing(t,u), & k = \varnothing, t < T \\
\varnothing(t,u) - \beta(t,u) \cdot \varnothing(t,u), & k = \varnothing, t = T, u = U \\
-\beta(t,u) \cdot \Pr(k|t,u), & \text{otherwise}
\end{cases} \\
&= \frac{\alpha(t,u) \cdot \Pr(k|t,u)}{\Pr(\mathbf{y}^*|\mathbf{x})} \cdot \beta(t,u) - \frac{\alpha(t,u) \cdot \Pr(k|t,u)}{\Pr(\mathbf{y}^*|\mathbf{x})} \begin{cases}
\beta(t,u+1), & k = y_{u+1}, u < U \\
\beta(t+1,u), & k = \varnothing, t < T \\
1, & k = \varnothing, t = T, u = U \\
0, & \text{otherwise}
\end{cases}
\end{aligned} \tag{12}$$

From Eq. (2)

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial f_t^k} &= \sum_{u=0}^{U} \frac{\partial \mathcal{L}}{\partial h(k,t,u)} \\
\frac{\partial \mathcal{L}}{\partial g_u^k} &= \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial h(k,t,u)}
\end{aligned} \tag{13}$$

There are two ways to calculate the gradient:

- Gradient to softmax output (Eq. 9).
  Easy to program. But consumes too much memory.
- Gradient to activations (Eq. 12).
  Memory efficient, can be accelerated in GPU.

Note that in practical programming, all probabilities are in log scale. Since the calculation may overflow float point, we calculate the two terms in Eq. (12) seperately.

References

[1] Alex Graves. Sequence Transduction with Recurrent Neural Network, arXiv preprint arXiv:1211.3711, 2012.