

# MT Interim Report: Neural Word Alignemnt

BAILEY PARKER

Johns Hopkins University  
bailey@jhu.edu

VIVIAN TSAI

Johns Hopkins University  
viv@jhu.edu

WILLIAM WATSON

Johns Hopkins University  
billwatson@jhu.edu

## Abstract

*We will discuss our intial progress made on our Neural Word Alignemnt. More specifically, we will discuss our data procurement methods and processing algorithms. In addition, we elaborate on our current experimental models, including the Dot Aligner and Bidirectional GRU Aligner. Our loss function, train/eval implmentation, and batching techniques are also discussed.*

## 1 Introduction

## 2 Data Procurement and Processing

### 2.1 Symbol Tokenization

### 2.2 Number Tokenization

### 2.3 Proper Noun Tokenization

### 2.4 Lemmatization Techniques

### 2.5 POS Tagging

## 3 Model Components

### 3.1 Preliminary Notation

We begin by stating several operations frequently used in our discussion.

#### 3.1.1 Hadamard Product

To perform element-wise multiplication of two matrices  $A$  and  $B$ , of equivalent dimensions, we use the hadamard product, defined as the  $\circ$ .

$$(A \circ B)_{ij} = A_{ij} \cdot B_{ij} \quad (1)$$

#### 3.1.2 Sigmoid Function

The sigmoid function  $\sigma$  is applied element wise and defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

#### 3.1.3 Hyperbolic Tangent Function

The hyperbolic tangent function  $\tanh$  is defined as follows:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3)$$

and is applied element-wise.

#### 3.1.4 Variables

We define our source sequence as  $s$ , target sequence as  $t$ , and refer to the  $i$ -th source token as  $s_i$ . We refer to the  $j$ -th target token as  $t_j$ . We refer to a matrix  $\psi$  whose rows represent values related to source tokens  $s_i$  and whose columns refer to values realted to target tokens  $t_j$ . Hence,  $\psi$  is an  $|s| \times |t|$  sized matrix.

## 3.2 Softmax and Log Softmax

The softmax function transforms a vector of values into a probability distirbution. Applying the softmax function to an n-dimensional input tensor rescales it so that the elements of the n-dimensional output tensor lie in the range (0,1) and sum to 1.

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (4)$$

For a matrix  $\psi$ , the rows correspond to source words  $s_i$  and the columns correspond to target words  $t_j$ . In addition, when we softmax with respect to the targets,

i.e.  $\sigma_t(\psi)$ , the softmax is applied per column. If applied per row, we denote this as  $\sigma_s(\psi)$ .

$$\sigma_t(\psi)_{ij} = \frac{e^{\psi_{ij}}}{\sum_k e^{\psi_{kj}}} \quad (5)$$

$$\sigma_s(\psi)_{ij} = \frac{e^{\psi_{ij}}}{\sum_k e^{\psi_{ik}}} \quad (6)$$

However, it is sometimes better to work in log-space, and use the log softmax operator for numerical stability.

$$\log \sigma_t(\psi)_{ij} = \log \frac{e^{\psi_{ij}}}{\sum_k e^{\psi_{kj}}} \quad (7)$$

$$\log \sigma_s(\psi)_{ij} = \log \frac{e^{\psi_{ij}}}{\sum_k e^{\psi_{ik}}} \quad (8)$$

### 3.3 Word Embeddings

Word Embedding layers allow for a simple lookup table that stores embeddings of a fixed dictionary and size. More specifically, these layers are often used to store word embeddings and retrieve them using indices. The input to the embedding layer is a list of indices, and the output is the corresponding word embeddings. This allows words to be represented numerically to a set embedding dimension size, and thus can be passed onto later layers.

Word Embeddings can be formulated as a weight matrix  $W_e$ , where the vector representation of word  $w_i$  is the  $i$ -th row of the matrix, and can be represented as  $W_e[w_i]$ .

$$W_e = \begin{bmatrix} \leftarrow w_1 \rightarrow \\ \vdots \\ \leftarrow w_i \rightarrow \\ \vdots \\ \leftarrow w_n \rightarrow \end{bmatrix} \quad (9)$$

### 3.4 Gated Recurrent Units (GRU)

Gated Recurrent Units are a more complex formulation to recurrent layers to process sequences, and improve the vanishing gradient problem found in vanilla RNN layers while using less parameters than a Long Short-Term Memory (LSTM) layer[1]. A GRU makes use of 3 gates:  $r_t$  reset gate;  $z_t$  update gate;  $n_t$  new gate. These 3 gates allow for a blending of the hidden state with

new input, and are computed as follows for each input  $x_t$  in a sequence  $\mathbf{x}$ :

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t \circ (W_{hn}h_{t-1} + b_{hn})) \\ h_t &= (1 - z_t) \circ n_t + z_t \circ h_{t-1} \end{aligned} \quad (10)$$

where  $x_t$  is input at time  $t$ ,  $h_{t-1}$  is the hidden state of the previous layer at time  $t - 1$  or the initial hidden state at time 0,  $h_t$  is the new hidden state at time  $t$ , and  $\sigma$  is the sigmoid function.

### 3.5 Bidirectional GRU

GRUs only process sequences in a forward direction. However, for translation and in general NLP, we also care about the succeeding input. We thus introduce the Bidirectional GRU, which that runs two separate GRU layers in opposite directions and stacks the output s.t. each element has the context of elements before and after it.

Let us define  $\text{GRU}(\text{input})$  as a GRU layer that outputs the hidden cells from *input* on a single forward pass. Let us also define the operator  $\vec{h}$  as the tensor with elements ordered in order  $[0, n]$ , and  $\overleftarrow{h}$  as the tensor elements ordered from  $[n, 0]$ , i.e. reversed.

When flipping the arrow, i.e., from  $\vec{h}$  to  $\overleftarrow{h}$ , we define this operation as an invert/reverse function, and it flips the elements to the opposite orientation. We additionally declare  $\parallel$  as the concatenation operator. Finally, we define  $\vec{h}_f$  as the forward output,  $\overleftarrow{h}_b$  as the backward output, and  $h_o$  as the final, stacked bidirectional output.

$$\begin{aligned} \vec{h}_f &= \text{GRU}(\overrightarrow{\text{input}}) \\ \overleftarrow{h}_b &= \text{GRU}(\overleftarrow{\text{input}}) \\ h_o &= \vec{h}_f \parallel \overleftarrow{h}_b \end{aligned} \quad (11)$$

From the above equations, we compute the forward output normally; compute the backwards output on the inverted input; and concatenate the forward outputs with the inverted backwards outputs. All final hidden states are provided as well. The output is of size  $(N, \text{batch}, \text{hidden size} * 2)$  since we concatenate the outputs of the two GRU layers.

### 3.6 Alignment Prior

In order to learn diagonal alignemnts, we describe our formulation of the IBM Model 2 reparameterization as a custom PyTorch layer. We define the prior alignment matrix  $A$  as an  $|s| \times |t|$ . Given a source token  $s_i$  and target token  $t_j$ , and alignment hyperparameter  $\lambda$ , we define the alignment distortion function  $a_{ij}$ :

$$a_{ij} = -\lambda|i - j| \quad (12)$$

We set the default  $\lambda$  value to 4. This encodes our diagonal prior, and allows us to add in the diagonal weights to our network’s weights. We allow for an optional, learnable scaling factor  $\alpha$  to control the strength of this prior, defaulted to 0.25. Hence our IBM Model 2 module outputs the alignemnt prior matrix  $A$ :

$$A_{ij} = \alpha \cdot a_{ij} \quad (13)$$

### 3.7 Batch Matrix Multiplication (BMM)

To combine source and target tensors, we use Batch Matrix Multiplication. Given a matrix  $A$  of size  $(b, n, m)$  and matrix  $B$  of size  $(b, m, p)$ , we perform matrix multiplication on the sub-matrices to create an output matrix  $O$  of size  $(b, n, p)$ .

$$O_i = A_i \times B_i \quad (14)$$

This allows us to combine the vectorized output of our network on each individual sequence and combine them into a matrix tensor of size  $(b, |s|, |t|)$ . This requires the use of simple transpose functions to align the dimensions of each pairing sentence to correctly compute the BMM.

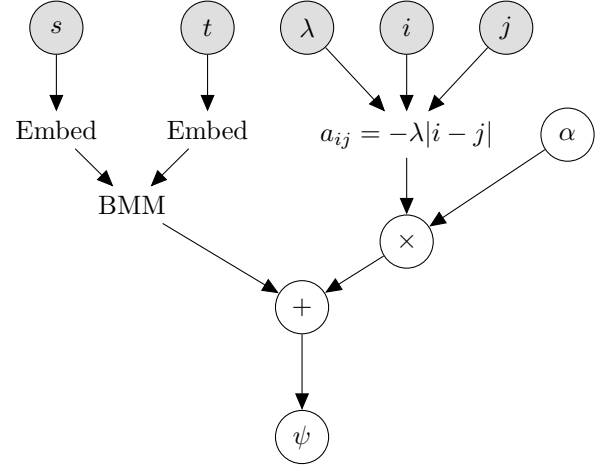
## 4 Model Descriptions

We use the aforementioned model components to construct complex alignment models, and provide a more through discussion on our techiques.

### 4.1 Dot Aligner

This is our baseline model. This model attempts to learn word embeddings to correctly give alignemnt outputs.

The intuition behind this baseline mdoel was to compute dot products between every source token  $s_i$  embedding and target token  $t_j$  embedding to generate an



**Figure 1:** Model Architecture for Dot Aligner for source  $s$ , target  $t$  alignments.  $\alpha$  is a global learnable scaling factor for the importance of alignment distribution  $a$ , and  $\lambda$  is the global alignment distortion parameter.

alignment matrix via BMM (Section 3.7), combined with the diagonal prior discussed in Section 3.6. The output is the alignment matrix  $\psi$ .

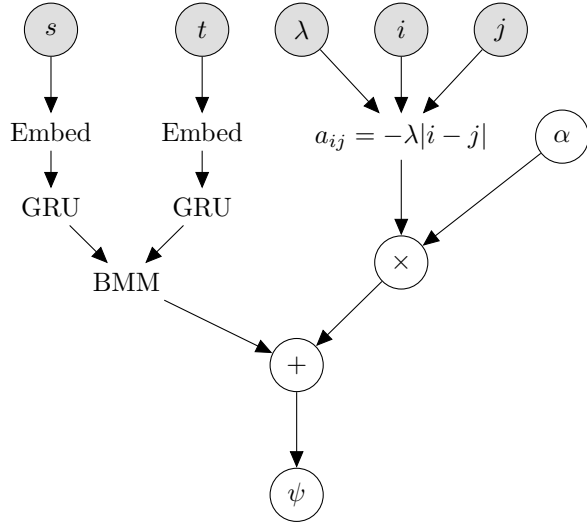
This is a very simple model that only learns the pure embeddings, and therefore we do not expect it to do as well as more complex models. The full computation graph can be seen in Figure 1.

### 4.2 Bidirectional GRU Aligner

An extension to the capacity of the Dot Aligner is to add a bidirectional GRU after the embedding layers. This will allow the model to consider not only forward propagation of the input sequence, but also the reverse. Therefore, our vectorized encodings of each sequence has the context of the words in its local area for context. The downstream algorithm for forward propagation is the same after the recurrent layers, as detailed in Figure 2.

### 4.3 Extensions

Here we discuss possible extensions to the two aforementioned models, to improve the capacity to learn alignments based upon our initial results.



**Figure 2:** Model Architecture for Bidirectional GRU Aligner for source  $s$ , target  $t$  alignments.  $\alpha$  is a global learnable scaling factor for the importance of alignment distribution  $a$ , and  $\lambda$  is the global alignment distortion parameter.

## 5 Loss Function

### 5.1 Supervised Loss

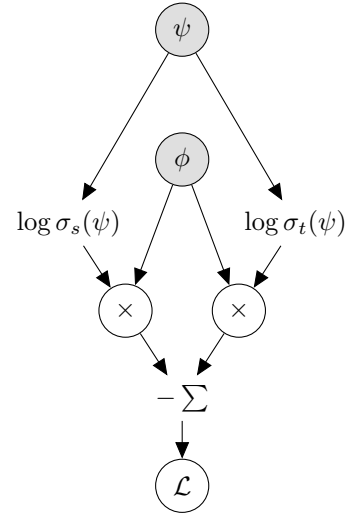
We can formulate our supervised loss as maximizing the probabilities of our alignments,  $\psi$ , with the ground truth alignments, represented as  $\phi$ .

Our targets are binary valued alignment matrices of size  $|s| \times |t|$ . Our targets  $\phi$  are defined as follows for a source token  $s_i$  and target token  $t_j$ :

$$\phi_{ij} = \begin{cases} 1 & \text{if } s_i \text{ aligns with } t_j \\ 0 & \text{else} \end{cases} \quad (15)$$

Hence, if a source token  $s_i$  aligns with  $t_j$ , our target is 1. We now formulate our loss objective as the maximum likelihood estimate (MLE) between our alignment matrix  $\psi$  and target matrix  $\phi$ . However,  $\psi$  are hard weights, and we use the softmax function to convert our alignment weights to valid probabilities. In addition, we seek to maximize the probability for both the row ( $\sigma_s$ ) and column ( $\sigma_t$ ) softmax. Hence our MLE function to be maximized is:

$$MLE(\psi, \phi) = \prod_i \prod_j \left[ \frac{\exp \psi_{ij}}{\sum_k \exp \psi_{ik}} \right]^{\phi_{ij}} \cdot \left[ \frac{\exp \psi_{ij}}{\sum_k \exp \psi_{kj}} \right]^{\phi_{ij}} \quad (16)$$



**Figure 3:** Computation graph for our supervised loss function. The generated alignment matrix  $\psi$  is compared to the ground truth alignment matrix  $\phi$ .

$$MLE(\psi, \phi) = \prod_i \prod_j [\sigma_s(\psi)_{ij}]^{\phi_{ij}} \cdot [\sigma_t(\psi)_{ij}]^{\phi_{ij}} \quad (17)$$

We can see that by maximizing the MLE function allows our network to drive the probabilities of alignment to be as close to 1 as possible, and implicitly depresses all other values along the row and column. However, neural networks do not maximize objective functions, and the multiplication of small probabilities leads to issues of numerical stability. We therefore define the Negative Log Likelihood function:

$$NLL(\psi, \phi) = - \sum_i \sum_j \phi_{ij} \cdot \log \sigma_s(\psi)_{ij} + \phi_{ij} \cdot \log \sigma_t(\psi)_{ij} \quad (18)$$

We sum up the log alignment probabilities corresponding to our target alignments, and this is minimized when the probability of alignment is high, and hence we can learn target alignments between source sentence  $s$  and target sentence  $t$ .

### 5.2 Unsupervised Alignment Loss

Our unsupervised loss function, to be minimized, is a 5-term equation. We define:

$a_t$  as the target prior alignment matrix normalized per column with respect to  $t$

$a_s$  as the source prior alignment matrix normalized per row with respect to  $s$

$\sigma_s$  as the softmax operator applied on the rows of a matrix

$\sigma_t$  as the softmax operator applied to each column of a matrix

Indexing is done under the assumption that source words  $s_i$  form rows and target words  $t_j$  form columns. The term  $ij$  is a shorthand for  $s_i$  and  $t_j$  indexing into our matrices.

$$\begin{aligned}
Loss = & \\
& - \sum_j^{|t|} \log \left[ \sum_i^{|s|} \exp(\log \sigma_s(\theta(t, s))_{ij} + \log \sigma_t(\psi)_{ij}) \right] \\
& + \sum_i^n \sum_j^m \sigma_t(\psi)_{ij} \cdot \log \left[ \frac{\sigma_t(\psi)_{ij}}{a_t(i, j)} \right] \\
& - \sum_i^{|s|} \log \left[ \sum_j^{|t|} \exp(\log \sigma_t(\theta(t, s))_{ij} + \log \sigma_s(\psi)_{ij}) \right] \\
& + \sum_i^n \sum_j^m \sigma_s(\psi)_{ij} \cdot \log \left[ \frac{\sigma_s(\psi)_{ij}}{a_s(i, j)} \right] \\
& - \log \sum_i^{|s|} \sum_j^{|t|} [\sigma_s(\psi) \circ \sigma_t(\psi)]_{ij}
\end{aligned} \tag{19}$$

The full derivation of the loss function and its component terms can be found in Appendix A.

## 6 Train/Eval Implmentation

### 6.1 Vocabulary Building

### 6.2 Batching

### 6.3 Optimizer

### 6.4 Additonal Items

## 7 Ground Truth Alignment Results

## 8 Current Status

## 9 Future Work

## References

- [1] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.
- [2] C. Dyer, V. Chahuneau, and N. A. Smith. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, 2013.

## A Loss Function

To create our loss function, we convert our alignment matrix  $\psi$  to a probability distribution. We define  $\sigma(\mathbf{x})$  as the softmax operator applied on vector  $\mathbf{x}$ , and  $\sigma(\mathbf{x})_i$  as the  $x_i$  softmax probability for vector  $\mathbf{x}$ .

$$\sigma(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (20)$$

We therefore define two operations on the alignment matrix  $\psi$ . For source  $s$  to target  $t$  probability generations, we define  $\sigma_t(\psi)$  as the softmax on each column, i.e., target word  $t_j$ .

$$\sigma_t(\psi) = [\sigma(\psi_{t_1}) \quad \dots \quad \sigma(\psi_{t_j}) \quad \dots \quad \sigma(\psi_{t_m})] \quad (21)$$

Equivalently,

$$\sigma_t(\psi)_{ij} = \frac{e^{\psi_{ij}}}{\sum_k e^{\psi_{kj}}} \quad (22)$$

In addition, we further define the target-to-source generation as  $\sigma_s(\psi)$ , where the softmax operator is applied on each row for target word  $s_i$  and produces a row vector.

$$\sigma_s(\psi) = \begin{bmatrix} \sigma(\psi_{s_1}) \\ \vdots \\ \sigma(\psi_{s_i}) \\ \vdots \\ \sigma(\psi_{s_n}) \end{bmatrix} \quad (23)$$

Equivalently,

$$\sigma_s(\psi)_{ij} = \frac{e^{\psi_{ij}}}{\sum_k e^{\psi_{ik}}} \quad (24)$$

While VAEs sample from the probability distribution derived by the neural network, our model's distribution is discrete. Therefore, we need not sample and can instead compute the loss in terms of all possibilities.

### A.1 Probability Maximization

In our original formulation, we maximized probabilities for word alignments. For this model, we define  $p(t, s|\theta, \psi)$  as such:

$$p(t, s|\theta, \psi) = \prod_j \sum_i^{s_i} p_\theta(t_j|s_i) \cdot p_a(i|j) = \prod_j \sum_i^{s_i} \sigma_s(\theta(t_j, s_i)) \cdot \sigma_t(\psi)_{ij} \quad (25)$$

This formula stems from our assumption that the alignments for each target word are conditionally independent, hence a product over target words  $t_j$ . We then sum over all the alignment possibilities from the source words  $s_i$ , thus marginalizing over the alignments.

For  $s \mapsto t$  alignments, alignments are softmaxed per column because we hold the target  $t_j$  constant and iterate over the source  $s_i$ , hence  $\sigma_t$ .

The translation probabilities are softmaxed per row (i.e., source word  $s_i$ ) since we are given a source word and want to know the probability of translating  $s_i$  to  $t_j$ , hence  $\sigma_s$ .

However, as we do not maximize in neural networks, we must transform this equation into a minimization problem by taking the negative log of  $p$ :

$$-\log p(t, s | \theta, \psi) = - \sum_j^{|t|} \log \left[ \sum_i^{|s|} \sigma_s(\theta(t_j, s_i)) \cdot \sigma_t(\psi)_{ij} \right] \quad (26)$$

We can simplify this into the equation below, for easier implementation (via PyTorch's `logsumexp` function):

$$-\log p(t, s | \theta, \psi) = - \sum_j^{|t|} \log \left[ \sum_i^{|s|} \exp(\log \sigma_s(\theta(t_j, s_i)) + \log \sigma_t(\psi)_{ij}) \right] \quad (27)$$

This is the term to be minimized for our source-to-target word alignment generation.

## A.2 KL Divergence and Prior Terms

We also need to minimize the Kullback-Leibler (KL) divergence between our distribution and a prior. For discrete probability distributions  $P$  and  $Q$  defined on the same probability space, the reverse KL divergence from  $Q$  to  $P$  is defined as:

$$D_{\text{KL}}(Q \| P) = \sum_i Q(i) \log \left( \frac{Q(i)}{P(i)} \right) \quad (28)$$

In other words, it is the expectation of the log difference between the probabilities  $P$  and  $Q$ , where the expectation is taken using the probabilities  $Q$ .

To calculate the KL divergence of our model, we must define distributions  $P$  and  $Q$ . We first consider  $P$  as our prior distribution, henceforth called  $a_t$ . This prior distribution  $a_t$  is a  $n \times m$  matrix filled with the alignment probabilities filled for source word  $s_i$ ; target word  $t_j$ ; source sentence  $s$  of length  $n$ ; and target sentence  $t$  of length  $m$ :

We define distortion exponent  $h$  as:

$$h(i, j) = -\lambda \left| \frac{i}{n} - \frac{j}{m} \right| \quad (29)$$

$$Z_j = \sum_{i'} \exp h(i', j) \quad (30)$$

$$a_t(i, j) = \begin{cases} p_0 & \text{if null} \\ (1 - p_0) \cdot \frac{e^{h(i, j)}}{Z_j} & \text{else} \end{cases} \quad (31)$$

In [2], parameter values were selected as  $\lambda = 4$  and  $p_0 = 0.08$  for the entire corpus. Each element value of  $a_t$  is normalized by the sum of the column distortion values to create a valid distribution via term  $Z_j$  (since we hold target  $t_j$  constant). For future notation, let the subscript on the distribution  $a$  denote the way we normalize; i.e.,  $a_t$  is the prior alignment distribution normalized with respect to target words  $t_j$  and thus normalized by the sum of the column for  $t_j$ .

We can then write our KL Divergence (to be minimized) as:

$$D_{\text{KL}}(\sigma_t(\psi) \| a_t) = \sum_i^n \sum_j^m \sigma_t(\psi)_{ij} \cdot \log \left[ \frac{\sigma_t(\psi)_{ij}}{a_t(i, j)} \right] \quad (32)$$

### A.3 Target-to-Source Loss Terms

We have described the loss terms for a source  $s$  to target  $t$  word alignment. However, we seek to perform alignment by agreement. Hence, we must add loss functions that describe the evaluation of target  $t$  to source  $s$ :

$$-\log p(t, s | \theta, \psi) = - \sum_i^{|s|} \log \left[ \sum_j^{|t|} \exp(\log \sigma_t(\theta(t_j, s_i)) + \log \sigma_s(\psi)_{ij}) \right] \quad (33)$$

Additionally, for the KL Divergence term for target  $t$  to source  $s$ :

$$D_{\text{KL}}(\sigma_s(\psi) \| a_s) = \sum_i^n \sum_j^m \sigma_s(\psi)_{ij} \cdot \log \left[ \frac{\sigma_s(\psi)_{ij}}{a_s(i, j)} \right] \quad (34)$$

where the normalization of the prior  $a_s$  is on the row instead of the column. In other words, we hold the source  $s_i$  constant; sum the row; and divide each row element by the sum.

The equations for  $a_s$  are as follows:

$$h(i, j) = -\lambda \left| \frac{i}{n} - \frac{j}{m} \right| \quad (35)$$

$$Z_i = \sum_{j'} \exp h(i, j') \quad (36)$$

$$a_s(i, j) = \begin{cases} p_0 & \text{if null} \\ (1 - p_0) \cdot \frac{e^{h(i, j)}}{Z_i} & \text{else} \end{cases} \quad (37)$$

We also perform the softmax operator on each column of the  $\psi$  matrix, previously defined as  $\sigma_t(\psi)$ .

### A.4 Alignment by Agreement

Finally, one last loss function term must be added to jointly train each model. We define  $\circ$  as the Hadamard Product, which is the element-wise multiplication of two matrices. For instance:  $(A \circ B)_{ij} = A_{ij} \cdot B_{ij}$ .

We can write the term as such:

$$-\log \sum_i^{|s|} \sum_j^{|t|} [\sigma_s(\psi) \circ \sigma_t(\psi)]_{ij} \quad (38)$$

### A.5 Combined Loss Function

Our final loss function, to be minimized, is a 5-term equation. We define:

$a_t$  as the target prior alignment matrix normalized per column with respect to  $t$

$a_s$  as the source prior alignment matrix normalized per row with respect to  $s$

$\sigma_s$  as the softmax operator applied on the rows of a matrix

$\sigma_t$  as the softmax operator applied to each column of a matrix



Indexing is done under the assumption that source words  $s_i$  form rows and target words  $t_j$  form columns. The term  $ij$  is a shorthand for  $s_i$  and  $t_j$  indexing into our matrices.

$$\begin{aligned}
 Loss = & \\
 & - \sum_j^{|t|} \log \left[ \sum_i^{|s|} \exp (\log \sigma_s(\theta(t, s))_{ij} + \log \sigma_t(\psi)_{ij}) \right] \\
 & + \sum_i^n \sum_j^m \sigma_t(\psi)_{ij} \cdot \log \left[ \frac{\sigma_t(\psi)_{ij}}{a_t(i, j)} \right] \\
 & - \sum_i^{|s|} \log \left[ \sum_j^{|t|} \exp (\log \sigma_t(\theta(t, s))_{ij} + \log \sigma_s(\psi)_{ij}) \right] \\
 & + \sum_i^n \sum_j^m \sigma_s(\psi)_{ij} \cdot \log \left[ \frac{\sigma_s(\psi)_{ij}}{a_s(i, j)} \right] \\
 & - \log \sum_i^{|s|} \sum_j^{|t|} [\sigma_s(\psi) \circ \sigma_t(\psi)]_{ij}
 \end{aligned} \tag{39}$$