11 Symbolic MT 1: The IBM Models and EM Algorithm

Up until now, we have seen Section 3 discuss n-gram models that count up the frequencies of events, then Section 4 move to a model that used feature weights to express probabilities. Section 5 to Section 8 introduced models with increasingly complex structures that still fit within the general framework of Section 4: they calculate features and a softmax over the output vocabulary and are trained by stochastic gradient descent, instead of counting. In the following chapter, we will move back to models similar to the n-gram models that rely more heavily on counting and **symbolic** models, which treat things to translate as discrete symbols, as opposed to the continuous vectors used in neural networks.

11.1 Contrasting Neural and Symbolic Models

Like all of the models discussed so far, the models we'll discuss in this chapter are based on predicting the probability of an output sentence E given an input sentence F, $P(E \mid F)$. However, these models, which we will call *symbolic* models, take a very different approach, with a number of differences that I'll discuss in turn.

Method of Representation: The first difference between neural and symbolic models is the method of representing information. Neural models represent information as low-dimensional continuous-space vectors of features, which are in turn used to predict probabilities. In contrast, symbolic models – including n-grams from Section 3 and the models in this chapter – express information by explicitly remembering information about single words (discrete symbols, hence the name) and the correspondences between them. For example, an n-gram model might remember that "given a particular previous word e_{t-1}, what is the probability of the next word e_t". As a result, well-trained neural models often have superior generalization capability due to their ability to learn generalized features, while well-trained symbolic models are often better at remembering information from low-frequency training instances that have not appeared many times in the training data. Section 19 will cover a few models that take advantage of this fact by combining models that use both representations together.

Noisy-channel Representation: Another large difference is that instead of directly using the conditional probability $P(E \mid F)$ they use a **noisy-channel model** and model translation by dividing translation up into a separate **translation model** and **language model**. Specifically, remembering that our goal is to find a sentence that maximizes the translation probability:

$$\hat{E} = \operatorname*{argmax}_E P(E \mid F), \tag{94}$$

we can use Bayes's rule to convert to the following

$$\hat{E} = \operatorname*{argmax}_E \frac{P(F \mid E)P(E)}{P(F)}, \tag{95}$$

then ignore the probability $P(F)$ because F is given and thus will be constant regardless of the $\hat{E}$ we choose.

$$\hat{E} = \operatorname*{argmax}_E P(F \mid E)P(E). \tag{96}$$

We perform this decomposition for two reasons. First, it allows us to separate the models for $P(F \mid E)$ and $P(E)$, allowing us to create models of $P(F \mid E)$ that make certain simplifying

assumptions to make models easier (explained in a bit). The neural network models that we've explained before do not make these simplifying assumptions, thus sidestepping this issue. Second, it allows us to train the two models on different resources: $P(F \mid E)$ must be trained on bilingual data, which is relatively scarce, but $P(E)$ can be trained on monolingual data, which is available in large quantities. Because standard neural machine translation systems do not take this noisy channel approach, they are unable to directly use monolingual data, although there have been methods proposed to incorporate language models [8], train NMT systems by automatically translating target language data into the source language and using this as pseudo-parallel data to train the model [15], or even re-formulating neural models to follow the noisy channel framework [18].

Latent Derivations: A second difference from the models in previous sections is that these symbolic models are driven by a latent **derivation** D that describes the process by which the translation was created. Because we don't know which D is the correct one, we calculate the probability $P(E \mid F)$ or $P(F \mid E)$ by summing over these latent derivations as follows:

$$P(E \mid F) = \sum_D P(E, D \mid F). \tag{97}$$

It is also common to approximate this value by using the derivation with the maximum probability

$$P(E \mid F) = \max_D P(E, D \mid F). \tag{98}$$

The neural network models also had variables that one might think of as part of a derivation: the word embeddings, hidden layer states, and attention vectors. The important distinction between these variables and the ones in the model above is whether they have a probabilistic interpretation (i.e. whether they are random variables or not). In the cases mentioned above, the hidden variables in neural networks do not have any probabilistic interpretation – given the input, the hidden variables are simply calculated deterministically, so we do not have any concept of the probability of the hidden state $\boldsymbol{h}$ given the input $\boldsymbol{x}$, $P(\boldsymbol{h} \mid \boldsymbol{x})$. This probabilistic interpretation can be useful if we, for example, express interest in the latent representations (e.g. word alignments) and would like to calculate the probability of obtaining particular latent representations.[31]

11.2 IBM Model 1

Because this is all a bit abstract, let's go to a concrete example: **IBM Model 1** [3], an example of which is shown in Figure 31. Model 1 is a model for $P(F \mid E)$, and is extremely simple (in fact, over-simplified), in that it assumes that we first pick the number of words in the source $|F|$, then independently calculate the probability of each word in F. By doing so, we can assume that the probability takes the following form:

$$P(F \mid E) = P(|F| \mid E) \prod_{j=1}^{|F|} P(f_j \mid E). \tag{99}$$

Because $|F|$ is the length of the source sentence, which we already know, Model 1 does not make much effort to estimate this length, setting the probability to a constant: $P(|F| \mid E) = \epsilon$.

[31] While not a feature of vanilla neural networks, there are ways to think about neural networks in a probabilistic framework, which we'll discuss a bit more in Section 16.

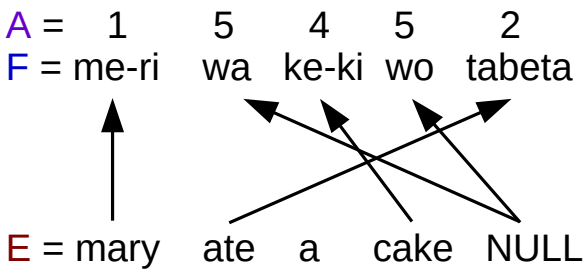

Figure 31: An example of the variables in IBM model 1.

More important is the estimation of the probability $P(f_j \mid E)$. This is done by assuming that f_j was generated by the following two-step process:

1. Randomly select an **alignment** a_j for word f_j. The value of the alignment variable is an integer $1 \leq a_j \leq |E| + 1$ corresponding to the word in E to which f_j corresponds. $e_{|E|+1}$ is a special `NULL` symbol, which is used as a catch-all token that can generate words that do not explicitly correspond to any other words in the source sentence. We assume that the alignment is generated according to a uniform distribution:

$$P(a_j \mid E) = 1/(|E| + 1). \tag{100}$$

2. Based on this alignment, calculate the probability of f_j according to $P(f_j \mid e_{a_j})$. This probability is a model parameter, which we learn using an algorithm described in the next section.

Putting these two probabilities together, we now have the following probability for the alignments and source sentence given the target sentence:

$$P(F, A \mid E) = P(|F| \mid E) \prod_{j=1}^{|F|} P(f_j \mid e_{a_j}) P(a_j \mid E), \tag{101}$$

$$= \epsilon \prod_{j=1}^{|F|} \frac{1}{|E| + 1} P(f_j \mid e_{a_j}). \tag{102}$$

$$= \frac{\epsilon}{(|E| + 1)^{|F|}} \prod_{j=1}^{|F|} P(f_j \mid e_{a_j}). \tag{103}$$

It should be noted that alignment A is one example of the derivation D described in the previous section. As such, according to Equation 97, we can also calculate the probability $P(E \mid F)$ by summing over the possible alignments A:

$$P(F \mid E) = \sum_{A} \frac{\epsilon}{(|E| + 1)^{|F|}} \prod_{j=1}^{|F|} P(f_j \mid e_{a_j}), \tag{104}$$

$$= \sum_{a_1=1}^{|E|+1} \sum_{a_2=1}^{|E|+1} \cdots \sum_{a_{|F|}=1}^{|E|+1} \frac{\epsilon}{(|E| + 1)^{|F|}} \prod_{j=1}^{|F|} P(f_j \mid e_{a_j}). \tag{105}$$

By noting that each sum over a variable a_j only affects a single element of the product on the right side of the equation, we can simplify to:

$$P(F \mid E) = \frac{\epsilon}{(|E|+1)^{|F|}} \prod_{j=1}^{|F|} \sum_{a_j=1}^{|E|+1} P(f_j \mid e_{a_j}). \tag{106}$$

Because this formula consists of a single product over $|F|$ elements and sum over $|E|+1$ elements, it can be calculated efficiently in $O(|F||E|)$ time.

Now that we have our model specified, we have two remaining questions to answer: the *learning* problem of finding the parameters specifying $P(f_j \mid e_{a_j})$, covered in Section 11.3, and the *search* problems of finding a translation E that maximizes the probability $P(E \mid F)$ or finding an alignment A that maximizes $P(A \mid F, E)$, which will be covered at the end of the chapter in Section 11.6.

11.3 The EM Algorithm

A general purpose learning algorithm for probabilistic models such as Model 1 is the **expectation-maximization algorithm** (EM algorithm; [4]). Put simply, EM is an iterative process that works by repeating two steps:

E Step: Based on the current model parameters, calculate the expectations of counts regarding latent variables D.

M Step: Based on these counts, update the model parameters.

This section will give an overall intuition of the EM algorithm using Model 1 as an example, and readers interested in the theory behind the algorithm can reference other materials, such as [1].

M Step: First, let's define a parameter $\theta_{f,e}$, which specifies $P(f|e;\theta)$ in the current model. If we know how many times source word f aligned to target word e, $c_{f,e}$, and how many times target word e appeared in the corpus c_e, we can calculate this parameter using maximum likelihood estimation, as we did in Section 3:

$$\theta_{f,e} = c_{f,e}/c_e. \tag{107}$$

This simple process is the M step of the EM algorithm.

E Step: So now the question becomes how we calculate the counts, which is the job of the E step. Calculating c_e is easy, we just count up the number of times a particular word appears in the corpus and we're done. Calculating $c_{f,e}$ is slightly more involved however, as it requires us to know the values of the latent alignments A, which are not given to us in our training data. In order to overcome this problem, we calculate the expectation of each value in A, $P(A \mid F, E)$ and use this to estimate the counts.

Because Model 1 decomposes nicely, the probabilities of each value a_j are independent, and thus, we can consider the probability of a single alignment $P(a_j = t|f_j, E)$ and further expand this using Bayes's rule:

$$P(a_j = t|f_j, E) = \frac{P(f_j|a_j = t, E)P(a_j = t|E)}{P(f_j|E)} \tag{108}$$

$$= \frac{P(f_j|a_j = t, E)P(a_j = t|E)}{\sum_{\tilde{t}=1}^{|E|+1} P(f_j|a_j = \tilde{t}, E)P(a_j = \tilde{t}|E)} \tag{109}$$

and take advantage of the fact that $P(a_j = t|E)$ is constant to cancel out

$$P(a_j = t|f_j, E) = \frac{P(f_j|a_j = t, E)}{\sum_{\tilde{t}=1}^{|E|+1} P(f_j|a_j = \tilde{t}, E)}. \tag{110}$$

So basically, we can calculate a probability for a_j by normalizing the translation probabilities for each word in $|E|$ by the sum over the entire sentence.

Once we have this alignment probability, we can then calculate the counts for the entire training corpus $\langle \mathcal{F}, \mathcal{E} \rangle$ as follows:

$$c_{f,e} = \sum_{\langle F,E \rangle \in \langle \mathcal{F}, \mathcal{E} \rangle} \sum_{j=1}^{|F|} \sum_{t=1}^{|E|+1} \delta(f_j = f, e_t = e) P(a_j = t \mid F, E), \tag{111}$$

where $\delta(f_j = f, e_t = e)$ is the Kronecker delta function:

$$\delta(f_j = f, e_t = e) = \begin{cases} 1 & \text{if } f_j = f \text{ and } e_t = e \\ 0 & \text{otherwise.} \end{cases} \tag{112}$$

Describing this in a more procedural way, similar to how this would actually be implemented in code, this means that for every iteration through the data we:

1. Initialize all counts $c_{e,f}$ to zero.
2. For every word f_j in every sentence pair $\langle F, E \rangle$
   (a) Calculate the probability $P(a_j = t|F, E)$ for every t.
   (b) Increment c_{f_j,e_t} by $P(a_j = t|F, E)$ for every t.

One thing to note is that this training algorithm also gives us a simple way to calculate one-best alignments $\hat{A}$ that maximize $P(A \mid F, E)$. In order to do so, we simply inspect the probabilities $P(a_j = t|F, E)$ that we calculated for each word j, and output the t that maximizes the probability.

11.4 IBM Model 2 and the HMM Model

As mentioned in the previous section, IBM Model 1 has a very simplified view of the world, where each word in the sentence is translated independently without any regard for word order. In this section, we introduce two models that consider word order.

The first model, **IBM Model 2** is based on the simple intuition that the reordering between sentences F and E essentially has a *canonical* word order. For example, if we are translating between French and English, because the word order in these two languages is quite similar, we could assume that aligned words will occur in roughly the same position in sentences in both languages. In equations, this boils down to saying:

$$a_j \approx j. \tag{113}$$

Of course, while this trend is true for many language pairs[32], for other language pairs with larger differences in word order this will not hold, and thus we prefer a model that can learn

[32] [6] use a model reflecting this rule with great effect.

the trends that each particular language follows from data. In order to do so, we replace the simple alignment model of Equation 100 with a more sophisticated one learned directly from data:

$$P(a_j = t \mid j, |F|, |E|) = c_{t,j,|F|,|E|}/c_{j,|F|,|E|}. \tag{114}$$

The estimation of the counts for $c_{t,j,|F|,|E|}$ can be done in the E step of the EM algorithm in a manner very similar to the estimation of $c_{f,e}$, allowing Model 2 to consider word order with minimal modifications to the training method for Model 1.[33]

A second, slightly more complicated model for considering word order is the **hidden Markov model** (HMM; [17]) alignment model. In contrast to Model 2, which was based on the idea of canonical word order for sentences of particular lengths, HMM alignment is based on the intuition that the position of the next a_j will tend to depend on the previous a_{j-1}. In other words, we calculate the probability

$$P(a_j = t \mid a_{j-1} = s, |F|, |E|) = c_{t,s,|F|,|E|}/c_{s,|F|,|E|}. \tag{115}$$

which can be similarly calculated using the EM algorithm.

However, in the case of the HMM, calculating expectations becomes a little bit more complicated. Because probabilities of each timestep j now depend on the alignments in the previous time step $j-1$, we can no longer use the independence assumptions that made Model 1 and Model 2 easy to calculate. Fortunately, however, it is possible to use a dynamic programming algorithm called the **forward-backward algorithm** that allows us to exactly calculate the expectations of alignments for the entire sentence. Similarly, we can calculate one-best alignments using the **Viterbi algorithm**, which we will cover in more detail in Section 12. Interested readers can find more details in [17].

11.5 IBM Models 3-5

IBM Models 3-5 gradually introduce more complexity into the modeling process for $P(F, A|E)$. Covering these in detail is beyond the scope of these materials, but because the IBM models are still used to obtain word alignments it is worth mentioning the basic ideas of these three models:

Model 3 introduces two concepts **fertility** and **distortion**. Fertility essentially models "given a particular word e_t, how many words in F is it likely to generate?" Many words in E will only have a single counterpart in F, leading their fertility to be 1. However, some words in E may lead to multiple words in F (e.g. "cats" in English will often be translated into "les chats" in French, leading to a fertility of 2). The distortion probability of Model 3 is similar to the alignment probability of Model 2, but estimated in the reverse direction, estimating j given t instead of t given j.

Model 4 modifies the distortion probability of Model 3 to handle *relative* distortion probabilities, conditioning the selection of the next word based on whether the previous word appeared. This is very similar in motivation to the HMM model.

Model 5 notes that Model 4 is *deficient*, assigning some probability mass to illegal configurations of the various variables (for example, translations where j is less than 1 or more

[33] *Question: What do the equations look like for this?*

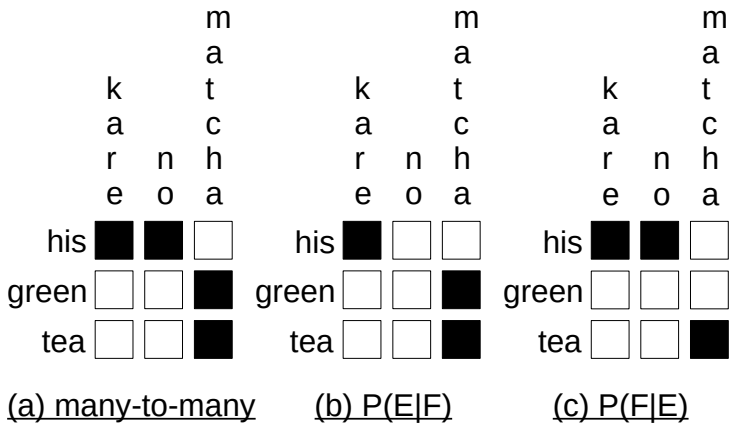

Figure 32: An example of (a) a many-to-many alignment where F is Japanese and E is English, (b) the alignments for an IBM model run in the $P(E|F)$ direction, and (c) alignments in the $P(F|E)$ direction.

than $|F|$). Model 5 fixes this, ensuring that all of the probability mass is assigned to legal configurations.

One thing to note about IBM Models 3-5 is that there no longer exist efficient exact training algorithms like the ones that we used for IBM Models 1 and 2, or the forward-backward algorithm for the HMM model. As a result, it is necessary to perform inexact inference using a number of tricks: models are implemented by greedily finding a reasonably good hypothesis, performing hill-climbing towards better hypotheses, and optimizing parameters based on the probabilities of hypotheses found during incomplete exploration of the space. The details of training these models are quite involved, interested readers can find details in the original paper, [3].

11.6 Synchronization and Evaluation of Alignments

One problem with the IBM and HMM Models is that they can only handle **one-to-many alignments**. We call these one-to-many because there is each word f_j corresponds to only a single word e_t, while in the opposite direction e_t may correspond to multiple words in F. As a result, if we look at a sentence such as the one in Figure 32, where we need to have many-to-one alignments in both directions, standard IBM models run in either direction are not able to obtain proper alignments. In order to solve this problem, it is common to run the IBM models in both directions $P(E|F)$ and $P(F|E)$, then perform **synchronization** the alignments according to heuristics such as the following [10]:

Intersection: Only use alignments discovered by both $P(E|F)$ and $P(F|E)$ models.

Union: Use alignments discovered by either or $P(E|F)$ or $P(F|E)$. In the example in Figure 32, this will result in the correct alignment.

Grow, Diag, Final, And: A variety of other heuristics that strike a middle ground between Intersection and Union were proposed by [10].

Once we have word alignments, we may want to assess how good they are. The most standard measure for doing so is **alignment error rate** (AER; [13]). Alignment error rate is measured by having a human annotator annotate "sure" and "possible" alignments, where sure alignments are words that are certainly aligned (such as two nouns that are exact translations of each-other), and possible alignments represent more difficult cases (such as phrasal alignments where a chunk of words are aligned, but it is not easy to tell exactly which words correspond to each-other).

11.7 Further Reading

Because alignment models play an instrumental role in symbolic translation models, there have been a number of improvements made over them.

Better generalization for alignment probabilities: One of the weak points of symbolic models is their lack of generalization. Specifically in the IBM models, the probabilities $P(f|e)$ for rare words are often not estimated properly, leading to mistaken alignments. There are a number of ways proposed to improve this, including the use of word classes [13] or neural-network based representations [16].

Constraints on alignment: It is also possible to put constraints on the alignments to encourage intuitive solutions and penalize unintuitive solutions. This has been done through encouraging models to discover similar alignments in both directions [11], or by adding soft constraints to solutions that encourage most words to have reasonable fertility values through **posterior regularization** [7].

Syntactic alignment: If syntactic trees are available on either side of the language pair, it is possible to learn alignments that are faithful to this syntactic structure [5].

Supervised alignment: Also, if hand-made alignments are available, the model can be trained to be faithful to these alignments [9, 14].

Phrasal alignment: Finally, it is possible to devise models that directly obtain many-to-many alignments without relying on the synchronization [2, 12].

11.8 Exercise

The exercise in this section will be to implement the training procedure for IBM Model 1. This will involve:

- Reading in the parallel corpus.
- Creating training code using the EM algorithm.
- Checking the log likelihood (Equation 106) to make sure that it is increasing at every iteration.
- Printing out the model parameters $\theta_{f,e}$, and visualizing the alignments obtained by the decoding algorithm of Section 11.6.

Potential improvements include:

- Implementing one of the more advanced IBM Models.
- Implementing synchronization heuristics.
- Measuring alignment accuracy on an annotated dataset.[34]

References

[1] Christopher M. Bishop. *Pattern recognition and machine learning*. 2006.

[2] Phil Blunsom, Trevor Cohn, Chris Dyer, and Miles Osborne. A Gibbs sampler for phrasal synchronous grammar induction. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 782–790, 2009.

[3] Peter F. Brown, Vincent J.Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–312, 1993.

[4] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[5] John DeNero and Dan Klein. Tailoring word alignments to syntactic machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 45, 2007.

[6] Chris Dyer, Victor Chahuneau, and Noah 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 (NAACL-HLT)*, pages 644–648, 2013.

[7] Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11(Jul):2001–2049, 2010.

[8] Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loic Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535*, 2015.

[9] Aria Haghighi, John Blitzer, John DeNero, and Dan Klein. Better word alignments with supervised ITG models. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 923–931, 2009.

[10] Philipp Koehn, Amittai Axelrod, Alexandra Birch Mayne, Chris Callison-Burch, Miles Osborne, and David Talbot. Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *Proceedings of the 2005 International Workshop on Spoken Language Translation (IWSLT)*, 2005.

[11] Percy Liang, Ben Taskar, and Dan Klein. Alignment by agreement. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 104–111, 2006.

[12] Graham Neubig, Taro Watanabe, Eiichiro Sumita, Shinsuke Mori, and Tatsuya Kawahara. An unsupervised model for joint phrase alignment and extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 632–641, 2011.

[34] Example: `http://www.phontron.com/kftt/#alignments`.

[13] Franz J. Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. *Proceedings of the 4th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 20–28, 1999.

[14] Jason Riesa and Daniel Marcu. Hierarchical search for word alignment. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 157–166, 2010.

[15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 86–96, 2016.

[16] Akihiro Tamura, Taro Watanabe, and Eiichiro Sumita. Recurrent neural networks for word alignment model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1470–1480, 2014.

[17] Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 836–841, 1996.

[18] Lei Yu, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Tomas Kocisky. The neural noisy channel. *arXiv preprint arXiv:1611.02554*, 2016.