Phrase-Based Translation Models

Michael Collins

April 10, 2013

1 Introduction

In previous lectures we've seen IBM translation models 1 and 2. In this note we will describe *phrase-based translation models*. Phrase-based translation models give much improved translations over the IBM models, and give state-of-the-art translations for many pairs of languages.

Crucially, phrase-based translation models allow lexical entries with more than one word on either the source-language or target-language side: for example, we might have a lexical entry

$$(\text{le chien}, \text{the dog})$$

specifying that the string le chien in French can be translated as the dog in English. The option of having multi-word expressions on either the source or target-language side is a significant departure from IBM models 1 and 2, which are essentially word-to-word translation models (i.e., they assume that each French word is generated from a single English word). Multi-word expressions are extremely useful in translation; this is the main reason for the improvements that phrase-based translation models give.

More formally, a phrase-based lexicon is defined as follows:

Definition 1 (Phrase-based lexicon) *A phrase-based lexicon $\mathcal{L}$ is a set of lexical entries, where each lexical entry is a tuple (f, e, g) where:*

- *f is a sequence of one or more foreign words.*
- *e is a sequence of one or more English words.*
- *g is a "score" for the lexical entry. The score could be any value in the reals.*

Note that there is no restriction that the number of foreign words and English words in a lexical entry should be equal. For example, the following entries would be allowed:

$$(\text{au}, \text{to the}, 0.5)$$

$$(\text{au banque}, \text{to the bank}, 0.01)$$

$$(\text{allez au banque}, \text{go to the bank}, -2.5)$$

(similar cases, where there are fewer English words than French words, would also be allowed). This flexibility in the definition of lexical entries is important, because in many cases it is very useful to have a lexical entry where the number of foreign and English words are not equal.

We'll soon describe how a phrasal lexicon $\mathcal{L}$ can be used in translation. First, however, we describe how a phrasal lexicon can be learned from a set of example translations.

2 Learning Phrasal Lexicons from Translation Examples

As before, we'll assume that our training data consists of English sentences $e^{(k)} = e_1^{(k)} \dots e_{l_k}^{(k)}$ paired with French sentences $f^{(k)} = f_1^{(k)} \dots f_{m_k}^{(k)}$, for $k = 1 \dots n$. Here the integer l_k is the length of the k'th English sentence, and $e_j^{(k)}$ is the j'th word in the k'th English sentence. The integer m_k is the length of the k'th French sentence, and $f_i^{(k)}$ is the i'th word in the k'th French sentence.

In addition to the sentences themselves, we will also assume that we have an *alignment matrix* for each training example. The alignment matrix $A^{(k)}$ for the k'th example has $l_k \times m_k$ entries, where

$$A_{i,j}^{(k)} = 1 \quad \text{if French word } i \text{ is aligned to English word } j, \text{ 0 otherwise}$$

Note that this representation is more general than the alignments considered for IBM models 1 and 2. In those models, we had alignment variables a_i for $i \in \{1 \dots m_k\}$, specifying which English word the i'th French word is aligned to. By definition, in IBM models 1 and 2 each French word could only be aligned to a single English word. With an alignment matrix $A_{i,j}^{(k)}$, the alignments can be many-to-many; for example, a given French word could be aligned to more than one English word (i.e., for a given i, we could have $A_{i,j}^{(k)} = 1$ for more than one value of j).

We'll remain agnostic as to how the alignment matrices $A^{(k)}$ are derived. In practice, a common method is something like the following (see the lecture slides, and the slides from Philipp Koehn's tutorial, for more details). First, we train IBM model 2, using the EM algorithm described in the previous lecture. Second, we use various heuristics to extract alignment matrices from the IBM model's output on each training example. To be specific, a very simple method would be as follows (the method is too naive to be used in practice, but will suffice as an example):

- Use the training examples $e^{(k)}, f^{(k)}$ for $k = 1 \dots n$ to train IBM model 2 using the EM algorithm described in the previous lecture. For any English string e, French string f, and French length m, this model gives a conditional probability $p(f, a|e, m)$.
- For each training example, define

  $$a^{(k)} = \arg\max_a p(f^{(k)}, a|e^{(k)}, m_k)$$

  i.e., $a^{(k)}$ is the most likely alignment under the model, for the k'th example (see the notes on IBM models 1 and 2 for how to compute this).

- Define

$$A^{(k)}_{i,j} = 1 \quad \text{if } a^{(k)}_i = j, \text{ 0 otherwise}$$

Assuming that we have derived an alignment matrix for each training example, we can now describe the method for extracting a phrase-based lexicon from a set of translation examples. Figure 1 shows a simple algorithm for this purpose. The input to the algorithm is a set of translation examples, with an alignment matrix for each training example. The algorithm iterates over all training examples ($k = 1 \ldots n$), and over all potential phrase pairs, where a phrase pair is a pair $(s,t),(s',t')$ where (s,t) is a sub-sequence within the source language sentence, and (s',t') is a sub-sequence within the target language sentence. For example, consider the case where the training example consists of the following sentences:

$$f^{(k)} = \text{wir müssen auch diese kritik ernst nehmen}$$

$$e^{(k)} = \text{we must also take these criticisms seriously}$$

then $(s,t) = (1,2)$, $(s',t') = (2,5)$ would correspond to the potential lexical entry

wir müssen, must also take these

For each possible (s,t), (s',t') pair, we test if it is *consistent* with the alignment matrix $A^{(k)}$: the function `consistent`$(A^{(k)},(s,t),(s',t'))$ returns true if the potential lexical entry ((s,t), (s',t')) is consistent with the alignment matrix for the training example. See figure 2 for the definition of the `consistent` function. Intuitively, the function checks whether any alignments from English or foreign words in the proposed lexical entry are to words that are "outside" the proposed entry. If any alignments are to outside words, then the proposed entry is not consistent. In addition, the function checks that there is at least one word in the English phrase aligned to some word in the foreign phrase.

For those phrases that are consistent, we add the lexical entry (f,e) to the lexicon, where $f = f_s \ldots f_t$, and $e = e_{s'} \ldots e_{t'}$. We also increment the counts $c(e,f)$ and $c(e)$, corresponding to the number of times that the lexical entry (f,e) is seen in the data, and the number of times the English string e is seen paired with *any* foreign phrase f. Finally, having extracted all lexical entries for the corpus, we define the score for any phrase (f,e) as

$$\log \frac{c(e,f)}{c(e)}$$

This can be interpreted as an estimate of the log-conditional probability of foreign phrase f, given English phrase e.

It is worth noting that these probabilities are in some sense heuristic—it is not clear what probabilistic model is underlying the overall model. They will, however, be very useful when translating with the model.

3 Translation with Phrase-Based Models

The previous described how a phrase-based lexicon can be derived from a set of training examples. In this section we describe how the phrase-based lexicon can be used to define a set of translations for

Inputs: $e^{(k)}, f^{(k)}, A^{(k)}$ for $k = 1 \ldots n$

Initialization: $\mathcal{L} = \emptyset$

Algorithm:

- For $k = 1 \ldots n$
  - For $s = 1 \ldots m_k$, for $t = s \ldots m_k$
    - For $s' = 1 \ldots l_k$, for $t' = s' \ldots l_k$
      - If $\texttt{consistent}(A^{(k)}, (s,t), (s',t')) = \texttt{True}$
        (1) Define $f = f_s^{(k)} \ldots f_t^{(k)}$, define $e = e_{s'}^{(k)} \ldots e_{t'}^{(k)}$
        (2) Set $\mathcal{L} = \mathcal{L} \cup \{(f, e)\}$
        (3) $c(e, f) = c(e, f) + 1$
        (4) $c(e) = c(e) + 1$
- For each $(f, e) \in \mathcal{L}$ create a lexical entry (f, e, g) where

$$g = \log \frac{c(e, f)}{c(e)}$$

Figure 1: An algorithm for deriving a phrasal lexicon from a set of training examples with alignments. The function $\texttt{consistent}(A^{(k)}, (s,t), (s',t'))$ is defined in figure 2.

Definition of $\texttt{consistent}(A, (s,t), (s',t'))$:
(Recall that A is an alignment matrix with $A_{i,j} = 1$ if French word i is aligned to English word j. (s,t) represents the sequence of French words $f_s \ldots f_t$. (s',t') represents the sequence of English words $e_{s'} \ldots f_{s'}$.)
For a given matrix A, define

$$A(i) = \{j : A_{i,j} = 1\}$$

Similarly, define

$$A'(j) = \{i : A_{i,j} = 1\}$$

Thus $A(i)$ is the set of English words that French word i is aligned to; $A'(j)$ is the set of French words that English word j is aligned to.
Then $\texttt{consistent}(A, (s,t), (s',t'))$ is true if and only if the following conditions are met:

1. For each $i \in \{s \ldots t\}$, $A(i) \subseteq \{s' \ldots t'\}$
2. For each $j \in \{s' \ldots t'\}$, $A'(j) \subseteq \{s \ldots t\}$
3. There is at least one (i, j) pair such that $i \in \{s \ldots t\}$, $j \in \{s' \ldots t'\}$, and $A_{i,j} = 1$

Figure 2: The definition of the `consistent` function.

a given input sentence; how each such translation receives a score under the model; and finally, how we can search for the highest scoring translation for an input sentence, thereby giving a translation algorithm.

3.1 Phrases, and Derivations

The input to a phrase-based translation model is a source-language sentence with n words, $x = x_1 \dots x_n$. The output is a sentence in the target language. The examples in this section will use German as the source language, and English as the target language. We will use the German sentence

wir müssen auch diese kritik ernst nehmen

as a running example.

A key component of a phrase-based translation model is a phrase-based lexicon, which pairs sequences of words in the source language with sequences of words in the target language, as described in the previous sections of this note. For example, lexical entries that are relevent to the German sentence shown above include

(wir müssen, we must)
(wir müssen auch, we must also)
(ernst, seriously)

and so on. Each phrase entry has an associated score, which can take any positive or negative value. As described before, a very simple way to estimate scores for phrases would be to define

$$g(f, e) = \log \frac{c(e, f)}{c(e)} \tag{1}$$

where f is a foreign sequence of words, e is an English sequence of words, and $c(e, f)$ and $c(e)$ are counts taken from some corpus. For example, we would have

$$g(\text{wir müssen, we must}) = \log \frac{c(\text{we must, wir müssen})}{c(\text{we must})}$$

The score for a phrase is then the log of the conditional probability of the foreign string, given the english string.

We introduce the following notation. For a particular input (source-language) sentence $x_1 \dots x_n$, a *phrase* is a tuple (s, t, e), signifying that the subsequence $x_s \dots x_t$ in the source language sentence can be translated as the target-language string e, using an entry from the phrase-based lexicon. For example, the phrase $(1, 2, \text{we must})$ would specify that the sub-string $x_1 \dots x_2$ can be translated as we must. Each phrase $p = (s, t, e)$ receives a score $g(p) \in R$ under the model. For a given phrase p, we will use $s(p)$, $t(p)$ and $e(p)$ to refer to its three components. We will use $\mathcal{P}$ to refer to the set of all possible phrases for the input sentence x.

Note that for a given input sentence $x_1 \dots x_n$, it is simple to compute the set of possible phrases, $\mathcal{P}$. We simply consider each substring of $x_1 \dots x_n$, and include all entries in the phrasal lexicon which

have this substring as their English string. We may end up with more than one phrasal entry for a particular source-language sub-string.

A *derivation* y is then a finite sequence of phrases, $p_1, p_2, \ldots p_L$, where each p_j for $j \in \{1 \ldots L\}$ is a member of $\mathcal{P}$. The length L can be any positive integer value. For any derivation y we use $e(y)$ to refer to the underlying translation defined by y, which is derived by concatenating the strings $e(p_1), e(p_2), \ldots e(p_L)$. For example, if

$$y = \text{(1, 3, we must also), (7, 7, take), (4, 5, this criticism), (6, 6, seriously)} \tag{2}$$

then

$$e(y) = \text{we must also take this criticism seriously}$$

3.2 The Set of Valid Derivations

We will use $\mathcal{Y}(x)$ to denote the set of valid derivations for an input sentence $x = x_1 x_2 \ldots x_n$. The set $\mathcal{Y}(x)$ is the set of finite length sequences of phrases $p_1 p_2 \ldots p_L$ which satisfy the following conditions:

- Each p_k for $k \in \{1 \ldots L\}$ is a member of the set of phrases $\mathcal{P}$ for $x_1 \ldots x_n$. (Recall that each p_k is a triple (s, t, e).)
- Each word is translated exactly once. More formally, if for a derivation $y = p_1 \ldots p_L$ we define

  $$y(i) = \sum_{k=1}^{L} [[s(p_k) \leq i \leq t(p_k)]] \tag{3}$$

  to be the number of times word i is translated (we define $[[\pi]]$ to be 1 if π is true, 0 otherwise), then we must have

  $$y(i) = 1$$

  for $i = 1 \ldots n$.
- For all $k \in \{1 \ldots L-1\}$,

  $$|t(p_k) + 1 - s(p_{k+1})| \leq d$$

  where $d \geq 0$ is a parameter of the model. In addition, we must have

  $$|1 - s(p_1)| \leq d$$

The first two conditions should be clear. The last condition, which depends on the parameter d, deserves more explanation.

The parameter d is a limit on how far consecutive phrases can be from each other, and is often referred to as a *distortion limit*. To illustrate this, consider our previous example derivation:

$$y = \text{(1, 3, we must also), (7, 7, take), (4, 5, this criticism), (6, 6, seriously)}$$

In this case $y = p_1 p_2 p_3 p_4$ (i.e., the number of phrases, L, is equal to 4). For the sake of argument, assume that the distortion parameter d, is equal to 4.

We will now address the following question: does this derivation satisfy the condition

$$|t(p_k) + 1 - s(p_{k+1})| \leq d \tag{4}$$

for $k = 1 \ldots 3$? Consider first the case $k = 1$. In this case we have $t(p_1) = 3$, and $s(p_2) = 7$. Hence

$$|t(p_1) + 1 - s(p_2)| = |3 + 1 - 7| = 3$$

and the constraint in Eq. 4 is satisfied for $k = 1$. It can be seen that the value of $|t(p_1) + 1 - s(p_2)|$ is a measure of how far the phrases p_1 and p_2 are from each other in the sentence. The distortion limit specifies that consecutive phrases must be relatively close to each other.

Now consider the constraint for $k = 2$. In this case we have

$$|t(p_2) + 1 - s(p_3)| = |7 + 1 - 4| = 4$$

so the constraint is satisfied (recall that we have assume that $d = 4$). For $k = 3$ we have

$$|t(p_3) + 1 - s(p_4)| = |5 + 1 - 6| = 0$$

Finally, we need to check the constraint

$$|1 - s(p_1)| \leq d$$

For this example, $s(p_1) = 1$, and the constraint is satisfied. This final constraint ensures that the first phrase in the sequence, p_1, is not too far from the start of the sentence.

As an example of a derivation that is invalid, because it does not satisfy the distortion constraints, consider

$$y = \text{(1, 2, we must), (7, 7, take), (3, 3, also), (4, 5, this criticism), (6, 6, seriously)}$$

In this case it can be verified that

$$|t(p_2) + 1 - s(p_3)| = |7 + 1 - 3| = 5$$

which is greater than the distortion limit, d, which is equal to 4.

The motivation for the distortion limit is two-fold:

1. It reduces the search space in the model, making translation with the model more efficient.
2. Empirically, it is often shown to improve translation performance. For many language pairs, it is preferable to disallow consecutive phrases which are a long distance from each other, as this will lead to poor translations.

However, it should be noted that the distortion limit is really a rather crude method for modeling word order differences between languages. Later in the class we will see systems that attempt to improve upon this method.

3.3 Scoring Derivations

The next question is the following: how do we score derivations? That is, how do we define the function $f(y)$ which assigns a score to each possible derivation for a sentence? The optimal translation under the model for a source-language sentence x will be

$$\arg \max_{y \in \mathcal{Y}(x)} f(y)$$

In phrase-based systems, the score for any derivation y is calculated as follows:

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})| \tag{5}$$

The components of this score are as follows:

- As defined before, $e(y)$ is the target-language string for derivation y. $h(e(y))$ is the log-probability for the string $e(y)$ under a trigram language model. Hence if $e(y) = e_1 e_2 \dots e_m$, then
$$h(e(y)) = \log \prod_{i=1}^{m} q(e_i | e_{i-2}, e_{i-1}) = \sum_{i=1}^{m} \log q(e_i | e_{i-2}, e_{i-1})$$
where $q(e_i | e_{i-2}, e_{i-1})$ is the probability of word e_i following the bigram e_{i-2}, e_{i-1} under a trigram language model.
- As defined before, $g(p_k)$ is the score for the phrase p_k (see for example Eq. 1 for one possible way of defining $g(p)$).
- η is a "distortion parameter" of the model. It can in general be any positive or negative value, although in practice it is almost always negative. Each term of the form
$$\eta \times |t(p_k) + 1 - s(p_{k+1})|$$
then corresponds to a penalty (assuming that η is negative) on how far phrases p_k and p_{k+1} are from each other. Thus in addition to having hard constraints on the distance between consecutive phrases, we also have a soft constraint (i.e., a penalty that increases linearly with this distance).

Given these definitions, the optimal translation in the model for a source-language sentence $x = x_1 \dots x_n$ is

$$\arg \max_{y \in \mathcal{Y}(x)} f(y)$$

3.4 Summary: Putting it all Together

Definition 2 (Phrase-based translation models) *A phrase-based translation model is a tuple* $(\mathcal{L}, h, d, \eta)$*, where:*

- *$\mathcal{L}$ is a phrase-based lexicon. Each member of $\mathcal{L}$ is a tuple (f, e, g) where f is a sequence of one or more foreign-language words, e is a sequence of one or more English words, and $g \in R$ is a score for the pair (f, e).*
- *h is a trigram language model: that is, for any English string $e_1 \dots e_m$,*

$$h(e_1 \dots e_m) = \sum_{i=1}^{m} \log q(e_i | e_{i-2}, e_{i-1})$$

  where q are the parameters of the model, and we assume that $e_{-1} = e_0 = *$, where $*$ is a special start symbol in the language model.
- *d is a non-negative integer, specifying the distortion limit under the model.*
- *$\eta \in R$ is the distortion penalty in the model.*

For an input sentence $x_1 \dots x_n$, define $\mathcal{Y}(x)$ to be the set of valid derivations under the model $(\mathcal{L}, h, d, \eta)$. The decoding problem is to find

$$\arg \max_{y \in \mathcal{Y}(x)} f(y)$$

where, assuming $y = p_1 p_2 \dots p_L$,

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

4 Decoding with Phrase-based Models

We now describe a decoding algorithm for phrase-based models: that is, an algorithm that attempts to find

$$\arg \max_{y \in \mathcal{Y}(x)} f(y) \tag{6}$$

where, assuming $y = p_1 p_2 \dots p_L$,

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

The problem in Eq. 6 is in fact NP-hard for this definition of $f(y)$; hence the algorithm we describe is an approximate method, which is not guaranteed to find the optimal solution.

A first critical data structure in the algorithm is a *state*. A state is a tuple

$$(e_1, e_2, b, r, \alpha)$$

where e_1, e_2 are English words, b is a bit-string of length n (recall that n is the length of the source-language sentence), r is an integer specifying the end-point of the last phrase in the state, and α is the score for the state.

Any sequence of phrases can be mapped to a corresponding state. For example, the sequence

$$y = \text{(1, 3, we must also), (7, 7, take), (4, 5, this criticism)}$$

would be mapped to the state

$$(\text{this}, \text{criticism}, 1111101, 5, \alpha)$$

The state records the last two words in the translation underlying this sequence of phrases, namely *this* and *criticism*. The bit-string records which words have been translated: the i'th bit in the bit-string is equal to 1 if the i'th word has been translated, 0 otherwise. In this case, only the 6'th bit is 0, as only the 6'th word has not been translated. The value $r = 5$ indicates that the final phrase in the sequence, $(4, 5, \text{this criticism})$ ends at position 5. Finally, α will be the score of the partial translation, calculated as

$$\alpha = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

where $L = 3$, we have

$$e(y) = \text{we must also take this criticism}$$

and

$$p_1 = \text{(1, 3, we must also)}, \quad p_2 = \text{(7, 7, take)}, \quad p_3 = \text{(4, 5, this criticism)}$$

Note that the state only records the last two words in a derivation: as will see shortly, this is because a trigram language model is only sensitive to the last two words in the sequence, so the state only needs to record these last two words.

We define the initial state as

$$q_0 = (*, *, 0^n, 0, 0)$$

where 0^n is bit-string of length n, with n zeroes. We have used * to refer to the special "start" symbol in the language model. The initial state has no words translated (all bits set to 0); the value for r is 0; and the score α is 0.

Next we define a function $\text{ph}(q)$ that maps a state q to the set of phrases which can be appended to q. For a phrase p to be a member of $\text{ph}(q)$, where $q = (e_1, e_2, b, r, \alpha)$, the following conditions must be satisfied:

- p must not overlap with the bit-string b. I.e., we must have $b_i = 0$ for $i \in \{s(p) \dots t(p)\}$.
- The distortion limit must not be violated. More specifically, we must have

  $$|r + 1 - s(p)| \leq d$$

  where d is the distortion limit.

In addition, for any state q, for any phrase $p \in \text{ph}(q)$, we define

$$\text{next}(q, p)$$

to be the state formed by combining state q with phrase p. Formally, if $q = (e_1, e_2, b, r, \alpha)$, and $p = (s, t, \epsilon_1 \dots \epsilon_M)$, then $\text{next}(q, p)$ is the state $q' = (e'_1, e'_2, b', r', \alpha')$ defined as follows:

- First, for convenience, define $\epsilon_{-1} = e_1$, and $\epsilon_0 = e_2$.
- Define $e'_1 = \epsilon_{M-1}$, $e'_2 = \epsilon_M$.
- Define $b'_i = 1$ for $i \in \{s \dots t\}$. Define $b'_i = b_i$ for $i \notin \{s \dots t\}$
- Define $r' = t$
- Define

$$\alpha' = \alpha + g(p) + \sum_{i=1}^{M} \log q(\epsilon_i | \epsilon_{i-2}, \epsilon_{i-1}) + \eta \times |r + 1 - s|$$

Hence e'_1 and e'_2 are updated to record the last two words in the translation formed by appending phrase p to state q; b' is an updated bit-string, which is modified to record the fact that words $s \dots t$ are now translated; r' is simply set to t, i.e., the end point of the phrase p; and α' is calculated by adding the phrase score $g(p)$, the language model scores for the words $\epsilon_1 \dots \epsilon_M$, and the distortion term $\eta \times |r + 1 - s|$.

The final function we need for the decoding algorithm is a very simple function, that tests for equality of two states. This is the function

$$\text{eq}(q, q')$$

which returns true or false. Assuming $q = (e_1, e_2, b, r, \alpha)$, and $q' = (e'_1, e'_2, b', r', \alpha')$, $eq(q, q')$ is true if and only if $e_1 = e'_1$, $e_2 = e'_2$, $b = b'$ and $r = r'$.

Having defined the functions ph, next, and eq, we are now ready to give the full decoding algorithm. Figure 3 gives the basic decoding algorithm. The algorithm manipulates sets Q_i for $i = 0 \dots n$. Each set Q_i contains a set of states corresponding to translations of length i (the length for a state q is simply the number of bits in the bit-string for q whose value is 1—that is, the number of foreign words that are translated in the state). Initially, we set Q_0 to contain a single state, the initial state q_0. We set all other sets Q_i for $i = 1 \dots n$ to be the empty set. We then iterate: for each $i \in \{1, 2, \dots n\}$, we consider each $q \in \text{beam}(Q_i)$ ($\text{beam}(Q_i)$ is a subset of Q_i, containing only the highest scoring elements of Q_i: we will give a formal definition shortly). For each $q \in \text{beam}(Q_i)$ we first calculate the set $\text{ph}(q)$; then for each $p \in \text{ph}(q)$ we calculate the next state $q' = \text{next}(q, p)$. We add this new state to the set Q_j where j is the length of state q'. Note that we always have $j > i$, so we are always adding elements to states that are further advanced than the set Q_i that we are currently considering.

Figure 4 gives a definition of the function $\text{Add}(Q, q', q, p)$. The function first checks whether there is an existing state q'' in Q such that $eq(q'', q')$ is true. If this is the case, then q' replaces q'' if its score is higher than q''; otherwise q' is not added to Q. Hence only one of the states q'' and q' remains in Q. If there is no such state q'', then q' is simply added to Q.

Note that the Add function records backpointers $bp(q')$ for any state q' added to a set Q. These backpointers will allow us to recover the final translation for the highest scoring state. In fact, the final step of the algorithm it to: 1) find the highest scoring state q in the set Q_n; 2) from this state recover the highest scoring translation, by tracing the backpointers for the states.

Finally, we need to define the function $\text{beam}(Q)$; this definition is given in Figure 5. This function first computes α^*, the highest score for any state in Q; it then discards any state with a score that is less than $\alpha^* - \beta$, where $\beta > 0$ is the *beam-width* of the decoding algorithm.

- Inputs: sentence $x_1 \ldots x_n$. Phrase-based model $(\mathcal{L}, h, d, \eta)$. The phrase-based model defines the functions $\text{ph}(q)$ and $\text{next}(q, p)$.
- Initialization: set $Q_0 = \{q_0\}$, $Q_i = \emptyset$ for $i = 1 \ldots n$.
- For $i = 0 \ldots n-1$
  - For each state $q \in \text{beam}(Q_i)$, for each phrase $p \in \text{ph}(q)$:
    (1) $q' = \text{next}(q, p)$
    (2) $\text{Add}(Q_j, q', q, p)$ where $j = \text{len}(q')$
- Return: highest scoring state in Q_n. Backpointers can be used to find the underlying sequence of phrases (and the translation).

Figure 3: The basic decoding algorithm. $\text{len}(q')$ is the number bits equal to 1 in the bit-string for q' (i.e., the number of foreign words translated in the state q').

$\text{Add}(Q, q', q, p)$

- If there is some $q'' \in Q$ such that $eq(q'', q') = \text{True}$:
  - If $\alpha(q') > \alpha(q'')$
    - $Q = \{q'\} \cup Q \setminus \{q''\}$
    - set $bp(q') = (q, p)$
  - Else return
- Else
  - $Q = Q \cup \{q'\}$
  - set $bp(q') = (q, p)$

Figure 4: The $\text{Add}(Q, q', q, p)$ function.

Definition of $\text{beam}(Q)$: define

$$\alpha^* = \arg\max_{q \in Q} \alpha(q)$$

i.e., α^* is the highest score for any state in Q.
Define $\beta \geq 0$ to be the *beam-width* parameter
Then

$$\text{beam}(Q) = \{q \in Q : \alpha(q) \geq \alpha^* - \beta\}$$

Figure 5: Definition of the beam function in the algorithm in figure 3.