Chapter 3

Machine Translation

3.1 Problem (again)

Remember that we motivated the language modeling problem by thinking about machine translation as “deciphering” the source language into the target language.

\[
P(f, e) = P(e) P(f | e) \quad \text{(3.1)}
\]

\[
\hat{e} = \arg \max_e P(e | f) \quad \text{(3.2)}
\]

\[
= \arg \max_e \frac{P(e, f)}{P(f)} \quad \text{(3.3)}
\]

\[
= \arg \max_e P(e, f) \quad \text{(3.4)}
\]

\[
= \arg \max_e P(e) P(f | e). \quad \text{(3.5)}
\]

In this chapter, we start by focusing on \(P(f | e)\). But we’ll also consider so-called direct models that estimate \(P(e | f)\), in particular neural networks.

All the models we’ll look at are trained on parallel text, which is a corpus of text that expresses the same meaning in two (or more) different languages. Usually we assume that a parallel text is already sentence-aligned, that is, it consists of sentence pairs, each of which expresses the same meaning in two languages. In the original work on statistical machine translation (Brown et al., 1993), the source language was French (f) and the target language was English (e), and we’ll use those variables even for other language pairs. Our example uses Spanish and English.

Here is an example parallel text (Knight, 1999):

1. Garcia and associates
   García y asociados

2. his associates are not strong
   sus asociados no son fuertes
3.2 Word Alignment

We want to define a model for generating Spanish sentences $f$ from English sentences $e$. Let’s make the simplifying assumption that each Spanish word depends on exactly one English word. For example:

1. García and associates EOS
   \[ \begin{array}{c}
   \text{García} \\
   \text{y asociados}
   \end{array} \]
   \[ \text{EOS} \]

2. his associates are not strong EOS
   \[ \begin{array}{c}
   \text{sus asociados no son}
   \end{array} \]
   \[ \text{fuertes EOS} \]

(We’ve made some slight changes compared to the original paper, which did not use EOS. But the basic idea is the same.)

More formally: let $\Sigma_f$ and $\Sigma_e$ be the Spanish and English vocabularies, and

- $f = f_1 \cdots f_n$ range over Spanish sentences ($f_n = \text{EOS}$)
- $e = e_1 \cdots e_m$ range over English sentences ($e_m = \text{EOS}$)
- $a = (a_1, \ldots, a_n)$ range over possible many-to-one alignments, where $a_j = i$ means that Spanish word $j$ is aligned to English word $i$.

We will use these variable names throughout this chapter. Remember that $e$, $i$, and $m$ come alphabetically before $f$, $j$, and $n$, respectively.

Thus, for our two example sentences, we have

1. $f = \text{García y asociados EOS}$
   $e = \text{Garcia and associates EOS}$
   $a = (1, 2, 3, 4)$

2. $f = \text{sus asociados no son fuertes EOS}$
   $e = \text{his associates are not strong EOS}$
   $a = (1, 2, 4, 3, 5, 6)$.

These alignments $a$ will be included in our “story” of how an English sentence $e$ becomes a Spanish sentence $f$. In other words, we are going to define a model of $P(f, a \mid e)$, not $P(f \mid e)$, and training this model will involve summing over all alignments $a$:

\[
\text{maximize } L = \sum_{(f,e)\in \text{data}} \log P(f \mid e) \\
= \sum_{(f,e)\in \text{data}} \log \sum_a P(f, a \mid e).
\]  

(3.6)  

(3.7)

(This is similar to training of NFAs in the previous chapter, where there could be more than one accepting path for a given training string.)
3.3 Model 1

IBM Model 1 (Brown et al., 1993) is the first in a series of five seminal models for statistical word alignment. The basic generative story goes like this.

1. Generate each alignment $a_1, \ldots, a_n$, each with uniform probability $\frac{1}{m}$.

2. Generate Spanish words $f_1, \ldots, f_n$, each with probability $t(f_j \mid e_{a_j})$.

In equations, the model is:

$$P(f, a \mid e) = \prod_{j=1}^{n} \left( \frac{1}{m} t(f_j \mid e_{a_j}) \right). \quad (3.8)$$

The parameters of the model are the word-translation probabilities $t(f \mid e)$. We want to optimize these parameters to maximize the log-likelihood,

$$L = \sum_{(f, e) \in \text{data}} \log \sum_a P(f, a \mid e). \quad (3.9)$$

The summation over $a$ is over an exponential number of alignments; as with NFAs, we can rearrange this to make it efficiently computable:

$$\sum_a P(f, a \mid e) = \sum_{a_1=1}^{m} \cdots \sum_{a_n=1}^{m} \prod_{j=1}^{n} \left( \frac{1}{m} t(f_j \mid e_{a_j}) \right) \quad (3.10)$$

$$= \sum_{a_1=1}^{m} \frac{1}{m} t(f_1 \mid e_{a_1}) \cdots \sum_{a_n=1}^{m} \frac{1}{m} t(f_n \mid e_{a_n}) \quad (3.11)$$

$$= \prod_{j=1}^{n} \sum_{e_1=1}^{m} \frac{1}{m} t(f_j \mid e_i). \quad (3.12)$$

The good news is that this objective function is convex, that is, every local maximum is a global maximum. The bad news is that there’s no closed-form solution for this maximum, so we must use some iterative approximation. The classic way to do this is expectation-maximization, but we can also use stochastic gradient ascent. The trick is ensuring that the $t$ probabilities sum to one. We do this by defining a matrix $T$ with an element for every pair of Spanish and English words. The elements are unconstrained real numbers (called logits), and are the new parameters of the model. Then we can use the softmax function to change them into probabilities, which we use as the $t$ probabilities.

$$T \in \mathbb{R}^{|V_r| \times |V_e|} \quad (3.13)$$

$$t(f \mid e) = \text{softmax} T_{.,e} \quad (3.14)$$

$$= \frac{\exp T_{f,e}}{\sum_{f' \in V_r} \exp T_{f',e}} \quad (3.15)$$

where $T_{.,e}$ means “the $e$’th column of $T$.”

For large datasets, $t(f \mid e)$ should be zero for the vast majority of $(f, e)$ pairs, which means that the vast majority of entries of $T$ would be $-\infty$. So to make this practical, we’d have to store $T$ as a sparse matrix.
### 3.4 Model 2 and beyond

In Model 1, we chose each $a_j$ with uniform probability $1/m$, which makes for a very weak model. For example, it’s unable to learn that the first Spanish word is more likely to depend on the first English word than (say) the seventh English word. In Model 2, we replace $1/m$ with a learnable parameter:

$$P(f, a | e) = \prod_{j=1}^{n} \left( a(i | j, m, n) t(f_j | e_{a_j}) \right).$$

where for each $i, j, m, n$, the parameter $a(i | j, m, n)$ must be learned. (This notation follows the original paper; I hope it’s not too confusing that $a_j$ is an integer but $a(\cdot)$ is a probability distribution.) Then we can learn that (say) $a(1 | 1, 10, 10)$ is high, but $a(7 | 1, 10, 10)$ is low.

There are also Models 3, 4, and 5, which can learn dependencies between the $a_j$, like:

- **Distortion**: Even if the model gives low probability to $a_1 = 7$, it should be the case that given $a_1 = 7$, the probability that $a_2 = 8$ is high, because it’s common for a block of words to move together.

- **Fertility**: It should be most common for one Spanish word to align to one English word, less common for zero or two Spanish words to align to one English word, and extremely rare for ten Spanish words align to one English word.

But for our purposes, it’s good enough to stop here at Model 2.

To train Model 2 by stochastic gradient ascent, we again need to express the $a$ probabilities in terms of unconstrained parameters. Let $M$ and $N$ be the maximum English and Spanish sentence length, respectively. Then:

$$A \in \mathbb{R}^{M \times N \times M \times N}$$

$$a(i | j, m, n) = \begin{cases} \text{softmax} A_{i,j,m,n} & \text{if } i = \text{max } A \end{cases}$$

$$a(i | j, m, n) = \frac{\exp A_{i,j,m,n}}{\sum_{i'} \exp A_{i',j,m,n}}.$$

Based on the progression of topics in the previous chapter, you might expect me at this point to show how the IBM models are instances of weighted finite automata. For a fixed $e$ and $n$, you can indeed construct a weighted finite automaton that generates strings $f$ with probability $P(f | e)$ under Model 1 or 2. But there isn’t a single machine (that I know of) that can read in any string $e$ and output strings $f$ with probability $P(f | e)$. Fear not, however; I still have something nutty in store.

### 3.5 From Alignment to Attention

So far, we’ve been working in the noisy-channel framework,

$$P(f, e) = P(e) P(f | e).$$

(3.19)
One reason for doing this is to divide up the translation problem into two parts so each model (language model and translation model) can focus doing its part well. But neural networks are rather good at doing two jobs at the same time, and so modern MT systems don’t take a noisy-channel approach. Instead, they directly model $P(e \mid f)$. Let’s start by rewriting Model 1 in the direct direction:

$$
P(e \mid f) = \prod_{i=1}^{m} \sum_{j=1}^{n} \frac{1}{n} \left[ \text{softmax} \ T_{fj} \right]_{c_i}.
$$

(3.20)

See Figure 3.2a for a picture of this model, drawn in the style of a neural network.

**Factoring $T$.** Above, we mentioned that matrix $T$ is very large and sparse. We can overcome this by factoring it into two smaller matrices (see Figure 3.2b):

$$
U \in \mathbb{R}^{|E| \times d}
$$

(3.21)

$$
V \in \mathbb{R}^{|S| \times d}
$$

(3.22)

$$
T = UV^T
$$

(3.23)

So the model now looks like

$$
P(e \mid f) = \prod_{i=1}^{m} \sum_{j=1}^{n} \frac{1}{n} \left[ \text{softmax} \ UV_{fj} \right]_{c_i}
$$

(3.24)

If you think of $T$ as transforming Spanish words into English words (more precisely, logits for English words), we’re splitting this transformation into two steps. First, $V$ maps the Spanish word into a size-$d$ vector, called a word embedding. This transformation $V$ is called an embedding layer because it embeds the Spanish vocabulary into the vector space $\mathbb{R}^d$ which is (somewhat sloppily) called the embedding space.

Second, $U$ transforms the hidden vector into a vector of logits, one for each English word. This transformation $U$, together with the softmax, are known as a softmax layer. The rows of $U$ can also be thought of as embeddings of the English words.

In fact, for this model, we can think of $U$ and $V$ as embedding both the Spanish and English vocabularies into the same space. Figure 3.1 shows that if we run factored Model 1 on a tiny Spanish-English corpus (Knight, 1999) and normalize the Spanish and English word embeddings, words that are translations of each other do lie close to each other.

The choice of $d$ matters. If $d$ is large enough (at least as big as the smaller of the two vocabularies), then $UV^T$ can compute any transformation that $T$ can. But if $d$ is smaller, then $UV^T$ can only be an approximation of the full $T$ (called a low-rank approximation). This is a good thing: not only does it solve the sparse-matrix problem, but it can also generalize better. Imagine that we have training examples

1. El perro es grande.
   The dog is big.

2. El perro es gigante.
   The dog is big.
Figure 3.1: Two-dimensional visualization of the 64-dimensional word embeddings learned by the factored Model 1. The embeddings were normalized and then projected down to two dimensions using t-SNE (Maaten and Hinton, 2008). In most cases, the Spanish word embedding is close to its corresponding English word embedding.
   The dog is large.

The original Model 1 would not be able to learn a nonzero probability for \( t(\text{gigante} | \text{large}) \). But the factorized model would map both \textit{grande} and \textit{gigante} to nearby embeddings (because both translate to \textit{big}), and map that region of the space to \textit{large} (because \textit{gigante} translates to \textit{large}). Thus it would learn a nonzero probability for \( t(\text{gigante} | \text{large}) \).

**Attention.** To motivate the next change, consider the Spanish-English sentence pairs

1. por favor
   please
2. por ejemplo
   for example

Model 1 would learn to generate \textit{please} when the source sentence contains \textit{por or favor}. Specifically, it would learn \( t(\text{please} | \text{por}) = \frac{1}{2} \), so if you asked it to re-translate \textit{por ejemplo}, it would prefer the translation \textit{please example} over \textit{for example}. What we really want is to generate \textit{please} when the source sentence contains \textit{por and favor}.

We can get this if we move the average \( \sum_{j=1}^{n} \frac{1}{n} [\cdot] \) inside the softmax. It can go anywhere inside, but let’s put it between \( V \) and \( U \) (see Figure 3.2c):

\[
P(e | f) = \prod_{i=1}^{m} \left[ \text{softmax} \left( U \sum_{j=1}^{n} \frac{1}{n} V f_j \right) \right]_{e_i}.
\]  \tag{3.25}

Remember that the softmax contains an exp in it, so moving the summation inside has (roughly) the effect of changing it into a product – in other words, changing an \textit{or} into an \textit{and}. So we can now generate \textit{please} when the source sentence contains \textit{por and favor}. Suppose \( U \) and \( V \) have the following values:

\[
U = \begin{pmatrix}
0 & 1 & 0 \\
1 & 0 & 1 \\
0 & 0 & 1
\end{pmatrix}, \quad V^T = \begin{pmatrix}
5 & 10 & 0 \\
10 & 0 & 0 \\
0 & 0 & 10
\end{pmatrix} \tag{3.26}
\]

If \( f = \text{por favor} \), then

\[
P(\text{please} | f) = 0.924
\]  \tag{3.27}
\[
P(\text{for} | f) = 0.076
\]  \tag{3.28}
\[
P(\text{example} | f) = 0.000
\]  \tag{3.29}

but if \( f = \text{por ejemplo} \), then

\[
P(\text{please} | f) = 0.039
\]  \tag{3.30}
\[
P(\text{for} | f) = 0.480
\]  \tag{3.31}
\[
P(\text{example} | f) = 0.480.
\]  \tag{3.32}
Figure 3.2: Variations of IBM Model 1, pictured as a neural network.
If the $V_{f_j}$ can be thought of as vector representations of words, then the average $\frac{1}{n} \sum_j V_{f_j}$ can be thought of as a vector representation of the whole sentence $f$. Recall that going from Model 1 to Model 2, we changed the uniform average into a weighted average, weighted by the parameters $a(j \mid i)$. Similarly, here, we can make the uniform average into a weighted average $\frac{1}{n} \sum_j a(j \mid i) V_{f_j}$.

At each time step $i$, the weights $a(j \mid i)$, which must sum to one ($\sum_j a(j \mid i) = 1$), provide a different “view” of $f$. This mechanism is known as attention, and the network is said to attend to different parts of the sentence at different times. The weights $a(j \mid i)$ are called attention weights, and the weighted average is sometimes called the context vector.

In Model 2, $a(j \mid i)$ just depended on the lengths $n$ and $m$, but in general, we can let it depend on anything in $f$ and $e$. These days, the attention weights are usually factored like we did for $t(\cdot \mid \cdot)$ earlier:

$$Q = \mathbb{R}^{m \times d} \quad (3.34)$$

$$K = \mathbb{R}^{n \times d} \quad (3.35)$$

$$a(j \mid i) = \left[ \text{softmax}(KQ) \right]_j . \quad (3.36)$$

This is called dot-product attention and intuitively works as follows. For each Spanish word $f_j$, the network computes a vector $K_j$, called a key. This vector could encode the position $j$, the word $f_j$, or any other words in $f$.

Then, when generating English word $e_i$, the network computes a vector $Q_i$, called a query. This vector could depend on the position $i$, or any preceding words $e_1, \ldots, e_{i-1}$. The above definition makes the network attend most strongly to Spanish words $f_j$ whose keys $K_j$ are most similar to the query $Q_i$. 
