Learning Underlying Forms With MaxEnt

Sarah Eisenstat
Advisor: Mark Johnson

May 15, 2009

Abstract

It has been shown previously that a Maximum Entropy model can be used to learn the correct weights on linguistic constraints in the supervised case [4]. In this paper, we show how to extend the existing work to learn underlying forms in addition to linguistic constraints, discussing the practical details of the priors and the constraint sets that allow this to work. In addition, we discuss a technique to factor variables out of the loss function, thus allowing for more efficient computation of an exponential number of possibilities. Also included is a discussion of how to extend the results to handle sampling.

1 The Problem

Phonology is the area of linguistics concerned with sound and pronunciation. Phonology has given us the International Phonetic Alphabet, an alphabet that can be used to represent any pronunciation of any word in any language. Each letter of the alphabet corresponds directly to a sound, unlike in English where the letter "g" can be pronounced in at least three or four different ways. (Consider its pronunciation in the words "good," "laugh," "night," and "gin" — each one is pronounced differently.) Each of these letters of the IPA has various phonological features. One such feature is "voicedness," indicating whether the sound is pronounced using the vocal cords or not. "b" is the voiced version of "p"; "z" is the voiced version of "s."

Morphology is the area of linguistics concerned with putting together small elements of meaning into larger words. For example, we know that the English plural is usually formed by appending an "s" to the end of a word. That plural "s" is known as a morpheme; the base word that it is appended to is also a morpheme. Morphology and phonology often interact a great deal. For instance, in the English plural case, we know that the plural of "bush" is not "bushs"; instead, it's "bushes." That is because it is almost impossible to pronounce "bushs" as written.

When analyzing words, we must take both morphology and phonology into account. In general, a word is composed of morphemes, each of which has some particular meaning, and some **underlying form** — the form of the morpheme that is concatenated with other morphemes to create a word, and then changed due to phonological properties to make the **surface form**. In general, underlying forms are displayed in slashes, like /this/, while surface forms are displayed in brackets, like [this].

This kind of analysis — determining the underlying forms for each of the morphemes, and the phonological rules that produce the surface form — is the problem that we wish to address. The following is a small Polish data set, which gives an example of the type of data that we would ideally like to be able to analyze. The spellings have been divided up into stem and suffix, indicated by a + sign:

meaning	**IPA spelling**
"sheaf singular"	[snop+Ø]
"sheaf plural"	[snop+i]
"club singular"	[klup+Ø]
"club plural"	[klub+i]

What is the result when we analyze this linguistically? Because "sheaf singular" and "club singular" share the null suffix [Ø], it is likely that the singular form of nouns is indicated with a null suffix. Similarly, because two plurals share the suffix [i], it seems likely that the plural form of nouns is indicated with the suffix /i/. By similar reasoning, we can conclude that the noun "sheaf" is indicated with the stem /snop/. But what about "club"? Although the two stems in the singular and plural are similar, they don't completely match — in the singular, the final consonant in the stem is [p], while in the plural, the final consonant in the stem is [b]. What explains this?

Well, we know that either /klup/ or /klub/ is likely to be the true underlying form of the stem. What if it were /klup/? Then there must be some phonological rule that transforms the /p/ in the underlying form to a [b] in the surface form of the word. But what kind of rule would act only on the /p/ in /klup/, but not on the /p/ in /snop/? Both are preceded by a vowel, and both are followed by the plural form /i/. There is nothing in their immediate context that would cause the pronunciation of one to shift, but not the other.

Therefore, we posit /klub/ as the true underlying form. That /b/ then becomes a [p] in the singular form, when it occurs at the end of the word. This is actually a fairly commonly-known linguistic phenomenon known as "word-final devoicing."

The kind of analysis given above is very common to introductory linguistics classes. But why is this problem interesting? Linguists have already solved this problem, which is why it mostly only shows up in introductory linguistics classes. However, consider what happens when a child is attempting to learn a language. Isn't this exactly the sort of task that the child must solve? Studies have shown that children start out learning individual words, and then at some point begin to try to analyze those words, resulting in some children saying "goed" instead of "went" or other such over-normalization. We know that this process occurs in children; we don't entirely know how. So, like some other problems in computational linguistics, this problem is interesting because it gives us insight into the process of human language learning.

Throughout this paper, we will be discussing both this small Polish example, and the problem of the English plural. In addition to the "bushs" example given above, there is a slighly more subtle phonological change in the English plural: the word "cats" ends in an "s" sound, whereas the word "dogs" ends in a "z" sound. In IPA spelling, this problem can be represented as follows:

meaning	IPA spelling
"cat singular"	[kat+Ø]
"cat plural"	[kat+s]
"dog singular"	[dog+Ø]
"dog plural"	[dog+z]
"bush singular"	[buʃ+Ø]
"bush plural"	[buʃ+ɪz]

Note that from the data given, it's impossible to tell which of the three given suffixes is the correct underlying one. However, other English data indicates that the underlying form of the plural is /z/, which becomes unvoiced when preceded by an unvoiced consonant and is followed by the end of the word. The [ɪ] is then inserted between two stridents (stridents include /s/, /z/, and /ʃ/).

2 Prior Work

2.1 Supervised Learning

Consider what would happen with the above Polish example if we knew all four of the correct underlying forms, and just wanted to learn the phonological rules that produced the surface form? Then we would know the following:

meaning	underlying	surface
"sheaf singular"	/snop+Ø/	[snop+Ø]
"sheaf plural"	/snop+i/	[snop+i]
"club singular"	/klub+Ø/	[klup+Ø]
"club plural"	/klub+i/	[klub+i]

In this case, we don't have to worry about the morphological structure of the word, only about the phonological processes going on underneath. This problem is the **supervised** case of this problem, and has been studied in great detail using many different methods, some of which we now describe.

2.2 Optimality Theory

In traditional linguistic theory, the phonological changes from underlying to surface forms are viewed as a sequence of transitions conditioned on contexts. For instance, one way to write the change in voicing for English is:

$$z \rightarrow s/[\text{-voiced}]\_\#$$

The # indicates a word boundary; the _ indicates that the change occurs when the underlying /z/ is in a context surrounded by the end of the word and an unvoiced segment. The analysis for English also includes an insertion rule between two stridents. In this type of analysis, the phonological process is represented as an ordered series of rules of this sort, which are applied in order to produce the correct surface form.

However, in Optimality Theory and many of the theories spawned from it, a different type of analysis is used [7]. Optimality Theory says that all of the changes that we observe

on the surface are as a result of constraints on what sorts of things are pronouncible, and constraints on what sorts of things can be changed, inserted, or deleted from words. Also, each constraint can be violated multiple times — for instance, if more than one letter is changed from the underlying to the surface form, then that constraint is violated twice. Violating a constraint more times is worse than violating it fewer times. Constraints are also ordered, so that higher-ranked constraints are worse to violate, and lower-ranked constraints are better to violate. This ranking then produces an ordering on potential surface forms. The best of those surface forms is the correct surface form.

Constraints that determine whether a surface form is pronouncible are called **markedness constraints**, because they usually include patterns of phonological features on the surface forms that appear very rarely, and are therefore marked. For the English case, there are two markedness constraints in play:

- StridStrid: having two stridents in a row is marked.
- VoicedEnd: having a voiced strident following an unvoiced consonant at the end of the word is marked.

Constraints that determine what sorts of things can be changed, inserted, or deleted from words are called **faithfulness constraints** — a violation of faithfulness occurs when a segment in the surface form does not match the corresponding segment in the underlying form that it is aligned to. There are several different types of common faithfulness constraints:

- Insertion: inserting is marked.
- Deletion: deleting is marked.
- VowelInsertion: inserting a vowel is marked.
- ConsonantInsertion: inserting a consonant is marked.
- VowelDeletion: deleting a vowel is marked.
- ConsonantDeletion: deleting a consonant is marked.
- Faithfulness: any change in a segment from the underlying to the surface form is marked.
- FeatureFaith(x): a change in feature x from underlying to surface is marked.

There has been research into ways to make it possible for a computer to learn the correct constraint ranking [9], thereby learning the phonological rules that produce the surface form.

2.3 Maximum Entropy

One variant of Optimality Theory is called a Harmonic Grammar [6]. Once again, we have constraints of various sorts, but now instead of ranking the constraints, we give each constraint a weight. The weight of a potential surface form is the sum over all constraints of the number of violations of that constraint times the weight of the constraint. Then the best surface form is the one with the lowest weight, because it is violating the fewest important constraints.

Maximum Entropy takes this approach a little further [4]. Once again, we have a set of constraints, each of which has a weight associated with it. However, instead of simply choosing the surface form of the "best" value, we calculate a probability distribution over surface values using the weight values for the constraints.

More formally, say that we have a set of constraint functions $\mathbf{f} = (f_1, f_2, \ldots f_m)$. The input for each constraint function is a triple (u, s, a), where u is the underlying word, s is the surface word, and a gives the alignment between underlying and surface. Each constraint function outputs the number of violations of that constraint caused by the underlying-surface-alignment triple (u, s, a). Now say that we are given weights $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ associated with each of those constraints. Then the unnormalized probability of an underlying-surface-alignment triple (u, s, a) is defined as:

$$p(u, s, a, \mathbf{w}) = e^{\mathbf{w} \cdot \mathbf{f}(u,s,a)} = e^{\sum_{j=1}^{m} w_j f_j(u,s,a)}$$

This means that the greater $p(u, s, a, \mathbf{w})$ is, the more likely we are to generate the triple (u, s, a). Since each constraint function f_j gives the number of violations, and we want to penalize triples that have more violations, this means that we want our constraint weights to be negative.

We then need to normalize this probability. In previous work on this topic, the partition function used was simply the sum of $p(u, s, a, \mathbf{w})$ over all triples (u, s, a), including all triples that do not contain the correct underlying and/or surface form. Let Ω be the set of all triples that we are considering. Then the partition function is formally defined as:

$$Z(\mathbf{w}) = \sum_{(u,s,a) \in \Omega} p(u, s, a, \mathbf{w})$$

Therefore, the probability of a triple (u, s, a) is defined as:

$$\Pr(u, s, a \mid \mathbf{w}) = \frac{p(u, s, a, \mathbf{w})}{Z(\mathbf{w})}$$

That probability gives us the probability of seeing the triple (u, s, a) given the weights $\mathbf{w}$.

How can we use this to let the computer learn the correct weights on each of the features f_j? Well, just as in most probabilistic models, we want to pick our weights $\mathbf{w}$ so that they're the most likely ones given our data. In other words, if D is our data set, we want to find:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \Pr(\mathbf{w} \mid D)$$

Because of Bayes' law, we know that:

$$\Pr(\mathbf{w} \mid D) = \frac{\Pr(D \mid \mathbf{w}) \cdot \Pr(\mathbf{w})}{\Pr(D)}$$

Because the probability of D is constant across different possible choices of $\mathbf{w}$, we can ignore it when we try to find the $\mathbf{w}$ maximizing that value. So what we need to do is find:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \Pr(D \mid \mathbf{w}) \cdot \Pr(\mathbf{w})$$

How do we calculate $\Pr(D \mid \mathbf{w})$? We know that some triples (u, s, a) are correct, and the other triples are incorrect. In fact, we have a list $D = ((u_1, s_1, a_1), \ldots, (u_n, s_n, a_n))$ of triples that we have seen. The probability of someone producing those triples given our weights $\mathbf{w}$ is:

$$\Pr(D \mid \mathbf{w}) = \prod_{i=1}^{n} \Pr(u_i, s_i, a_i \mid \mathbf{w})$$

This allows us to calculate the probability of our data given the grammar. So if we decide on a value for $\Pr(\mathbf{w})$ — known as the prior — then we can optimize over all $\mathbf{w}$ to find the best possible assignment of weights to produce the data we have seen.

To some extent, we don't know what the value of $\Pr(\mathbf{w})$ should be — how can we figure out whether one weight set is good and the other is bad? So it might be tempting to set $\Pr(\mathbf{w})$ to be proportional to some constant, so that all weight functions are equally good. But consider what happens if we find some good weight function $\mathbf{w}$ that generates all of the good underlying-surface-alignment triples with high probability, and everything else with low probability. Now consider what happens if we scale $\mathbf{w}$ by some constant c:

$$\Pr(u, s, a \mid c\mathbf{w}) = \frac{p(u, s, a, c\mathbf{w})}{Z(c\mathbf{w})} = \frac{p(u, s, a, \mathbf{w})^c}{\sum_{(u,s,a)\in\Omega} p(u, s, a, \mathbf{w})^c}$$

If $p(u, s, a, \mathbf{w})$ is high, then $\Pr(u, s, a \mid c\mathbf{w})$ will get higher as c increases. In other words, there is no maximum — if we were to optimize this function, it would simply keep increasing. In order to limit this, we use a simple gaussian prior to keep the weights closer to zero:

$$\Pr(\mathbf{w}) \propto e^{-\sum_{j=1}^{m} \frac{w_j^2}{\sigma_j}}$$

The values σ_j are width parameters for the gaussian, and usually set to some single σ value for all j. This will ensure that all weights remain fairly close to zero, with each σ controlling how close to zero the weight should remain. High values of σ allow the constraint weights to vary more; low values restrict it to be closer to zero. Note that we only need to know what $\Pr(\mathbf{w})$ is proportional to, because we are using it to find the weights $\mathbf{w}$ maximizing that function.

Rather than optimize our likelihood function, we can optimize the log likelihood. This gives us the following function to optimize:

$$\sum_{i=1}^{n} \log \Pr(u_i, s_i, a_i \mid w) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C = \left(\sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(u_i, s_i, a_i) \right) - n \cdot \log Z(\mathbf{w}) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C$$

How do we take the derivative of this function? The first and third terms are a polynomial function of the weights w_j, and so it is easy to calculate their derivatives. This means that we must find the derivative of the more complex $\log Z(\mathbf{w})$:

$$\frac{\partial}{\partial w_j} \log Z(\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \cdot \sum_{(u,s,a)\in\Omega} \frac{\partial}{\partial w_j} \left(e^{\mathbf{w}\cdot\mathbf{f}(u,s,a)} \right) = \frac{1}{Z(\mathbf{w})} \cdot \sum_{(u,s,a)\in\Omega} f_j(u, s, a) \cdot e^{\mathbf{w}\cdot\mathbf{f}(u,s,a)}$$

Note that the form of this derivative is very familiar:

$$\mathrm{E}[f_j(u, s, a) \mid \mathbf{w}] = \sum_{(u,s,a)\in\Omega} \Pr(u, s, a \mid \mathbf{w}) \cdot f_j(u, s, a) = \frac{\partial}{\partial w_j} \log Z(\mathbf{w})$$

This means that the partial derivative of the log probability with respect to some weight w_j is:

$$\frac{\partial}{\partial w_j} \log \Pr(\mathbf{w} \mid D) = \sum_{i=1}^{n} \left(f_j(u_i, s_i, a_i) - \mathrm{E}[f_j(u, s, a) \mid \mathbf{w}] \right) - \frac{2w_j}{\sigma_j}$$

3 Unsupervised Learning

In the unsupervised case, the problem is somewhat different. Instead of being given a list of underlying-surface-alignment triples $((u_1, s_1, a_1), \ldots, (u_n, s_n, a_n))$, we are now only given a list D of surface forms $(s_1, \ldots, s_n)$. We must somehow use those correct surface forms to learn both the correct underlying forms and the correct phonological rules that result in those surface forms.

To do so, we wish to extend the supervised MaxEnt framework to handle this. The first problem we face is that we can no longer use the correct underlying form u_i and alignment a_i in conjunction with the observed surface form s_i in order to calculate the probability. So instead of calculating the probability of an underlying-surface-alignment triple (u, s, a), we must be able to calculate the probability of a surface form.

We know that if a surface form appears, it must have come from some underlying form. So the probability of seeing a surface form can be defined as the probability of seeing any triple with that surface form. More formally:

$$\Pr(s^* \mid \mathbf{w}) = \sum_{(u,s,a)\in\Omega \text{ s.t. } s=s^*} \Pr(u, s, a \mid \mathbf{w})$$

Then we can define the probability of our dataset $s_1, \ldots, s_n$ given the weights $\mathbf{w}$ as follows:

$$\Pr(s_1, \ldots, s_n \mid \mathbf{w}) = \prod_{i=1}^{n} \Pr(s_i \mid \mathbf{w}) = \prod_{i=n}^{\ell} \left(\sum_{(u,s,a)\in\Omega \text{ s.t. } s=s_i} \Pr(u, s, a \mid \mathbf{w}) \right)$$

Unfortunately, simply changing the function we wish to optimize is not sufficient. Currently, the only constraints that we have in effect are faithfulness constraints and markedness constraints. Markedness constraints are only affected by the surface form, and so all triples with the same surface form will receive the same weight from markedness constraints. Faithfulness constraints can only penalize changes between the underlying and surface forms, and so of all of the potential underlying forms to match the surface form, the one that receives the highest weight (and is therefore most likely to be the underlying form) will always be the one that most closely matches the surface form. This means that this system, as currently proposed, will always propose the underlying form that is equal to the surface form, with no changes. In the Polish case, that means that this system will propose /klup+Ø/ as the underlying form of [klup+Ø], and /klub+i/ as the underlying form of [klub+i]. So it won't even propose a consistent underlying form for the morpheme meaning "club."

How can we fix this? We fix this by introducing a new type of constraint: an **underlying form constraint**. This type of constraint will report a violation when it sees a particular underlying form. For instance, let UNDER(CLUB, /klub/) be the constraint associated with the underlying form /klub/ for the morpheme meaning "club." Here is a table showing what it would evaluate to on various sample underlying-surface pairs (we ignore alignment here because it does not affect the underlying form):

underlying	**surface**	**value of constraint**
/klub+i/	[klub+i]	1
/klup+i/	[klub+i]	0
/klub+i/	[klup+i]	1
/klup+i/	[klup+i]	0

If we have one of each type of constraint associated with every possible underlying form for every morpheme type, then we can learn weights on these constraints that tell us which of these underlying forms is correct for each morpheme type.

How does this change affect our optimization? Just as in the supervised case, we can select the weights $\mathbf{w}$ optimizing this function by selecting the weights that optimize the log probability. For the sake of more compact notation, we defined the set $\Omega(s^*) = \{(u, s, a) \in \Omega \mid s = s^*\}$. Then the log probability of some weights given our data is:

$$\begin{aligned}
\log \Pr(\mathbf{w} \mid s_1, \ldots, s_n) &= \log \left(\prod_{i=1}^{n} \Pr(s_i \mid \mathbf{w}) \right) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C \\
&= \sum_{i=1}^{n} \log \Pr(s_i \mid \mathbf{w}) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C \\
&= \sum_{i=1}^{n} \left(\log \left(\sum_{(u,s,a) \in \Omega(s_i)} p(u, s, a, \mathbf{w}) \right) - \log Z(\mathbf{w}) \right) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C
\end{aligned}$$

What happens when we take the derivative of this function with respect to some weight w_j? We have already seen how to take the derivative of the log of the partition function, and how to take the derivative of the log of the prior. So we calculate:

$$\begin{aligned}
\frac{\partial}{\partial w_j} \log \sum_{(u,s,a) \in \Omega(s_i)} p(u, s, a, \mathbf{w}) &= \frac{\sum_{(u,s,a) \in \Omega(s_i)} \frac{\partial}{\partial w_j} p(u, s, a, \mathbf{w})}{\sum_{(u,s,a) \in \Omega(s_i)} p(u, s, a, \mathbf{w})} \\
&= \sum_{(u,s,a) \in \Omega(s_i)} \frac{f_j(u, s, a) \cdot p(u, s, a, \mathbf{w})}{\sum_{(u',s',a') \in \Omega(s_i)} p(u', s', a', \mathbf{w})} \\
&= \mathrm{E}\left[f_j(u, s, a) \mid \mathbf{w}, s = s_i \right]
\end{aligned}$$

This means that the derivative works out to:

$$\log \Pr(\mathbf{w} \mid s_1, \ldots, s_n) = \sum_{i=1}^{n} \left(\mathrm{E}\left[f_j(u, s, a) \mid \mathbf{w}, s = s_i \right] - \mathrm{E}\left[f_j(u, s, a) \mid \mathbf{w} \right] \right) - \frac{2 w_j}{\sigma_j}$$

4 Contrastive Estimation

In this section, we consider the effects of a technique known as **contrastive estimation**, as introduced by Smith and Eisner in their 2005 paper [8]. Consider what happens when we are evaluating the probability of some underlying-surface-alignment triple (u, s, a). Whether in the supervised or unsupervised case, we have what is called a **focus set** of underlying-surface-alignment triples — the set of triples, for each observed data item, that contributes to the probability of generating that data item. When optimizing, we want to increase the probability of the items in the focus set.

For the supervised case, there is only one correct triple that could have generated the observed underlying-surface-alignment triple, so that triple is the only thing in the focus set

for that data item. More mathematically speaking, for the data item (u_i, s_i, a_i), the focus set consists of:

$$F_{\text{supervised}}(u_i, s_i, a_i) = \{(u, s, a) \in \Omega \mid u = u_i \text{ and } s = s_i \text{ and } a = a_i\}$$

For the unsupervised case, there is a set of correct underlying-surface-alignment triples that could have generated the observed surface form: any triple containing the correct surface form. Therefore, the focus set for the data item s_i is:

$$F_{\text{unsupervised}}(s_i) = \{(u, s, a) \in \Omega \mid s = s_i\}$$

This definition of focus sets means that when we have some data item d and a focus set $F(d)$, regardless of whether we're in the supervised or unsupervised case, we can write the probability of d given $\mathbf{w}$ as:

$$\Pr(d \mid \mathbf{w}) = \sum_{(u,s,a) \in F(d)} \frac{p(u, s, a, \mathbf{w})}{Z(\mathbf{w})}$$

The concept of a focus set expresses the idea that we have some set of underlying-surface-alignment triples that we want to increase the probability for. We can define, in a similar manner, the **contrast set** of a triple (u, s, a) — that is, the set of underlying-surface-alignment triples over which we are taking the probability of seeing (u, s, a). In both of the cases we have seen before, supervised and unsupervised, the contrast set consists of the set of all triples. This means that the contrast set for some data item d is:

$$C(d) = \Omega$$

We can then use this definition of C to redefine our partition function to allow for greater generality:

$$Z(d, \mathbf{w}) = \sum_{(u,s,a) \in C(d)} p(u, s, a, \mathbf{w})$$

Our current choice of contrast set means that the probability of some triple (u, s, a) is the probability of it being produced when selected at random from the set of all potential underlying-surface-alignment triples. In the supervised Polish case, this means that we have four correct underlying-surface-alignment triples. The probability of the data will therefore be maximized when each of those triples receives probability 1/4.

But do we really care about the difference between the probability of seeing (/snop+Ø/, [snop+Ø]) and the probability of seeing (/klub+i/, [klub+i])? All we want to do is determine the correct underlying form for "sheaf singular" or "club plural" — we have no real purpose for learning the probability of seeing a word that means "sheaf singular" rather than one that means "club plural."

This is the principle behind as contrastive estimation. The basic idea is that, in MaxEnt, we really only care about increasing the probability of our focus set with respect to its near neighbors, not the entire space Ω. In addition, using a smaller contrast set can make the probability easier to compute, especially when handling very large values of n, or dong stochastic gradient ascent, or sampling.

Contrastive estimation is particularly appropriate in this application. We can then define our contrast set for some data item to be the set of all triples related to the meaning of that data item. This means that the probability of some triple (u, s, a) is no longer the probability of producing it when trying to generate any possible underlying-surface-alignment triple. Instead, the probability of some triple (u, s, a) is the probability of producing it when trying to generate a word of some particular meaning.

How do these changes affect the derivative of the log probability? The function we wish to optimize is:

$$\begin{aligned}\log \Pr(\mathbf{w} \mid D) &= \sum_{i=1}^{n} \log \Pr(d_i \mid \mathbf{w}) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C \\ &= \sum_{i=1}^{n} \left(\log \sum_{(u,s,a) \in F(d)} p(u, s, a, \mathbf{w}) - \log \sum_{(u,s,a) \in C(d)} p(u, s, a, \mathbf{w}) \right) - \sum_{j=1}^{m} \frac{w_j^2}{\sigma_j} + C\end{aligned}$$

This means that the partial derivative with respect to some w_j is:

$$\begin{aligned}&\frac{\partial}{\partial w_j} \log \Pr(\mathbf{w} \mid D) = \\ &\quad \sum_{i=1}^{n} \left(\mathrm{E}\left[f_j(u, s, a) \mid (u, s, a) \in F(d_i), \mathbf{w} \right] - \mathrm{E}\left[f_j(u, s, a) \mid (u, s, a) \in C(d_i), \mathbf{w} \right] \right) - \frac{2 w_j}{\sigma_j}\end{aligned}$$

5 Implementation Details

5.1 Potential Triples

How do we choose the sets of potential underlying-surface-alignment triples? We would like to allow morpheme alternate forms of varying lengths — in case there are insertions or deletions — and we would like to allow for any possible form of that length. Unfortunately, this means that the set of potential forms for any single morpheme is infinite. In a subsequent section, we address this problem. For now, however, we must restrict our set of morpheme alternate forms in some way, while still allowing it to vary enough to handle uncommon cases.

Let us first consider the simpler problem: trying to find morpheme alternates with no insertion or deletion of segments. This means that we only have to consider alternate forms of the same length as our observed morphemes, and we only have to consider one type of alignment. But even with this restriction, we must handle an exponential number of potential morphemes if we allow for every possible character combination of that length. More specifically, if the set of characters available is C and the length of the morpheme is n, then the set of possibilities has size $|C|^n$. As soon as you have any morpheme of significant length, iterating through all of those alternates becomes computationally infeasible. So instead, we return to linguistics to let us know which potential underlying forms we should consider.

Say that we have seen two morphemes that mean the same thing, but which are pronounced differently. For instance, let's consider the [klub] and [klup] in the Polish example. If we line up the two morphemes character by character, we observe that they only differ

in one character: the final character can be a [b] or a [p]. All of the other characters are the same. This means that we have observed an alternation between [b] and [p], but no alternations between the other characters. Because languages tend to value faithfulness between the underlying and surface forms, if two characters are never observed to alternate, it's unlikely that they will. It's a little like Occam's Razor — if we see a [k] in the first position of the morpheme meaning "club" every single time, then it's most likely that the underlying form was a /k/ as well, because any other underlying form would require a lot more explanation.

This gives us the inspiration for the set of potential alternate morphemes. For each character c, we can construct an **alternation set** consisting of the set of all characters that we might reasonably expect c to alternate with. One way to do this might be to let the alternation set of c consist of all characters that c has been observed to alternate with. However, I chose to use slightly larger alternation set: the alternation set for a character c consists of all characters that were observed to alternate with c, as well as all characters that were observed to alternate with the alternates of c, and so on. In other words, if we start with $A(c)$ as the set of all characters that c is observed to alternate with (including c itself), then we perform the following algorithm to get the new alternate character sets:

```
changed = true
while changed:
    changed = false
    for each character c:
        S = ∅
        for each character c_2 ∈ S:
            S = S ∪ A(c_2)
        if A(c) ≠ S:
            changed = true
        A(c) = S
```

What do we do now that we have these alternation sets? It's very straightforward: we simply say that wherever we see the character c, we can potentially substitute any of the characters in $A(c)$. This could get exponential quite quickly, but in practice this tends to produce no more than 8 alternatives per morpheme, which is far more manageable than $|C|^n$, especially since $|C| \approx 30$ and n can be as large as 4.

This covers the Polish dataset, which has no insertion and deletion between the underlying and surface forms. However, this does not explain how to generate potential alternates for cases when we must handle insertion and deletion. As previously mentioned, if we allowed arbitrary amounts of insertion and deletion, the resulting set of alternate morpheme forms would be infinite. Thus, once again we will limit the alternate forms of our morphemes to allow only reasonable amounts of insertion and deletion.

First, in order to perform this task, we must be given an alignment between every morpheme. If we did not have such an alignment, then we wouldn't even be able to perform the simple character alternations explained above, because the difference in lengths means that we don't know which characters in one version of a morpheme correspond to which characters in the other, and so we can't even learn which characters alternate.

So assume that we have a set of morphemes, for which we know an alignment with each other morpheme of its type. Using this alignment, we can convert each morpheme of the same meaning to strings of the same length, with null characters inserted where there are insertions or deletions. For the English case, this means that the data set becomes:

meaning	**IPA spelling**
"cat singular"	[kat+Ø]
"cat plural"	[kat+Øs]
"dog singular"	[dɔg+Ø]
"dog plural"	[dɔg+Øz]
"bush singular"	[buʃ+Ø]
"bush plural"	[buʃ+ɪz]

This way, we see three plural morphemes: [Øs], [Øz], and [ɪz]. From those morphemes, we know that the [s] is aligned with the [z], and the [ɪ] was either inserted or deleted. Once we have converted each of our morphemes to this new aligned format, we can perform the same technique as with the underlying forms in the case of no deletions and insertions. This does not give the full range of potential underlying and surface forms — for example, it tends to ignore the possibility of insertions in morphemes where there was no insertion in the first place — but it gives us a good approximation of the space, which can then be used to learn underlying and surface forms.

Once we have the potential alternate forms of each morphemes, how do we generate the set of potential underlying-surface-alignment triples? The set of potential underlying forms for some meaning, such as "club plural," is simply the cross product of the set of potential forms for the stem and the set of potential forms for the suffix. The set of potential surface forms is created in a similar way, and the set of potential underlying-surface-alignment triples is simply the cross-product of the potential underlying forms with the potential surface forms. In all cases, the alignments are straightforward: we align each character in the underlying form with the character in the same position in the surface form.

5.2 Constraints

Now that we have this system for learning weights on constraints, and in doing so learning the underlying forms of various morphemes, which constraints should we use?

For underlying form constraints, we need one for every underlying morpheme that occurs in our choice of Ω, so that we can make sure to learn weights for each form of each morpheme.

What about faithfulness constraints? There are two different sets of faithfulness constraints that we have experimented with: the first is just a single flat faithfulness constraint, that fires whenever there is a violation of faithfulness between the underlying and surface forms; the second is a set of constraints, one for each phonological feature, indicating whether there is a change in that feature for some segment between underlying and surface forms. Although the first method seems to work on small datasets, it falls apart on larger ones; the latter seems to work on datasets of all sizes.

What about markedness constraints? In traditional OT and its variants, the standard is to use a small set of markedness constraints, targeted at the particular example we are attempting to learn weights for [7]. In general, this will include some constraints which are necessary for the linguistically correct analysis, and some constraints which are necessary for

a linguistically incorrect but sort of plausible analysis. It is generally considered a success if the computer correctly learns the linguistically correct analysis.

This approach does work in the data sets that we have tested. For example, the Polish dataset has two possible interpretations for the difference between the surface forms of "club": either the voiced /b/ becomes an unvoiced [p] at the end of the word, or the unvoiced /p/ becomes a voiced [b] between two vowels (which are also voiced). Those interpretations correspond to the following two constraints:

- VoicedEnd: a final voiced consonant is marked.
- VoicedUnvoicedVoiced: unvoiced consonant between two voiced segments is marked.

With the restrictions from section 5.3, the prior from section 5.4, and the alternate morpheme sets from section 5.1, the learner comes up with the correct underlying forms for the data. In addition, it correctly assigns a weight near zero to the second constraint.

Just to make sure that this wasn't a fluke, we also ran a similar test using a small Hungarian dataset with similar properties:

meaning	**IPA spelling**
"gap (subject)"	[res+Ø]
"gap (inside)"	[rez+ben]
"water (subject)"	[viz+Ø]
"water (inside)"	[viz+ben]

This Hungarian data is exactly the opposite of the Polish data: instead of the voicing change being a result of word-final devoicing, it's a result of voicing between voiced segments. When the learner is fed the Hungarian data, under the same circumstances as the Polish data, it correctly learns that the second constraint is the one in force here, and the first constraint should have weight almost zero.

However, this approach, although successful, seems almost too easy: just having to decide between two alternate interpretations makes the search for underlying forms a lot easier, and may in fact bias the learner towards the correct underlying forms, if we feed it markedness constraints that are too specific. So instead, we have chosen a more general approach. Constraints consist of a series of two or three phonological features, including a feature indicating a word boundary. The constraint fires when its features are seen in sequence in a word. So for instance, the above constraints can be written as a sequence of features as follows:

- VoicedEnd: $\{+\text{voiced}, \#\}$
- VoicedUnvoicedVoiced: $\{+\text{voiced}, -\text{voiced}, +\text{voiced}\}$

Therefore, if our feature set is F, then the possible sequences of features used for our constraint set is $(F \times F) \cup (F \times F \times F)$. Since $|F| \approx 20$, this means that we are working with at least 8000 constraints. This means that if we are trying to evaluate the unnormalized probability of even one underlying-surface-alignment triple, we must take the dot-product of two vectors in $\mathbb{R}^{8000}$. If we were doing that once or twice, it would not contribute much at all to the overall runtime. But if we have a data set of size $n \approx 700$, with each data item resulting in around 20 elements of the contrast set, then just calculating the probability of our data given the weights takes over 100 million computations. Calculating the derivative

in addition to the probability is even more computationally intensive. And since we are taking the derivative many times in order to perform optimization, this computation can contribute a lot to the time required for computation.

Still, this approach ensures that our choice of constraints is not making our task artificially easy by limiting the number of possible interpretations. However, this new constraint set introduces other issues. Say that we have decided that $\{+\text{voiced}, \#\}$ is marked, and is the only rule that should be in effect. That means that it should receive some amount of weight — say, w_i. But we can duplicate the effect of setting its weight to w_i by considering all constraints that contain those two features as a substring, and assigning a fraction of w_i to those constraints. What does this accomplish? Well, the smaller the weights on the constraints are, the smaller the prior associated with those constraints is. Therefore, we have achieved the same data likelihood, with a smaller prior. This means that the learner will, instead of learning the correct weight for a single constraint, will learn a distribution of weights on a bunch of constraints.

How can we prevent this? With the prior. Once again, we turn to linguistics to figure out what is reasonable for this. Linguistically speaking, constraints tend to be spread out over very few features: a change in voicing is most likely caused by some number of surrounding voiced segments, rather than some number of stridents, or laterals. Having some complex constraint like $\{+\text{voiced}, -\text{strid}, +\text{cons}\}$ is virtually unheard-of. Still, we don't want to just remove sufficiently complex constraints — what if in some strange language, they turn out to be relevant?

So what we can do instead is have a different prior set for each possible constraint, that takes into account the number of distinct features in the constraint, not counting polar opposites. For instance, the features +voiced and −voiced would count as one feature, whereas +voiced and +cons would count as completely different features. In this way, we still allow for the possibility of strange constraints — if it's the only possible interpretation, then the constraint will become negatively weighted even if the prior strongly advises against it — while also make sure that those constraints are not commonly used.

5.3 Restrictions on Constraint Weights

Unfortunately, the approach presented above is insufficient for learning the correct underlying forms and correct phonological rules for even the simplest examples, such as the Polish data used throughout this paper. To see why, we illustrate with a toy case on the Polish dataset.

First, we need to figure out which constraints we are considering. For this toy example, we will consider a set of six constraints. Consider the following weightings on those six constraints:

constraint	**value**
Faithfulness	3.0
FinalVoiced	−4.0
Under(Club, /klub/)	−2.0
Under(Club, /klup/)	2.0
Under(Sheaf, /snob/)	2.0
Under(Sheaf, /snop/)	−2.0

These constraints are saying that violating faithfulness is actually a good thing, even though that's never true in linguistics. It correctly assigns negative weight to the FinalVoiced constraint, thus making sure that surface forms are less likely to end in a voiced segment. However, it picks exactly the wrong weights for the underlying forms — the incorrect forms, which should be penalized using negative weights, actually have positive weights in this circumstance. But will our learning system actually consider this analysis correct? Well, the items with the highest probability are displayed in the following table:

meaning	**underlying**	**surface**
"club singular"	/klup+Ø/	[klup+Ø]
"club plural"	/klup+i/	[klub+i]
"sheaf singular"	/snob+Ø/	[snop+Ø]
"sheaf plural"	/snob+i/	[snop+i]

Note that each of the surface forms is correct, but the underlying forms are completely wrong, because the analysis says that faithfulness violations are actually a good thing, and should be sought out. As previously mentioned, that is never the correct approach to take linguistically, so what we must do is constrain the faithfulness weight to always be negative. In fact, in general all of the constraints brought in from OT are bad things to have, and should never receive positive weight. So we can stop our learner from considering this as a reasonable possibility by making sure that each of those constraints weights is restricted to being only negative.

5.4 Tuning the Prior

But restricting the markedness and faithfulness constraint weights to be positive is still not sufficient to let you learn the correct underlying forms and phonological rules. Once again, we will work out a toy example on the Polish data set, using only six constraints. Here is a linguistically reasonable analysis:

constraint	**value**
Faithfulness	−3.0
FinalVoiced	−4.0
Under(Club, /klub/)	2.0
Under(Club, /klup/)	−2.0
Under(Sheaf, /snob/)	−2.0
Under(Sheaf, /snop/)	2.0

These weights mean that violating faithfulness is bad, but not as bad as having a final voiced segment. It also assigns positive weight to the underlying forms that are correct, and negative weight to the ones that are incorrect. It is easy to verify that the underlying-surface-alignment triple with the highest probability is correct for each of the words in the Polish data set.

Now consider the following incorrect analysis:

constraint	value
FAITHFULNESS	−4.0
FINALVOICED	−3.0
UNDER(CLUB, /klub/)	1.0
UNDER(CLUB, /klup/)	−1.0
UNDER(SHEAF, /snob/)	−2.0
UNDER(SHEAF, /snop/)	2.0

This analysis still has reasonable constraint weights — nothing is unreasonably high or unreasonably low, and no weight has changed by more than 1.0. It still places positive weight on the correct underlying forms and negative weight on the incorrect underlying forms. But when we calculate the triples with the maximum probability for each meaning, we get the following:

meaning	underlying	surface
"club singular"	/klup+Ø/	[klup+Ø]
"club plural"	/klub+i/	[klub+i]
"sheaf singular"	/snop+Ø/	[snop+Ø]
"sheaf plural"	/snop+i/	[snop+i]

This analysis is not correct: the underlying form of [klup+Ø] should be /klub+Ø/, not /klup+Ø/. This is the sort of analysis that our learning system should rule out. But as currently presented, there is nothing that would rule this out as a better solution, and no constraints that can be placed to ensure that this analysis can be prevented.

Instead, we turn to the prior. What is the reason that this analysis comes up with the incorrect underlying form for "club" in the singular, even though the weights specify the correct underlying form? Well, the difference between the constraint weights on the two potential underlying forms for "club" is 2.0; the faithfulness constraint has weight 4.0. This means that the faithfulness constraint will overwhelm the difference between the underlying form constraints, and therefore just use the surface form as the underlying form.

This behaviour is undesirable. When we select an underlying form for a particular morpheme, we want to use that underlying form every time we try to express that meaning. And we want that choice of underlying form to take precedence over any other constraints, whether they are markedness constraints or faithfulness constraints.

In order to accomplish this, we use the σ_j weights in our prior. We want the faithfulness weights to be restricted to vary much less than the underlying form constraints, so we set:

$$\sigma_{\text{underlying}} \gg \sigma_{\text{markedness}}$$
$$\sigma_{\text{underlying}} \gg \sigma_{\text{faithfulness}}$$

This ensures that the underlying form constraints can have their weights driven up a very large amount before the prior stops it; faithfulness and markedness constraints, on the other hand, can only go up to much smaller values, and therefore their values will be dominated by the underlying form values. This produces the correct results.

5.5 Results

We ran our learner on four data sets: the small Polish, Hungarian, and English datasets given earlier in this paper, and one huge English dataset derived from a subset of the MRC

Psycholinguistic Database [1]. For each one, we used the same constraint set and the same values of σ for the prior.

The faithfulness constraint set used consisted of a single faithfulness constraint for each phonological feature. Each of these constraints had a prior weight of $\sigma = 10$ and an initial value of -1. If these constraints are initialized to zero instead, this causes the optimizer to find an incorrect solution with probability smaller than that of the true solution. This means that the learner is simply having trouble finding the correct solution, not that the most probable weights produce a linguistically incorrect analysis.

The markedness constraint set used consists of $(F \times F) \cup (F \times F \times F)$, where F is the feature set, and could also consist of the word boundary marker #. Markedness constraints had a fairly complex value of σ, ranging from 2.5 to 40 depending on the number of features shared across the feature pattern for the constraints. If ℓ was the number of distinct features in the feature pattern, then $160/4^{\ell}$ was the value for σ. Here are some example values for σ:

feature sequence	**value of σ**
+voiced −voiced +voiced	40
+voiced −voiced #	40
+voiced −cons	10
+voiced −cons +strid	2.5
+voiced #	2.5

All of the markedness constraints had weight initialized to zero.

Underlying form constraints had $\sigma = 1000$ for this experiment, thus ensuring that they could easily outweigh both markedness and faithfulness constraints. All underlying form constraints had their weight initialized to zero as well.

When we optimized using those settings on each of the data sets, we got the following log probabilities:

dataset	**# of data items**	**log probability**
Polish	4	−0.452076
Hungarian	4	−0.394878
English (small)	6	−0.329214
English (large)	782	−10.3192

All but the small English dataset correctly selected the underlying form for each morpheme. However, as previously discussed, the small English dataset does not contain enough information to allow even a human to deduce the correct underlying form, so failure in that case is to be expected.

For the Polish case, the most probable pair for each meaning is as follows:

meaning	**underlying**	**surface**	**probability**
“sheaf singular”	/snop+Ø/	[snop+Ø]	97.9086%
“sheaf plural”	/snop+i/	[snop+i]	92.6709%
“club singular”	/klub+Ø/	[klup+Ø]	79.9767%
“club plural”	/klub+i/	[klub+i]	88.1321%

As you can see, the pairs that are most probable are linguistically correct as well. The constraint weights also reflect the correct analysis. Here is a list of the five most influential constraint weights in effect:

constraint	weight
UNDER(SHEAF, /snop/)	+1.5947
UNDER(SHEAF, /snob/)	−1.5947
UNDER(CLUB, /klub/)	+1.20147
UNDER(CLUB, /klup/)	−1.19076
MARKED(+voiced #)	−1.04923

6 Modifications

6.1 Stochastic Gradient Ascent

In the unsupervised case, the log probability of the entire data set is:

$$\log \Pr(s_1, \ldots, s_n \mid \mathbf{w}) = \sum_{i=1}^{n} \log \Pr(s_i \mid \mathbf{w})$$

Therefore, when we take the derivative with respect to some weight value w_j, we find that:

$$\begin{aligned} \frac{\partial}{\partial w_j} \log \Pr(s_1, \ldots, s_n \mid \mathbf{w}) &= \sum_{i=1}^{n} \frac{\partial}{\partial w_j} \log \Pr(s_i \mid \mathbf{w}) \\ &= \sum_{i=1}^{n} \frac{\frac{\partial}{\partial w_j} \Pr(s_i \mid \mathbf{w})}{\Pr(s_i \mid \mathbf{w})} \end{aligned}$$

However, when calculating this for a very large number of data points ($n \approx 700$), it can get very time intensive to calculate that value for each observed surface form. But in order to perform optimization, we have to calculate that every time we want to take a step to optimize.

Luckily, because of the structure of the function (and therefore, the structure of its derivative), it can be approximated fairly quickly by calculating that same log probability for a small subset of all n datapoints. For a randomly selected subset, the expected value of the gradient is a scaled version of the gradient of the entire dataset. This means that we should be able to calculate the gradient of our smaller subset — which approximates the true gradient — and take a step in that direction to bring us closer to the true solution.

This is a technique called stochastic gradient ascent. The difficulty of doing it lies in finding the correct step size, the correct batch size (the size of the subset of the data points that we use to approximate the gradient), and the correct number of iterations to perform.

Through extensive empirical testing, we have found that a batch size of 1 will work, provided that the step function $s(i)$ giving the size of the step to take at iteration i out of ℓ total iterations resembles the following:

$$s(i) = \frac{\ell - 0.5 * i}{20\ell}$$

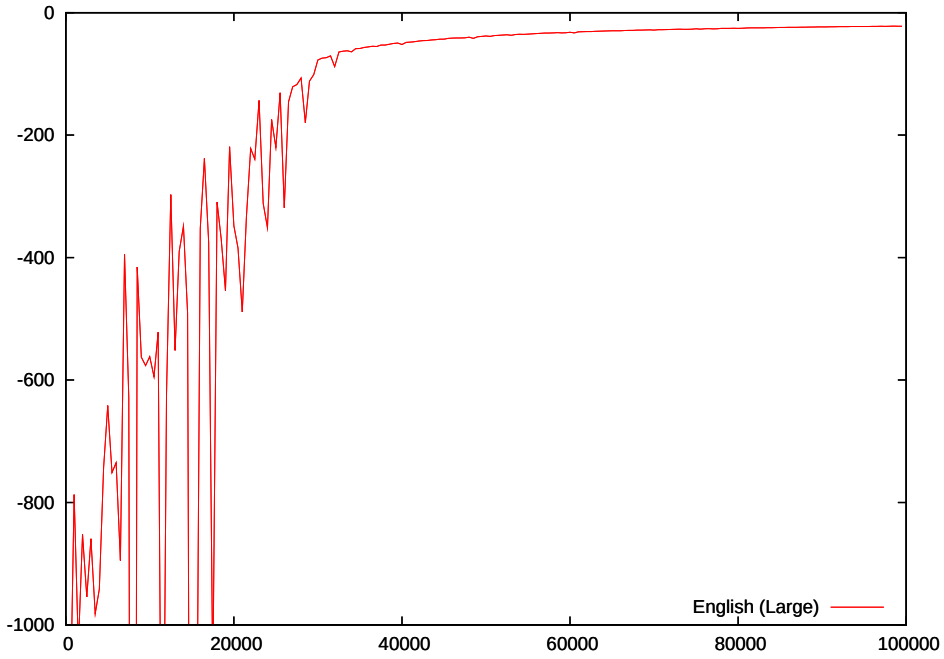
We found that setting $\ell = 100000$ makes this method converge to weights fairly similar to those in the deterministic optimization. The only exception was the large English data set, which converged to linguistically reasonable weights that produced the correct underlying forms, but with lower probability. This seems to be a result of the number of iterations

— increasing the number of iterations increases the probability of the produced English solution, but at the expense of more time.

Here is a graph showing how quickly the three small datasets converge:

Note that we are only showing the first 5000 of 100000 iterations, because these three converge so quickly that the graph would be unreadable if it spanned all 100000 iterations.

And here is a graph showing how slowly the large dataset converges:



6.2 Factoring the Weights

There are three kinds of constraints over which we are learning weights: faithfulness features, underlying form features, and markedness features. Neither underlying form features nor markedness features are affected by the alignment between surface and underlying forms; only the faithfulness features are. Therefore, we can divide up our features into three sets:

$$\begin{aligned}\mathbf{f}_m &: \text{markedness features}\\ \mathbf{f}_u &: \text{underlying form features}\\ \mathbf{f}_f &: \text{faithfulness features}\end{aligned}$$

For each of these sets, we have the associated weight vectors $\mathbf{w}_m, \mathbf{w}_u, \mathbf{w}_f$. With this new formulation, the score associated with a single pair of words u, s and a particular alignment a is now:

$$p(u, s, a, \mathbf{w}) = e^{\mathbf{w}_m \cdot \mathbf{f}_m(u,s,a) + \mathbf{w}_u \cdot \mathbf{f}_u(u,s,a) + \mathbf{w}_f \cdot \mathbf{f}_f(u,s,a)}$$

Because $\mathbf{f}_m$ only depends on s and $\mathbf{f}_u$ only depends on u, we can write this as:

$$p(u, s, a, \mathbf{w}) = e^{\mathbf{w}_m \cdot \mathbf{f}_m(s) + \mathbf{w}_u \cdot \mathbf{f}_u(u) + \mathbf{w}_f \cdot \mathbf{f}_f(u,s,a)}$$

Now, say that we want to figure out the probability of a pair u, s, regardless of our choice of alignment. If we assume for the moment that all alignments are equally likely, then we we can calculate the score associated with the pair u, s as:

$$\begin{aligned}p(u, s, \mathbf{w}) &= \sum_{\text{alignment } a} e^{\mathbf{w}_m \cdot \mathbf{f}_m(s) + \mathbf{w}_u \cdot \mathbf{f}_u(u) + \mathbf{w}_f \cdot \mathbf{f}_f(u,s,a)}\\ &= \sum_{\text{alignment } a} e^{\mathbf{w}_m \cdot \mathbf{f}_m(s)} e^{\mathbf{w}_u \cdot \mathbf{f}_u(u)} e^{\mathbf{w}_f \cdot \mathbf{f}_f(u,s,a)}\\ &= e^{\mathbf{w}_m \cdot \mathbf{f}_m(s)} e^{\mathbf{w}_u \cdot \mathbf{f}_u(u)} \sum_{\text{alignment } a} e^{\mathbf{w}_f \cdot \mathbf{f}_f(u,s,a)}\end{aligned}$$

So if we can find a way to efficiently compute that sum over all alignments, then we can efficiently compute the total score associated with $p(u, s, \mathbf{w})$. We define:

$$b(u, s, \mathbf{w}) = \sum_{\text{alignment } a} e^{\mathbf{w}_f \cdot \mathbf{f}_f(u,s,a)}$$

How do we calculate b? Just as when calculating the edit distance between two words, the set of all possible alignments of the characters consists of a sequence of decisions between three options:

- align the first unaligned character of u with a null character, and align the rest of u with the unaligned characters in s
- align the first unaligned character of s with a null character, and align the rest of s with the unaligned characters in u
- align the first unaligned character of u with the first unaligned character of s, and align the rest of u with the rest of s

Therefore, we can write a recurrence relation for b as follows:

$$\begin{aligned} b(u, s, \mathbf{w}) = & e^{\mathbf{w}_f \cdot \mathbf{f}_f(u[0], s[0])} \cdot b(u[1:], s[1:], \mathbf{w}) \\ & + e^{\mathbf{w}_f \cdot \mathbf{f}_f(\emptyset, s[0])} \cdot b(u, s[1:], \mathbf{w}) \\ & + e^{\mathbf{w}_f \cdot \mathbf{f}_f(u[0], \emptyset)} \cdot b(u[1:], s, \mathbf{w}) \end{aligned}$$

The base cases for this recurrence relation are:

$$\begin{aligned} b(u, \epsilon, \mathbf{w}) &= e^{\mathbf{w}_f \cdot \mathbf{f}_f(u[0], \emptyset)} \cdot b(u[1:], \epsilon, \mathbf{w}) \\ b(\epsilon, s, \mathbf{w}) &= e^{\mathbf{w}_f \cdot \mathbf{f}_f(\emptyset, s[0])} \cdot b(\epsilon, s[1:], \mathbf{w}) \\ b(\epsilon, \epsilon, \mathbf{w}) &= 1 \end{aligned}$$

where ϵ is the empty string. Note that this recurrence relation is ideal for dynamic programming, and can be calculated in $O(n^2)$ time using a table to store previously calculated values. An entry at (i, j) corresponds to the value of b associated with the substrings $u[i:]$ and $s[j:]$. Then the above base cases and recurrence relations can be used to calculate the values for b for u and s.

Thus, we have a way to efficiently calculate the unnormalized probability associated with each underlying-surface pair. However, this is not sufficient — in order to efficiently optimize this function, we need to be able to efficiently compute the derivative of this value.

What happens when we take the derivative with respect to some w_i? Then we get the following:

$$\begin{aligned} b'(u, s) = & f_i(u[0], s[0]) \cdot e^{f_f(u[0], s[0]) \cdot w_f} \cdot b(u[1:], s[1:]) \\ & + e^{f_f(u[0], s[0]) \cdot w_f} \cdot b'(u[1:], s[1:]) \\ & + f_i(\emptyset, s[0]) \cdot e^{f_f(\emptyset, s[0]) \cdot w_f} \cdot b(u, s[1:]) \\ & + e^{f_f(\emptyset, s[0]) \cdot w_f} \cdot b'(u, s[1:]) \\ & + f_i(u[0], \emptyset) \cdot e^{f_f(u[0], \emptyset) \cdot w_f} \cdot b(u[1:], s) \\ & + e^{f_f(u[0], \emptyset) \cdot w_f} \cdot b'(u[1:], s) \end{aligned}$$

Note that, once again, this can be calculated efficiently using dynamic programming: given the table for each value of b, we need only compute n^2 values of b'. So using this derivative for b, we can calculate the derivative of the overall score function p, which can then be used to calculate the derivative of the log likelihood.

When we introduce these new concepts into our old framework, the results are unchanged. However, this experiment is a proof of concept for ideas that will be discussed in the next section.

7 Future Work

7.1 Why We Need Sampling

So far, we have deliberately restricted ourselves to a limited set of alternative morphemes. But what happens when we lift that restriction? Once we allow for arbitrary character

alternations, and arbitrary insertions and deletions, we can no longer count on the small size of our set of alternate morpheme forms. In fact, unless there turns out to be an efficient way of summing over an exponential number of forms, we will not be able to compute the exact derivative, or even the exact value of our probability function.

We have considered two possible approaches for this, both based on existing concepts: stochastic EM [3], and the wake-sleep algorithm [5]. Both approaches require us to have some way of sampling underlying and surface forms, with probability at least roughly proportional to their unnormalized probability given weights $\mathbf{w}$.

7.2 Sampling Underlying Forms

Assume that we have some surface form s for which we are trying to find candidate underlying forms of length n. That is, we are trying to sample underlying forms u so that the probability of selecting u is proportional to the unnormalized weight of the underlying-surface pair (u, s). For the moment, we will ignore the underlying form constraints $\mathbf{f}_u$, and focus on trying to generate underlying forms based only on our faithfulness constraints $\mathbf{f}_m$. Note that the markedness constraints $\mathbf{f}_m$ don't affect our choice of underlying form, because they depend only on the surface form, which is fixed.

In order to perform sampling, we need to know the total unnormalized weight associated with all candidate underlying forms of length n. Let $v(s, n, \mathbf{w})$ be that weight. Just as in the case for our factored faithfulness features, we know that one of the following three possibilities must occur:

- the first character of the underlying form is aligned with the first character of s, thus leaving the rest of s to be aligned with $n-1$ characters of the underlying form
- the first character of the underlying form is not aligned to any characters in s, thus leaving all of s to be aligned with $n-1$ characters of the underlying form
- the first character of the surface form is not aligned to any characters of the underlying form (whatever it is), thus leaving the rest of s to be aligned with n characters

From those three possibilities, we can figure out the recurrence relation:

$$\begin{aligned} v(s, n, \mathbf{w}) = &\sum_{c \in C} e^{\mathbf{w}_f \cdot \mathbf{f}_f(c, s[0])} \cdot v(s[1:], n-1, \mathbf{w}) \\ &+ \sum_{c \in C} e^{\mathbf{w}_f \cdot \mathbf{f}_f(c, \emptyset)} \cdot v(s, n-1, \mathbf{w}) \\ &+ e^{\mathbf{w}_f \cdot \mathbf{f}_f(\emptyset, s[0])} \cdot v(s[1:], n, \mathbf{w}) \end{aligned}$$

Where C is the set of non-null characters. The base cases for this recurrence relation are:

$$\begin{aligned} v(s, 0, \mathbf{w}) &= e^{\mathbf{w}_f \cdot \mathbf{f}_f(\emptyset, s[0])} \cdot v(s[1:], 0, \mathbf{w}) \\ v(\epsilon, n, \mathbf{w}) &= \sum_{c \in C} e^{\mathbf{w}_f \cdot \mathbf{f}_f(c, \emptyset)} \cdot v(\epsilon, n-1, \mathbf{w}) \\ v(\epsilon, 0, \mathbf{w}) &= 1 \end{aligned}$$

Note that, once again, we can construct a dynamic program to calculate this total value.

Once we have this value, how do we use it to sample from the set of potential underlying forms? Each item in the sum of the recurrence relation represents a possible choice to take; the probability of that choice is proportional to the weight of the value contributed to $v(s, n, \mathbf{w})$. This means that we can sample underlying forms with the correct probability by moving backwards through the table, picking transitions with probability proportional to our weight.

To be more specific, if we are given a table $T(\cdot, \cdot)$ with the property that $T(i, j) = v(s[i :], j, \mathbf{w})$, we can use the following pseudocode to help sample underlying forms:

define SAMPLEU(i, j):
    if $i = 0$ and $j = 0$:
        return an empty string

    pick a random real number r from 0 to $T(i, j)$
    if $i > 0$ and $j > 0$:
        for each $c \in C$:
            set $r = r - e^{\mathbf{w}_f \cdot \mathbf{f}_f(c, s[i-1])} \cdot T(i-1, j-1)$
            if $r \leq 0$:
                return c concatenated with SAMPLEU$(i-1, j-1)$

    if $i > 0$:
        set $r = r - e^{\mathbf{w}_f \cdot \mathbf{f}_f(\emptyset, s[i-1])} \cdot T(i-1, j)$
        if $r \leq 0$:
            return SAMPLEU$(i-1, j)$

    if $j > 0$: for each $c \in C$:
            set $r = r - e^{\mathbf{w}_f \cdot \mathbf{f}_f(c, \emptyset)} \cdot T(i, j-1)$
            if $r \leq 0$:
                return c concatenated with SAMPLEU$(i, j-1)$

We can sample an underlying form of length n by calling SAMPLEU with the length of s as its first argument, and n as its second.

This means that we can sample underlying forms of a fixed length based only on our faithfulness constraints. But how can we then sample the length? Well, it would be nice to be able to use the different $v(s, n, \mathbf{w})$ values to determine the weight associated with each possible length n, and then sample the length based on those weights.

However, consider what happens if your faithfulness constraints have fairly low weight. With weights sufficiently close to zero, then the following equation will hold:

$$\sum_{c \in C} e^{\mathbf{w}_f \cdot \mathbf{f}_f(c, \emptyset)} > 1$$

When that equation holds, each value of $v(s, n, \mathbf{w})$ will be at least that value times the value of $v(s, n-1, \mathbf{w})$. This means that the value of $v(s, n, \mathbf{w})$ increases exponentially in n, and so any sampler would select underlying forms that are clearly too long.

There are two ways that we can approach this: we can either try to ensure that the above equation will never hold, and therefore that v will always crest at around the right

length; or we can create a new way of sampling lengths, based on our knowledge of what the length should be. The first is fairly infeasible; although the above equation is sufficient to cause $v(s, n, \mathbf{w})$ to increase exponentially in n, it is not the only equation that would cause it to have that problem. Restricting the weights would be nearly impossible.

What we really want is a system for sampling underlying forms. But we are already not sampling with perfect accuracy; we have decided to ignore the weights $\mathbf{f}_u$. So how can we handle these problems of length and of $\mathbf{f}_u$? The answer is sampling-importance-resampling [2].

The idea behind sampling-importance-resampling, and related algorithms, is this: we have a distribution from which we are trying to generate samples. Unfortunately, for one reason or another, it is hard to generate samples from that distribution. However, the unnormalized probabilities for that distribution are easy to compute. So instead of trying to directly sample from that distribution, we create another distribution, called the **proposal distribution**, for which it is easy to generate samples and calculate their probabilities. We can then sample from the proposal distribution, and then either reject, reweight, or otherwise resample based on the unnormalized probabilities from the distribution that we are ultimately trying to sample from.

So if we can create such a proposal distribution for the problem at hand, we will be able to sample underlying forms. This means that we don't need the length sampler to be exactly correct, as long as it generates a lot of fairly plausible samples, with probability that we can actually compute. A simple Gaussian over lengths suffices.

Now that we have a way to sample underlying forms based only on our faithfulness constraints, how can we adjust our sampling to account for the underlying form weights given by our constraint functions $\mathbf{f}_u$? Once again, we can rely on importance resampling to handle this. We have a finite number of morphemes that have non-zero constraint weight, so sampling from those morphemes is possible. Thus we have one way to sample using $\mathbf{f}_u$, and one way to sample using $\mathbf{f}_f$. We can combine the two by performing a coin flip to determine which distribution we sample from. Then the probability of an underlying form is simply the average of its probability under the $\mathbf{f}_u$ distribution and its probability under the $\mathbf{f}_f$ distribution. This gives us a good proposal distribution that can then be reweighted and resampled to get a set of sampled underlying forms, each of which was sampled with probability proportional to its unnormalized weight.

7.3 Sampling Surface Forms

Once we know how to sample underlying forms for a given surface form, how can we then use those underlying forms to sample surface forms for our contrast set? In other words, say that we're given some underlying form u. How can we then sample surface forms s such that the probability of sampling s is proportional to the unnormalized probability of (u, s)? In this case, the unnormalized probability depends only on $\mathbf{f}_f$ and $\mathbf{f}_m$, since the underlying form is constant.

As previously noted, $\mathbf{f}_f$ acts on an individual character level. If we were only trying to sample using $\mathbf{f}_f$, we could sample in a way very similar to sampling underlying forms. Unfortunately, the same way of splitting up the probabilities into ones depending on $\mathbf{f}_f$ and ones depending on $\mathbf{f}_u$ is not possible in this case: $\mathbf{f}_m$, unlike $\mathbf{f}_u$, does not store entire subsections of words.

However, the nice thing about $\mathbf{f}_m$ is that each of the features only depends on a series of at most three characters. This suggests that we may be able to have another dynamic programming approach, albeit with many more cells in the table to store the most recent two characters selected.

Just as before, suppose we are given some u and some length n. Let the function $v(u, n, c_1, c_2, \mathbf{w})$ be the total unnormalized weight of all pairs between u and some surface form of length n such that the first two characters of the surface form are c_1 and c_2 respectively. Then we can write the following recurrence relation for v:

$$\begin{aligned} v(u, n, c_1, c_2, \mathbf{w}) = &\sum_{c_3 \in C} e^{\mathbf{w}_f \cdot \mathbf{f}_f(u[0], c_1)} e^{\mathbf{w}_m \cdot \mathbf{f}_m(c_1, c_2, c_3)} v(u[1:], n-1, c_2, c_3, \mathbf{w}) \\ &+ \sum_{c_3 \in C} e^{\mathbf{w}_f \cdot \mathbf{f}_f(\emptyset, c_1)} e^{\mathbf{w}_m \cdot \mathbf{f}_m(c_1, c_2, c_3)} v(u, n-1, c_2, c_3, \mathbf{w}) \\ &+ e^{\mathbf{w}_f \cdot \mathbf{f}_f(u[0], \emptyset)} \cdot v(u[1:], n, c_1, c_2, \mathbf{w}) \end{aligned}$$

Note that this is not quite accurate: this only works if all constraints are three-feature sequences, otherwise they will be double-counted. However, in an actual implementation, it is easy enough to account for this double-counting.

Just as in the case with sampling underlying forms, we can use this to sample our surface forms by making decisions based on each component of the sum, and stepping through the table. This gives us a complete system for sampling underlying and surface forms, and therefore a system for solving this on arbitrary problems.

References

[1] UWA Psychology: MRC Psycholinguistic Database. `http://www.psy.uwa.edu.au/mrcdatabase/uwa_mrc.htm`.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.

[3] Alexandre Bouchard-Côté, Percy Liang, Thomas L. Griffiths, and Dan Klein. A probabilistic approach to language change. In *NIPS*. MIT Press, 2007.

[4] Sharon Goldwater and Mark Johnson. Learning OT constraint rankings using a Maximum Entropy model. *Proceedings of the Stockholm Workshop on Variation within Optimality Theory*, pages 111–120, April 2003.

[5] Geoff Hinton, Peter Dayan, Brendan Frey, and Radford Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1160, 1995.

[6] Géraldine Legendre, Yoshiro Miyata, and Paul Smolensky. Can connectionism contribute to syntax? Harmonic Grammar, with an application. Technical Report #90–12, University of Colorado at Boulder, 1990.

[7] Alan Prince and Paul Smolensky. Optimality Theory: Constraint interaction in generative grammar. Manuscript, Rutgers University and University of Colorado at Boulder. Available at ROA, 1993.

[8] Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 354–362, June 2005.

[9] Bruce Tesar and Paul Smolensky. Learnability in Optimality Theory (short version). Technical Report JHU-CogSci-96-2, Johns Hopkins University, October 1996.