High-Level Explanation of Variational Inference

by Jason Eisner (2011)

[This was a long email to my reading group in January 2011. See the link below for further reading.]

By popular demand, here is a high-level explanation of variational inference, to read before our unit in the NLP reading group. This should be easy reading since I've left out almost all the math. So please do spend 30 minutes reading it, if only to make it worth my while to have written it. :-)

I threw in some background on what we are trying to do with inference in general. Those parts will be old hat to some of you. The variational stuff is near the start and end of the message.

The part-of-speech tagging example at the end is a good one to think about whenever you are trying to remember how variational inference works. The setting is familiar in NLP, and it illustrates all the important points.

Some people may find this page more valuable *after* they have learned one or more specific variational methods, such as the mean-field approximation, which is used in variational Bayes and elsewhere. In general, this page assumes familiarity with models like Markov Random Fields.

-cheers, jason

Overview

Problem: (1) Given an input x, the posterior probability distribution over *outputs* y is too complicated to work with. Or (2) Given a training corpus x, the posterior probability distribution over *parameters* y is too complicated to work with.

Solution: Approximate that complicated posterior p(y | x) with a simpler distribution q(y).

Typically, q makes more independence assumptions than p. This is more or less "okay" because q only has to model y for the particular x that we actually saw, not the full relation between x and y.

Which simpler distribution q? Well, you define a whole family Q of distributions that would be computationally easy to work with. Then you pick the q in Q that best approximates the posterior (under some measure of "best").

Terminology

Case (1) above leads to variational decoding, which can be used within variational EM training. Case (2) leads to variational Bayes, which combines training and decoding into a single optimization problem.

Some examples of variational methods include the mean-field approximation, loopy belief propagation, tree-reweighted belief propagation, and expectation propagation.

q is called the variational approximation to the posterior. The term variational is used because you pick the **best** q in Q -- the term derives from the "calculus of variations," which deals with optimization problems that pick the best *function* (in this case, a distribution q). A particular q in Q is specified by setting some variational parameters -- the knobs on Q that you can turn to get a good approximation.

Why It's Not Trivial

Notice that the method is not the same as just throwing away the complex model p(x,y) and using a simpler one q(x,y) in its place. We never define anything like q(x,y), only q(y) for a given input x. The complex model p is still used to define what we're trying to approximate by q(y), namely p(y | x), which may differ for each input x.

A similar approach would be to do MCMC sampling of y values from p(y | x) and then train a simpler model q(y) on those samples. Even though q is simple, it may be able to adequately model the variation in those samples for a given, typical x.

But in contrast to the above, variational methods don't require any sampling, so they are fast and deterministic. They achieve this by carefully choosing the objective function that decides whether q is a good approximation to p, and by exploiting the simple structure of q.

Below, I'll give a little background on inference, and then a couple of simple NLP examples of variational inference.

Background on Probabilistic Inference

Background: Your Probability Model

The general setting for all inference problems is that you're working with a joint probability distribution over a bunch of variables. (For example, a graphical model.) The variables might be discrete, continuous, structured, whatever.

You *know* the probability distribution and you have an efficient function to compute it. That is, for any configuration defined by an assignment of values to the random variables, you can compute the probability of that configuration.

More precisely, you only have to be able to compute an unnormalized probability (i.e., the probability times some unknown constant Z). That's helpful because it lets you define MRFs and CRFs and such.

You might protest that you *don't* know the distribution -- that's why you have to train the model. However, read on! We'll regard that as just part of inference.

Background: Input, Output, And Nuisance Variables

You observe some of the variables (input to your system). You want to infer the values of some of the other variables (output of your system).

Of course, this inference is to be conditioned on the observed input: you want to know the posterior distribution p(output | input). But that's proportional to p(input, output), so the function for computing *unnormalized* probabilities is the same.

In addition to input and output variables, there may be nuisance variables that are useful in defining the relation between input and output. Examples include alignments and clusters: Google Translate may reconstruct these internally when translating a sentence, but they are ultimately not part of the input or output. So p(input,output) is defined as $\sum_{\text{nuisance}}$ p(input,nuisance,output).

Think now about continuous parameters of the system, like transition probabilities or mixture weights or grammar probabilities. These too are usually nuisance variables, since usually they are not part of the input or output! The system merely guesses what they are (e.g., "training") in order to help map input to output. So they really are no different from alignments.

Remark: The previous paragraph adopted a Bayesian perspective, where the parameters are regarded as just more variables in your model. So now you have a joint distribution over input, output, parameters, and other nuisance variables. This joint distribution is typically defined to include a prior over the parameters (maybe a trivial uniform prior, or maybe a prior that combats overfitting by encouraging simplicity).

Advanced remark: I'll assume that the model has a fixed number of variables. That's not too much of a limitation, since some of these variables could be unboundedly complicated (a sequence, tree, grammar, distribution, function, etc.). However, you'll sometimes see inference techniques that change the number of variables before inference starts (collapsed, block, and auxiliary-variable methods) or even during inference (e.g., reversible-jump MCMC).

Background: Inference Methods

To design an inference method, the first and most important step is to decide how you will handle the uncertainty in each variable.

Input: For input variables, there is no uncertainty.

Output: For output variables, it depends on who your customer is. Are you being asked to report the whole posterior distribution over values (given the input)? Samples from that distribution? The mode of that distribution (i.e., the single most likely value)? The minimum-risk estimate (i.e., the estimate with lowest expected loss under the posterior distribution)?

And if there are several output variables, do you report these things about the joint posterior distribution over all of them? Or do you report separately about each output variable, treating the others temporarily as nuisance variables? (This is like doing several separate tasks with the input.)

Nuisance: For nuisance variables, the right thing to do is to sum over their possible values (integrate them out or marginalize them out). However, as we'll see below, for purely computational reasons, you might need to sum approximately, either by sampling or by variational methods.

Another traditional approximation is to maximize over the nuisance variables. Some particular examples of this strategy have special names: EM training (maximize over parameters), "hard EM" or "Viterbi EM" training (maximize over both parameters and hidden data), and MAP decoding (maximize over hidden data for given parameters). But maximizing over variables V,W,... can be regarded as a special case of a variational method (where Q is limited so that each distribution q in Q puts all its mass on some single assignment to V,W, so that picking the best q will pick the best single assignment). It's probably not the best choice of variational method, since you can usually use a larger Q (which provides better choices of q) at about the same computational cost.

Background: Intractable Coupling

The problem with inference is computation. Of course it's very easy to *define* the result you want mathematically, e.g.,

$\text{argmax}_{\{\text{assignment to output variables}\}} \sum_{\{\text{assignment to nuisance variables}\}} p(\text{output,nuisance,input})$

This expression happens to define what is called the marginal MAP output. However, computing this output is quite another story! The above expression maximizes and sums over exponentially many assignments, or even infinitely many in the case of continuous variables.

Sometimes there is a nice efficient way to compute such expressions, by exploiting properties of conjugate priors, or by using dynamic programming or other combinatorial optimization techniques. But sometimes there isn't any efficient way.

The trouble is that these variables aren't independent: they covary in complicated ways. To figure out the distribution over one variable, you have to look at its interactions with the other variables, including nuisance variables. We speak of intractable coupling when these interactions make it computationally intractable to find the marginal distribution of some variable.

A good general solution to this problem is MCMC. You can often design a method for sampling from the probability distribution. This handles the coupling between variables by letting the variables evolve in a co-dependent way. (Gibbs sampling is the most basic technique. Fancier samplers may exploit conjugacy or dynamic programming as subroutines, e.g., for collapsed Gibbs, block Gibbs, and Metropolis-Hastings proposals.)

Motivation: Variational Methods

Unfortunately, MCMC can be slow to get accurate answers. Sure, the longer you run it, the more accurate your sampling distribution and the more samples you can take. If you have infinite time, it's perfectly accurate. But brute-force summations are perfectly accurate too and they'd only take finite time.

Often you want to say, "Look, it can't be all that hard! The different variables may covary in the posterior distribution, but most of them don't interact all that much ..."

For one thing, some of the variables don't even *vary* very much in the posterior distribution p(y | x) -- the input x pretty much tells you what they must be. So they certainly can't *covary* much. (That is, many variables have low entropy under the posterior distribution, which implies that they also have low mutual information with other variables.)

Even when variables do vary a lot, we may be able to greatly speed up inference by ignoring some of the interaction, and pretending that the variables are just behaving that way "on their own." The mean-field method throws away all of the interactions. Other methods keep as many interactions as they can handle efficiently.

Examples

Hand-Waving Example: Speech IR

Consider IR over speech:

Input: The spoken document.
Output: A relevance score (with respect to a query).
Nuisance variable: The text transcription of the document. This is related to the input using an ASR system.

What one should do in principle:
For each individual path through the ASR speech lattice, evaluate how relevant that transcription is.
Average over the paths in proportion to their probability to get an *expected* relevance score for the document. (There are exponentially many paths, but the average may be approximated by sampling paths.)

What people do in practice:
Relevance depends on the bag of words along the true path.
Since the true path is uncertain, just get the *expected* bag of words, and compute the relevance of that.
The expected bag of words is a vector of fractional expected counts, obtained by running forward-backward over the lattice. For example, maybe the word "pen" is in the document an expected 0.4 times.

In other words, people compute the *relevance of the expectation* instead of the *expected relevance*. This is essentially a variational approximation, because it pretends that the count of "pen" is independent of the count of other word types. That's an approximation! Suppose the lattice puts "pencil" (0.6) in competition with "pen sill" (0.4). So if "pen" occurs, then "sill" probably occurs as well, and "pencil" doesn't.

Since the approximation ignores those interactions, it would incorrectly judge the document as being 0.4*0.6 likely to match the query "pen AND pencil" (the true probability is 0), and as being only 0.4*0.4 likely to match "pen AND sill" (the true probability is 0.4).

Even so, the approximation is pretty good for most queries, since the interactions are very weak for any word tokens that are separated by more than a few seconds.

Formally, we're approximating the lattice using a simple family Q of distributions over bags of words: in these distributions, p("pen") at one position is independent of p("sill") at the next position. What the forward-backward algorithm is doing is to find the best approximation q in this family (for some sense of "best").

Then, we can compute the expected relevance under this approximate distribution q. That's much easier than computing it under the true distribution over paths! Under the approximate distribution, "pen" appears with probability 0.4 and "pencil" appears *independently* with probability 0.6. So the probability that a document drawn from this distribution would match the boolean query "pen AND pencil" is 0.4*0.6, as noted above. Or if you prefer vector space queries, the expected TF-IDF dot product of such a document with the bag-of-words query "pen pencil" would be 0.4 * 1 * (IDF weight of "pen") + 0.6 * 1 * (IDF weight of "pencil").

In short, once we throw away the interactions between different terms, the expected relevance rearranges easily into something that is easy to compute. In the vector space case, the expected relevance rearranges into the relevance of the expectation -- that is, apply the usual dot product relevance formula to a fractional vector representing the expected bag of words -- which was exactly the intuition at the start of the section.

More Formal Example: Variational Bayes For HMMs

Consider HMM part of speech tagging:

$p(\theta,\text{tags},\text{words}) = p(\theta) * p(\text{tags} \mid \theta) * p(\text{words} \mid \text{tags},\theta)$

where θ is the unknown parameter vector of the HMM. The term $p(\theta)$ represents a prior distribution over θ.

Let's take an unsupervised setting: we've observed the words (input), and we want to infer the tags (output), while averaging over the uncertainty about θ (nuisance):

$p(\text{tags} \mid \text{words}) = (1/Z(\text{words})) * \sum_\theta p(\theta,\text{tags},\text{words})$

Why is this so hard? Where's the intractable coupling? Well, if θ were observed , we could just run forward-backward. Forward-backward is fast: it exploits the conditional independence guaranteed by the Markov property. It also exploits the independence of sentences from one another (that's what lets us run forward-backward on one sentence at a time).

But θ is not observed in this case! Remember that if a variable in a graphical model is unobserved, then its children become interdependent (because observing one child tells you something about the parent, which tells you something about the other children). In this case, when θ is unobserved, we lose the independence assumptions mentioned above. E.g., our tagging of one sentence is no longer independent of our tagging of the next sentence, because the two taggings have to agree on some plausible θ that would make both taggings plausible at once.

(In fact, you can see exactly *how* the taggings become interdependent. Consider the case where the prior p(θ) is defined using Dirichlet distributions for $p(\text{tag}_i \mid \text{tag}_{i-1})$ and $p(\text{word}_i \mid \text{tag}_i)$. Then collapsing out θ leaves us with a Polya urn process (the finite version of a Chinese restaurant process) in which the probability of using a particular transition or emission goes up in relation to the number of times it's been used already.)

EM tries to get around this problem by fixing θ to a particular value whenever it runs forward-backward. Unfortunately, EM is only maximizing over θ rather than summing over θ.

To approximately sum over θ, as we want, we'll use variational Bayes. This will approximate the posterior by one in which different sentences are independent again, and in which the tags within a sentence are conditionally independent again in that they satisfy a Markov property.

For pedagogical reasons, I'll add one more variable, so that we still have an EM-style learning problem even though we're summing over θ. Let's suppose p has some implicit hyperparameters α (for example, which define Dirichlet concentration parameters in the prior p(θ)). That gives us something to learn: although we're summing over θ, let's maximize over α.

Given observed words, we want to adjust α to increase the log likelihood of our observations, log p(words). However, we will settle for increasing a lower bound.

The key trick -- used in many though not all variational methods -- is to write the true log-likelihood as the log of an expectation under some q. We can then get a lower bound via Jensen's inequality, which tells us that log expectation >= expectation log (since log is concave and q is a distribution). For *any* p and q,

$$\begin{aligned}
&\log p(\text{words}) \\
&= \log \textstyle\sum_{\{\theta,\text{tags}\}} p(\theta,\text{tags},\text{words}) \\
&= \log \textstyle\sum_{\{\theta,\text{tags}\}} q(\theta,\text{tags})\ (p(\theta,\text{tags},\text{words})/q(\theta,\text{tags})) \\
&= \log E_q\ (p(\theta,\text{tags},\text{words})/q(\theta,\text{tags})) \\
\\
&>= E_q \log\ (p(\theta,\text{tags},\text{words})/q(\theta,\text{tags})) \quad \text{by Jensen's inequality} \\
&= E_q \log p(\theta,\text{tags},\text{words}) - E_q \log q(\theta,\text{tags})
\end{aligned}$$

The right-hand side is called the variational lower bound. We can increase it by adjusting p (via the parameters α) and q (via the variational parameters provided by the family Q). Our goal will be to jointly find p and q that make it as large as possible, on the twofold grounds of

(a) **Approximate learning** (choosing p's parameters α). The left-hand side is what we'd really like to maximize. If we've found (p,q) that make the right-hand side (the variational lower bound) large, then the p we've found makes the left-hand side even larger. In short, we've found a good α. (Warning: This justification is a bit hand-wavy, since the right-hand side may be -5928349.23, and all that tells us about the left-hand side is that it is between -5928349.23 and 0. What we're really hoping is that improving the right-hand side tends to improve the left-hand side, but there's no guarantee that the opposite doesn't happen.)

(b) **Approximate inference** (choosing q's parameters). Once we've reached a local maximum (p,q), we can query that q to find out (approximately) about the hidden tags and θ parameters under that p. This is because at any local maximum (p,q), we cannot improve q further for that p, so q(θ,tags) is the best possible approximation (within Q) of the posterior p(θ,tags | words). We query the approximate distribution q only because the actual distribution p is too complicated to query. We'll see below why this is true.

In general, the variational lower bound is a non-convex function, so we will only be able to find a local maximum. Hope you're not disappointed.

Variants

Here are a few variants that may help you think about the framework:

α	θ	tags	optimization problem	name of (approximation) method
1. maximize	sum	sum	empirical Bayes	variational EM
2. given	sum	sum	Bayesian posterior inference	variational Bayes
3. given	maximize	sum	MAP estimation	(variational) EM
4. given	given	sum	marginal inference	(variational) inference
5. given	given	max	MAP decoding	Viterbi algorithm or beam search

In the final column, EM means we're maximizing over some unknowns (hyperparameters α or parameters θ). More precisely, EM refers to a specific kind of "alternating optimization" algorithm for this. Bayes means that we're instead doing the proper Bayesian thing and summing (integrating) over all unknowns. Variational says we're willing to use an approximation.

Case 1. is the variational EM setting above, where we are trying to maximize p(words) = $\sum_{\{\theta,\text{tags}\}}$ p(θ,tags,words) with respect to α, summing over some nuisance variables. This maximization objective is dubbed empirical Bayes because it's not quite Bayesian: instead of stating a distribution over θ that describes our *true* prior belief, we lack the courage to commit to such a belief, so we sneakily tune α to find an "empirical prior" that places mass on θ values that were likely to have generated our data. Of course, variational EM only manages to optimize a lower bound on the empirical Bayes objective. In optimizing this bound, we are interested in both (a) and (b) above. **Where not otherwise stated, this webpage focuses on Case 1.**

Case 2. Earlier I introduced α "for pedagogical reasons" so we'd have a hyperparameter to optimize. Suppose instead I'd just stated α=1, which defines a specific, fixed prior over θ (presumably representing our true Bayesian beliefs). Then we'd be summing over all unknowns, namely the parameters θ, which is a proper Bayesian method—or at least a variational approximation, dubbed variational Bayes. Unlike case 1, when we optimize the variational lower bound, p(θ,tags,words) is already fixed. We only have to optimize q(θ,tags) to approximate p(θ,tags | words). That is, (b) above is the main goal since there is no longer an α for (a) to learn.

Case 3. Let's simplify further. Suppose we continue to fix α=1, but now we maximize over θ instead of summing. This is traditionally called maximum *a posteriori* (MAP) estimation since it finds the parameters θ with the highest posterior probability, $p(\theta \mid \text{words}) \propto p(\theta,\text{words}) = \sum_{\text{tags}} p(\theta,\text{tags},\text{words})$.

Formally, this looks the same as case 1—a maximization of a sum. Indeed, you can regard case 1's empirical Bayes objective as MAP estimation too. (More specifically, case 1 was maximum likelihood estimation, since we assumed no prior or equivalently a flat prior over α.) To be precise, case 1 finds the best parameters α of a *hierarchical* model that generates θ from α and then generates the tags and words given θ. Case 3 merely gets rid of the top level of this hierarchical model, so it finds the best θ instead of the best α. But abstractly it's the same setting. Just as in case 1, our variational objective in case 3 is a lower bound on log p(words), and our definition of p(words) still sums over something, this time just the tags. Our model is now p(tags,words | θ), and our variational distribution is now just q(tags), which we tune to approximate p(tags | θ,words). We are again optimizing both p and q, i.e., our goals are again both (a) and (b).

So the method is again called variational EM, as in case 1. But why *variational* EM? We don't have a variational distribution over θ anymore. You'll protest that case 3 is just the *ordinary* EM setting! Or more precisely, MAP-EM, the trivial extension of EM that seeks the MAP estimate of θ (instead of just the maximum likelihood estimate) by including p(θ) at the M step.

The reason it's still variational EM in general is that we do still have a variational distribution q(tags). Now, if this distribution is unrestricted, so that we can set it to *exactly* equal p(tags | θ,words), then the variational gap shrinks to 0. Then the alternating optimization algorithm for variational EM reduces to the ordinary EM algorithm.

That is the case where you can exactly compute the objective $\sum_{\text{tags}} p(\theta,\text{tags},\text{words})$ rather than approximating it with a lower bound. You can indeed do this for an HMM. But more generally, you might have a fancy tagging model where this sum is intractable: the model is not an HMM, or it's an HMM with an astronomical number of states. Then you'll have to settle for maximizing a variational lower bound to the intractable sum, just as in the previous cases. Ordinary EM is just the special case where the approximate distribution q(tags) is exact.

(An example of a fancy tagging model is one where multiple tokens of the same word are rewarded for having the same tag. This is not an HMM since it no longer has the Markov property.)

(Does the special case of an ordinary HMM work out in a familiar way? Yes. The maximization objective is a sum over all tag sequences and can be computed by the forward algorithm. To adjust θ to *maximize* this sum, the algorithms in the next section follow the gradient, which can be computed with the forward-backward algorithm. Alternating optimization in this case turns out to be identical to the ordinary presentation of EM for HMMs (Baum-Welch): the E step uses forward-backward to find quantities that parameterize q(tags) = p(tags | θ,words), and the M step uses those quantities to improve p's parameters θ.)

Case 4. Simplifying even more, if α and θ are both fixed, there's no training. We are just reconstructing the tag sequence under a known model. This can be done by the forward-backward algorithm—or a variational approximation to it in the case of a fancier tagging model, known as variational inference. Either way, this corresponds to only the E step of case 3.

Formally, case 4 is the same as case 2. They're both probabilistic inference of unknown variables: (θ,tags) jointly in case 2 and just tags in case 4. The different terminology ("learning" vs. "inference") merely reflects how we think about those variables. (We regard the tags as variables to be inferred because they're output variables of interest to the user. We regard θ as parameters to be learned because θ represents knowledge about the language that could be used to tag future sentences. Of course, that same knowledge about the language is implicit in the tagged corpus -- think about how EM derives θ from the tagged corpus -- but the tagged corpus has unbounded size (non-parametric) whereas θ is finite-dimensional (parametric). "Learning" usually means that you have derived some *compact* sufficient statistics of the training data, such as the parameter vector θ, that could be applied to new test data.)

Case 5. As one last simplification, we could give up on summation altogether and simply look for a single good tag sequence for the observed words, instead of an approximate distribution q(tags) over possible sequences. Making a single choice of the output variables is called decoding, and it's MAP decoding if we make the choice of maximum posterior probability. For an HMM, this can be done by the Viterbi algorithm. For a fancier tagging model, you might have to use a heuristic search algorithm like beam search or simulated annealing or something. Or you could do variational decoding, which approximates p(tags | words) by q(tags) as in case 4 and then does Viterbi decoding on q.

In fact, you could regard all 5 cases as formally the same, since maximization can be viewed as yet another a variational approximation to summation -- where "Q consists only of distributions q(y) that put all their mass on a single value y." So when you're maximizing over α or θ or tags in case 1 or 3 or 5, you can regard that as approximating the distribution over α or θ or tags in a particularly crude way, as a point distribution.

More Examples

How would you use variational Bayes for an LDA topic model?

How about a factorial HMM, factorial CRF, or factored language model?

How about the integration of a syntax-based MT system with a 5-gram language model?

Variational Optimization Techniques

We now return to our variational EM example (case 1).

Optimization: Gradient Ascent

We saw above that we can do approximate learning by maximizing the variational lower bound. But how? The most straightforward way (to my mind) is gradient ascent (or online gradient ascent), where the gradient is with respect to both the α parameters of p and the variational parameters of q.

The gradient is not too hard to compute, because the expectations in the variational lower bound are expectations under q, and q is specifically designed so that these expectations will be manageable.

For example, we'll see below what happens if Q says that q is a product distribution:

$$q(\theta,\text{tags}) = q_1(\theta) * q_2(\text{tags})$$

where furthermore q_1 is a Dirichlet distribution and q_2 is a Markov process (i.e., a weighted FSA trellis of taggings). So optimizing q actually means optimizing q_1 and q_2 by adjusting their variational parameters.

By the way, people often seem to name variational parameters by turning the true parameter name sideways: θ becomes ϕ, α becomes γ. For example, the variational parameters that define $q_1(\theta)$ might be denoted by ϕ, which specifies the mean and concentration of a Dirichlet distribution over θ.

The Case of Mean Field

Factoring $q(a,b,c,...) = q_A(a) * q_B(b) * q_C(c)$ ... like this is called a mean-field approximation. Are you curious where that term comes from? Statistical physics. Imagine that A, B, C, ... are all objects that are influencing each other magnetically or gravitationally or something (the n-body problem). p explicitly models their interactions, and we could use Gibbs sampling on p to simulate how the system evolves randomly over time. But instead, we approximate p with a model q that just describes the gyrations of each object as if it were independent of the others. This model describes the *mean* behavior of A as if it were caused by a constant background magnetic or gravitational *field*, without reference to what B, C, ... are doing at the time.

Notice how the variational lower bound becomes easy to compute (and easy to differentiate) in this case. Suppose p is defined by a graphical model as a product of potential functions,

$$p(a,b,c,...) = p_{AB}(a,b) * p_{BC}(b,c) * p_{AC}(a,c) * ...$$

Then the harder term in the variational bound can be decomposed into a sum over several terms, each of which only looks at a few variables:

$E_q \log p(a,b,c,...)$
$= E_q \log p_{AB}(a,b) + E_q \log p_{BC}(b,c) + E_q \log p_{AC}(a,c) + ...$
$= \sum_{\{a,b\}} q(a,b) \log p_{AB}(a,b) + \sum_{\{b,c\}} q(b,c) \log p_{BC}(b,c) + \sum_{\{a,c\}} q(a,c) \log p_{AC}(a,c) * ...$

This does involve the marginals of q, such as q(a,c). But crucially, taking q to be in the mean-field family makes those easy to compute (unlike the marginals of p!), because the variables are not coupled in q:

$$\begin{aligned} q(a,c) &= \sum_{\{b,d,...\}} q(a,b,c,d,...) \\ &= \sum_{\{b,d,...\}} q_A(a) * q_B(b) * q_C(c) * q_D(d) * \ldots \\ &= q_A(a) * (\sum_b q_B(b)) * q_C(c) * (\sum_d q_D(d)) * \ldots \\ &= q_A(a) * q_C(c) \end{aligned}$$

(*Warning:* The definition of p above assumed that we did not need a global normalizing constant 1/Z. Global normalization complicates things; see the final section of this tutorial.)

The Case of Structured Mean Field

Our $q(\theta,\text{tags}) = q_1(\theta) * q_2(\text{tags})$ case is called structured mean-field, because θ and tags are complex variables. We still have interactions *within* each complex variable: specifically, the components of θ must sum to 1, and the individual tags are not fully independent of one another but rather evolve according to a Markov process. But these remaining interactions are tractable. They can be handled efficiently by standard techniques like dynamic programming.

Note that there is no requirement for q_2(tags) to be a stationary Markov process (i.e., the same at every time step). This is important! Remember, q_2(tags) is approximately modeling which tags are likely at each position in the sentence, *according to the posterior distribution given the words*. So it had better treat these positions differently. If the sentence is "Mary has loved John," then a good q_2 will be very likely to transition from Verb to Verb at tag 3, but from Verb to Noun at tag 4.

Formally, q_2(tags) is defined by a trellis of taggings for each sentence in the corpus. The parameters of q_2 (within the family of approximations Q) are simply the arc probabilities in this trellis. In our example, the optimal q_2 will give a higher probability to the Verb -> Noun arc at time 4 than to the Verb -> Noun arc at time 3. So it is non-stationary.

(A trellis is a weighted FSA with a special, regular topology. Each path corresponds to a tagging of the sentence; the path's weight is proportional to the probability of that tagging.)

This non-stationarity shouldn't be surprising or computationally hard, because the usual trellis for an HMM isn't stationary either. How can that be if the HMM is stationary? Because the trellis takes specific observations into account, and the observations are whatever they are. That is, the trellis describes p(tags,words) for a particular sequence of words. Since its arc weights consider emission probabilities like p(John | Noun) as well as emission probabilities, the Verb -> Noun arc at time 4 can have a higher probability as the Verb -> Noun arc at time 3, simply because "John" is emitted at time 4.

Now, the q_2(tags) trellis is just trying to approximate the conditionalization of that HMM trellis, i.e., p(tags | words) rather than p(tags, words). For fixed θ, the approximation is exact. Notice that the E step of ordinary EM computes p(tags | words) in just this way.

Alternating Optimization: Variational EM

Gradient ascent is not the only way to maximize the variational lower bound. A more common (and illuminating) technique is alternating optimization between p and q. That is the variational EM algorithm (although above, I took the liberty of using "variational EM" to mean any algorithm that jointly optimizes (p,q), not necessarily by alternating optimization).

Remember the variational lower bound we derived above: for any α,

```
   log p(words)      >=   Eq log p(θ,tags,words) - Eq log q(θ,tags)
```

Consider the variational gap -- the difference between the left-hand and right-hand sides. You should be able to rearrange it to see that the gap is just

```
     D(q || p)
```

where

```
     D is the KL divergence
     p denotes the posterior distribution p(θ,tags | words)
     q denotes our variational approximation to it, q(θ,tags)

         (Hint: Start by rewriting log p(words) as Eq log p(words).)
```

So when we proved the lower bound, we were equivalently proving that D(q || p) >= 0. In fact, we were just repeating the usual proof via Jensen's inequality that KL divergences are always >= 0.

Okay, now for variational EM:

1. At the variational E step we adjust q given our current p. Since the left-hand side is fixed in this case, maximizing the right-hand side is equivalent to minimizing the variational gap. In other words, we want to find q that minimizes D(q || p) -- that is, q that approximates p as well as possible under this divergence measure.
2. At the variational M step, we adjust p given our current q. Since q is fixed, this means changing α to improve E_q log p(θ,tags,words).

To locally maximize the variational lower bound, we iterate these two steps to convergence. The overall algorithm does (a) approximate learning of p. Within this, the variational E step is (b) approximate inference.

Earlier at (b) I wrote: "At any local maximum ... q(θ,tags) is the best possible approximation (within Q) of the posterior p(θ,tags | words)." You can now see why: if we're at a local maximum, then q can't be improved any further by step 1, so it must already be the minimizer of D(q || p).

If the family Q is expressive enough, then we may even be able to get D(q || p) down to 0 by setting q to p. That is just the E step of ordinary EM, which actually finds the posterior distribution q(θ,tags) = p(θ,tags | words). In other words, ordinary EM is just the special case where the variational approximation q is exact. But for more complicated models, that would mean setting q to a complicated distribution that we may not be able to represent in a compact way that the M step can work with. So instead, we play the variational game, and restrict Q to force q to be simple.

More Alternating Optimization: Message Passing And Other Connections

Often the variational E step will itself involve an alternating optimization. In particular, consider our structured mean-field setting where q(θ,tags) = q_1(θ) * q_2(tags) Then the typical approach would be to argmax q_1 (with q_2 and p fixed), then argmax q_2 (with q_1 and p fixed), and finally argmax p (with q_1 and q_2 fixed).

As a special case, recall that in variational Bayes, p is fixed from the start: there is nothing to learn. Then we are just alternately improving q_1 and q_2 in order to find a q that locally minimizes D(q || p), i.e, that nicely approximates the posterior of p. At that point, q_1 tells us about likely values of θ, and q_2 tells us about likely taggings.

The update equations here end up looking like a message-passing algorithm where q_1 is sending a message to influence q_2, and vice-versa. That might remind you of belief propagation, which is beyond the scope of this note, but which also uses message-passing updates among the factors of the model, and which can be interpreted as a different kind of variational method (using D(p || q) rather than D(q || p)).

Here's another connection for you. Since q_2(tags) decomposes over sentences, we actually get q(θ,tags) = q_1(θ) * $q_{2,1}$(tags for sentence 1) * $q_{2,2}$(tags for sentence 2) * ... Alternating optimization among all these factors will update each sentence in turn given all the other sentences and θ, and then update θ given all the sentences. This starts to look like block Gibbs sampling! (Each sentence constitutes a block: "block" Gibbs (or "blocked" Gibbs) is like "structured" mean-field and "generalized" belief propagation, in that multiple variables are grouped together; it's annoying that these all use different adjectives.)

The difference is that block Gibbs would *randomly* resample a *specific* tagging for each sentence, given specific taggings of all other sentences and a specific value for θ. The variational method will *deterministically* update the *distribution* over taggings for each sentence, given distributions over the taggings for other sentences and a distribution over θ. So, it is working with distributions rather than with samples -- approximate but faster. Here's a nice analogy:

	random samples (exact)	deterministic distribs (fast approx.)
want maximum	simulated annealing	deterministic annealing
want distribution	Gibbs sampling	alternating variational opt.

Finally, consider the simpler case of variational decoding ("case 4" in earlier discussion). Here θ is given, so all we're trying to do is to decode the tags, but our tagging model is so fancy that it still requires a variational approximation. Now we don't have to learn θ: we're merely trying to find out about the fancy model's posterior p(tags | words) -- which we can't easily represent -- by constructing a simple q(tags) that approximates it. This can be done by exactly the same alternating optimization procedure as above, except that we no longer need to update q_1(θ) (i.e., there's no M step) or even have a q_1 factor. It's all about the factors $q_{2,1}$, $q_{2,2}$, etc.: we just iteratively update the distribution over each sentence's tagging given the other sentences' taggings. Alternatively, we could update these distributions in parallel by gradient descent on D(q || p), since that's what the alternating optimization is trying to optimize.

Beyond Alternating Optimization

Of course, there are other ways to try to optimize the variational objective besides gradient ascent or alternating optimization.

In particular, one could try methods that are better at avoiding local optima, which the mean-field approximation is prone to (since roughly speaking, D(q || p) is locally minimized when q fits one of the modes of p, and there may be many modes). One could thus try methods such as simulated annealing or deterministic annealing on the mean-field objective.

- I haven't seen simulated annealing used in a variational setting. It would work like alternating optimization, except that when updating one factor's variational parameters, one might not update them optimally. Rather, one would inject some randomness, particularly at early iterations.
- I believe that deterministic annealing has been applied to optimize the mean-field approximation, under the name mean-field annealing. Deterministic annealing is a form of alternating optimization, where a better-behaved function is used on earlier iterations.

A Warning About Undirected Models

In the mean-field section, I argued that the variational lower bound can be easy to compute. To do this, however, I supposed that p could be written as
`p(a,b,c,...) =` p_{AB}`(a,b) *` p_{BC}`(b,c) *` p_{AC}`(a,c) * ...`
so that each factor depended on only a few variables. Implicitly, this assumed a *directed* graphical model, which just multiplies together a number of conditional probability distributions.

What if we had an *undirected* graphical model (Markov Random Field), defined by p(...) = Φ(...)/Z? Here Z is chosen to ensure that p is normalized. Then the log p term in the variational bound would include a summand -log Z, which *depends on the p's parameters* α and is *intractable to compute*.

Good news: If -log Z is just a constant, then its value doesn't matter. We still know the variational lower bound up to an additive constant (-log Z), and can maximize it without ever computing that constant. To be precise, we are maximizing a lower bound on log Φ(words) instead of log p(words). Hence, it's still possible to do variational Bayes and variational decoding (the cases where we only adjust q, leaving p and hence -log Z constant). This allows us to still do approximate inference (referred to above as case (b)).

Bad news: However, variational EM is no longer tractable in this undirected case, since then we are also trying to adjust p (via α), and we can't tractably figure out how adjustments to p will affect the -log Z term in the variational lower bound. Thus, variational EM (like ordinary EM) fundamentally requires p to be a directed model (Z=1), or some other model whose -log Z is tractable to compute.

So what are your options for estimating p in the bad news case?

- Decide to do variational Bayes instead of variational EM, so that -log Z is an (unknown) constant. This means fixing α, on which Z depends.
- Fall back to sampling methods: use MCMC to try to estimate the gradient of log p with respect to α. This explicitly tries to get at the gradient of -log Z by sampling. (An approximation to this is contrastive divergence, which uses a quick input-specific estimate of that gradient.)
- A common trick: Recall that we're trying to tune α to maximize
- $\log p(\text{words}) = \log \sum_{\{\theta,\text{tags}\}} p(\theta,\text{tags},\text{words})$
- $= \log \sum_{\{\theta,\text{tags}\}} (\Phi(\theta,\text{tags},\text{words})/Z)$
- $= \log \sum_{\{\theta,\text{tags}\}} \Phi(\theta,\text{tags},\text{words}) - \log Z$
- $= \log \sum_{\{\theta,\text{tags}\}} \Phi(\theta,\text{tags},\text{words}) - \log \sum_{\{\theta,\text{tags},\text{words}'\}} \Phi(\theta,\text{tags},\text{words}')$,

  where Φ is parameterized by α. For the current α, we can do variational inference *twice*. As before, we get a lower bound on the first term by optimizing $q_{\text{clamped}}(\theta,\text{tags}) \approx p(\theta,\text{tags} \mid \text{words})$. We can similarly get a lower bound on the log Z term by optimizing $q_{\text{free}}(\theta,\text{tags},\text{words}') \approx p(\theta,\text{tags},\text{words}')$. The difference of these two lower bounds is not a lower bound on anything; we optimize the bounds separately, rather than optimizing their difference. The payoff is that we can now use our q functions to approximate the gradient of the true objective with respect to α:

  $$\begin{aligned}\nabla\log p &= \nabla\log \sum_{\{\theta,\text{tags}\}} \Phi(\theta,\text{tags},\text{words}) - \nabla\log \sum_{\{\theta,\text{tags},\text{words}'\}} \Phi(\theta,\text{tags},\text{words}') \\ &= E_{p(\theta,\text{tags} \mid \text{words})} \nabla\log \Phi(\theta,\text{tags},\text{words}) - E_{p(\theta,\text{tags},\text{words}')} \nabla\log \Phi(\theta,\text{tags},\text{words}') \\ &\approx E_{q\text{clamped}(\theta,\text{tags})} \nabla\log \Phi(\theta,\text{tags},\text{words}) - E_{q\text{free}(\theta,\text{tags},\text{words}')} \nabla\log \Phi(\theta,\text{tags},\text{words}')\end{aligned}$$

  The danger here is that q_{free} might be quite a poor approximation, as it must approximate the full prior distribution p(θ,tags,words) using an inadequate family Q. By contrast, q_{clamped} only has to approximate the posterior distribution p(θ,tags | words). That is often peaked around a single (hopefully correct) value (θ*,tags*), and so is easy to approximate within the mean-field family Q, by choosing $q_1(\theta)$ that is peaked at θ* and $q_2(\text{tags})$ that is peaked at tags*.

  One solution is to use better approximations. You could seek a richer family Q (at some computational expense). Or you could use (generalized) belief propagation, a generalization of mean-field that allows more parameters than mean-field; BP does not exactly construct approximations q_{clamped} and q_{free} to p, but it does approximate their marginals, which is all we really need to approximate the expectations as above.

  Another way to alleviate the difficulty in approximating p(θ,tags,words) is to switch to a contrastive estimation objective. Then the summation in Z is restricted to sum only over words' ∈ neighborhood N(words), and q_{free} becomes an approximation of p(θ,tags,words | N(words)), which might be easier to approximate within Q.

- Do conditional EM to make Z tractable: instead of maximizing likelihood, maximize conditional likelihood. This helps particularly in case 3. We switch our model from a Markov Random Field to a Conditional Random Field, defining p(tags | words) = Φ(tags,words)/Z(words). Often Z(words) is tractable. Φ(tags, words) may be complicated, with factors that relate tag unigrams or tag bigrams to arbitrary properties of the sentence. But if each of those factors considers at most a tag bigram together with the sentence, then Z(words) = $\sum_{\text{tags}}$ Φ(tags,words) can be computed with the forward-backward algorithm.
