CS 4650/7650, Lecture 4
Logistic Regression and
Expectation Maximization

Jacob Eisenstein

August 29, 2013

1 Recap

1.1 WSD

Last time we talked about word sense disambiguation, a little faster than I wanted. Here's are the key concepts:

- **Word**: *toasted* + **POS**: V
- **Lemma**: TOAST/V
- **Sense**: V:TOAST#2
- **Synset**: V:{TOAST#2, PLEDGE#3, DRINK#3, SALUTE#1, WASSAIL#2}

WordNet is a big **lexical** resource with synsets and senses for each lemma.

The "all-words" sense disambiguation problem is to label each ambiguous word with its sense.

This implicitly requires POS tagging and lemmatisation.

"Right[RB#1] now[RB#1], the focus[NN#1] is on the elected[JJ#1] official[NN#1]," she said.

Now let's talk about your examples.

1.2 Recap of discriminative classification

Last time we talked about why Naive Bayes is "naive": it assumes observations are conditionally independent. This is not true for bag-of-words features (e.g., collocations like *Georgia tech*), and it is even worse for n-gram features (e.g., *The President of*, *President of the*, *of the United*, *the United States*, ...).

One appealing solution is "error-driven" classification. We discussed two algorithms:

- Perceptron

$$\hat{y} = \arg\max_y \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}_n, y) \tag{1}$$

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}_n, y_n) - \boldsymbol{f}(\boldsymbol{x}_n, \hat{y}) \tag{2}$$

- MIRA

$$\hat{y} = \arg\max_y \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}_n, y) \tag{3}$$

$$\tau_t = \min\left(C, \frac{\ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i)}{||\boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, \hat{y})||^2}\right) \tag{4}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t(\boldsymbol{f}(y_i, \boldsymbol{x}_i) - \boldsymbol{f}(\hat{y}, \boldsymbol{x}_i)) \tag{5}$$

We can differentiate these classifiers in terms of the **loss functions** that they optimize.

- Perceptron optimizes a non-convex and discontinuous loss function, the **zero-one loss**.
  - Weight averaging can reduce thrashing and improve generalization.
  - Care must be taken to make this operation efficient.
- MIRA optimizes a convex loss function, the **hinge loss**.
  - MIRA tries to make the smallest weight change that will satisfy a constraint of zero hinge loss on the current example.
  - The introduction of slack variables allows MIRA to tolerate mistakes, improving generalization.
  - The slack parameter C controls the tradeoff between small weight changes and slack; consequently it is related to the bias-variance tradeoff.

Unlike Naive Bayes, neither perceptron nor MIRA is probabilistic. Probabilities allow us to reason about uncertainty, which may be useful in both the input and output of a classifier.

2 Logistic regression

Logistic regression is error-driven like the perceptron, but probabilistic like Naive Bayes.

Recall that NB selects weights to optimize the joint probability $P(Y, X) = P(Y|X)P(X)$.

Since we know X, we really care only about $P(Y|X)$. Logistic regression optimizes this directly. We begin by defining this conditional probability as,

$$P(y|\boldsymbol{x}) = \frac{\exp\{\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}, y)\}}{\sum_{y'} \exp\{\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}, y')\}} \tag{6}$$

$$\log P(y|x) = \sum_i \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y'} \exp \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_i, y') \tag{7}$$

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \sum_i \log P(y_i|\boldsymbol{x}_i; \boldsymbol{w}) \tag{8}$$

This is (additive inverse of) the **logistic loss**. In binary classification, we can write this as

$$\ell_{\text{logistic}}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = -(y_i \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i - \log\left(1 + \exp \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i\right)) \tag{9}$$

(draw loss function)

This loss is smooth and convex, so we can optimize it through gradient steps:

$$\ell = \sum_i \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y'} \exp \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{10}$$

$$\frac{\partial \ell}{\partial \boldsymbol{w}} = \sum_i \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \frac{\sum_{y'} \exp \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y') \boldsymbol{f}(\boldsymbol{x}_i, y')}{\sum_{y''} \exp \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y')} \tag{11}$$

$$= \sum_i \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \sum_{y'} \frac{\exp \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y')}{\sum_{y''} \exp \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y')} \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{12}$$

$$= \sum_i \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \sum_{y}{}' P(y' | \boldsymbol{x}_i; \boldsymbol{w}) \boldsymbol{f}(\boldsymbol{x}_i, y') \tag{13}$$

$$= \sum_i \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y')] \tag{14}$$

This gradient has a very pleasing interpretation as the difference between the observed counts and the expected counts.[1] Compare it with perceptron update rule.

The bias-variance tradeoff is handled by penalizing large $\boldsymbol{w}$ in the objective, adding a term of $\frac{\lambda}{2}||\boldsymbol{w}||_2^2$. This is called L2 regularization, because of the L2 norm. It can be viewed as placing a 0-mean Gaussian prior on $\boldsymbol{w}$.

This penalty contributes a term of $\lambda \boldsymbol{w}$ to the gradient, so we have,

$$\ell = \sum_i \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \log \sum_{y'} \exp \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}_i, y') + \frac{\lambda}{2}||\boldsymbol{w}||_2^2$$

$$\frac{\partial \ell}{\partial \boldsymbol{w}} = \sum_i \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y')] - \lambda \boldsymbol{w}.$$

2.1 Optimization

Batch optimization is when you can keep all the data in memory and iterate over it many times.

- The logistic loss is smooth and convex, so we can find the global optimum using gradient descent. But in practice, this can be very slow.
- Second-order (Newton) optimization would incorporate the inverse Hessian. The Hessian is

$$H_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} \ell, \tag{15}$$

[1] Recall that the definition of an expected value $E[f(x)] = \sum_x f(x) P(x)$

but this matrix is usually too big to deal with.

- In practice, people usually apply **quasi-Newton optimization**, which approximates the Hessian matrix. The specific method that is particularly popular is L-BFGS[2] NLP people usually treat L-BFGS as a black box; you will typically pass it a pointer to a function that computes the likelihood and gradient. L-BFGS is provided in `scipy.optimize`.

Online optimization is when you consider one example at a time. *Stochastic gradient descent* makes a stochastic online approximation to the overall gradient:

$$\begin{aligned}\boldsymbol{w}^{(t+1)} \leftarrow & \boldsymbol{w}^{(t)} - \eta_t \nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}^{(t)}, \boldsymbol{x}, \boldsymbol{y}) \\ = & \boldsymbol{w}^{(t)} - \eta_t \frac{1}{N}(\lambda \boldsymbol{w}^{(t)} - \sum_i^N \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y')]) \\ = & (1 - \lambda\eta_t)\boldsymbol{w}^{(t)} + \eta_t \frac{1}{N} \sum_i^N \boldsymbol{f}(\boldsymbol{x}_i, y_i) - E[\boldsymbol{f}(\boldsymbol{x}_i, y')]\end{aligned}$$

where η_t is the **stepsize** at time t.

- If we set $\eta_t = \eta_0 t^{-\alpha}$ for $\alpha \in [1, 2]$, we have guaranteed convergence.
- We can also just fix η_t to a small value. In either case, we have to tune this parameter on a development set.
- Note how similar this update is to the perceptron.

2.2 Names

Logistic regression is so named because in the binary case where $y \in \{0, 1\}$, we are performing a regression of $\boldsymbol{x}$ against $\boldsymbol{y}$, after passing the inner product $\boldsymbol{w}^\mathsf{T}\boldsymbol{x}$ through a logistic transformation. You could always do a linear regression, but this would ignore the fact that the $\boldsymbol{y}$ is to a few values.

Logistic regression is also called **maximum conditional likelihood** (MCL), because it maximizes... the conditional likelihood $P(Y|X)$.

Logistic regression can be viewed as part of a larger family, called **generalized linear models**. If you use R, you are probably familiar with `glmnet`.

[2] A friend of mine told me you can remember the order of the letters as "Large Big Friendly Giants." Does this help you?

Logistic regression is also called *maximum entropy*, especially in the older NLP literature. This is due to an alternative formulation, which tries to find the maximum entropy probability function that satisfies moment-matching constraints.

The moment matching constraints specify that the empirical counts of each label-feature pair should match the predicted counts:

$$\forall j, \sum_i f_j(\boldsymbol{x}_i, y_i) = \sum_i \sum_y P(y|\boldsymbol{x}_i; \boldsymbol{w}) f_j(\boldsymbol{x}_i, y) \tag{16}$$

Note that this constraint will be met exactly when the derivative of the likelihood function (equation 14) is equal to zero. However, this will be true for many values of $\boldsymbol{w}$. Which should we choose?

The entropy of a conditional function $p(Y|X)$ is $H(p) = -\sum_x \tilde{p}(x) P(Y|x) \log P(Y|x)$, where $\tilde{p}(x)$ is the empirical probability of x. If the entropy is large, this function is smooth across possible values of y; if it is small, the function is sharp. The entropy is zero if $P(y|x) = 1$ for some particular $Y = y$ and zero for everything else. By saying we want maximum-entropy classifier, we are saying we want to make the least commitments possible, while satisfying the moment-matching constraints:

$$\begin{aligned} \max_{\boldsymbol{w}} \quad & -\sum_{\boldsymbol{x}} \tilde{p}(\boldsymbol{x}) P(Y|\boldsymbol{x}; \boldsymbol{w}) \log P(Y|\boldsymbol{x}; \boldsymbol{w}) \\ s.t. \quad & \forall j, \sum_i f_j(\boldsymbol{x}_i, y_i) = \sum_i \sum_y P(y|\boldsymbol{x}_i; \boldsymbol{w}) f_j(\boldsymbol{x}_i, y) \end{aligned}$$

Now, the solution to this constrained optimization problem is identical to the maximum conditional likelihood (logistic-loss) formulation we've considered in the previous section.

This view of logistic regression is arguably a little dated, but it's useful to understand what's going on. The information-theoretic concept of entropy will pop up again a few times in the course. For a tutorial on maximum entropy, see`http://www.cs.cmu.edu/afs/cs/user/aberger/www/html/tutorial/tutorial.html`.

3 Summary of learning algorithms

- **Naive Bayes**. pros: easy and probabilistic. cons: arguably optimizes wrong objective; usually has poor accuracy, especially with overlapping features.
- **Perceptron and MIRA**. pros: easy, online, and error-driven. cons: not probabilistic. this can be bad in pipeline architectures, where the output of one system becomes the input for another.

- **Logistic regression**. pros: error-driven and probabilistic. cons: batch learning requires black-box software; hinge loss sometimes yields better accuracy than logistic loss.

3.1 What about non-linear classification?

The feature spaces that we consider in NLP are usually huge, so non-linear classification can be quite difficult. More often, people will add non-linear *features*, such as bigrams. Another option is to apply non-linear transformations to the feature vector. Recall that the reading defined the feature function as $\boldsymbol{f}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}) \otimes \boldsymbol{e}_y$. We can then apply non-linear transformations, such as $\boldsymbol{g}(\boldsymbol{x}) = [\boldsymbol{x}; \boldsymbol{x} \circ \boldsymbol{x}; |\boldsymbol{x}|]$, etc.

There is some work in NLP on using kernels for strings, bags-of-words, sequences, trees, etc. Kernelized learning algorithms are outside the scope of this class. Boosting and decision tree algorithms sometimes do well on NLP tasks, but they are less frequently these days, especially as the field increasingly emphasizes big data and simple classifiers.

If you propose to do a final project comparing a bunch of linear and non-linear classifiers, I won't be very excited about it. These sorts of results rarely generalize beyond the specific problem at hand. However, if you come up with a persuasive argument about why a particular type of non-linear classifier is necessary for a particular linguistic phenomenon, I would be much more encouraging.

4 Summary of classifiers

So now we've talked about four different classifiers. That's it! No more classifiers in this class. Yay? Anyway, let's review.

	Naive Bayes	Logistic Regression	Perceptron	MIRA
Objective	Joint likelihood	Conditional likelihood	0-1 loss	Hinge loss
	$\max \sum_n \log P(\boldsymbol{x}_n, y_n)$	$\max \sum_n \log P(y_n \mid \boldsymbol{x}_n)$	$\min \sum_n \delta(y_n, \hat{y})$	$\sum_n [1 - \gamma(\boldsymbol{w}; \boldsymbol{x}_n, y_n)]_+$
estimation	$\theta_{ij} = \frac{c(x_i, y=j)+\alpha}{c(y=j)+V\alpha}$	$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \sum_n \boldsymbol{f}(\boldsymbol{x}_n, y_n) - E[\boldsymbol{f}(\boldsymbol{x}_n, y)]$	$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}_n, y_n) - \boldsymbol{f}(\boldsymbol{x}_n, \hat{y})$	$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \tau_t(\boldsymbol{f}(\boldsymbol{x}_n, y_n) - \boldsymbol{f}(\boldsymbol{x}_n, \hat{y}))$
tuning	smoothing α	regularizer $\lambda \lVert \boldsymbol{w} \rVert_2^2$	weight averaging	slack penalty C
complexity	$\mathcal{O}(NV)$	$\mathcal{O}(NVT)$	$\mathcal{O}(NVT)$	$\mathcal{O}(NVT)$
easy?	very	not really	yes	yes
probabilities?	yes	yes	no	no
features?	no	yes	yes	yes

Table 1: Comparison of classifiers. N = number of examples, V = number of features, T = number of instances.

5 Document and sense clustering with EM

5.1 Motivation

So far we've assumed the following setup:

- A **training set** where you get observations $\boldsymbol{x}_n$ and labels y_n
- A **test set** where you only get observations $\boldsymbol{x}_n$

What if you never get labels y_n?
For example, you get a bunch of text, and you suspect that there are at least two different meanings for the word *concern*.[3]

The immediate context includes two groups of words:

- services, produces, banking, pharmaceutical, energy, electronics
- about, said, that, over, in, with, had

Suppose we plot each instance of *concern* on a graph

- x-axis is the density of words in group 1
- y-axis is the density of words in group 2

Two blobs might emerge. These blobs would correspond to two different sense of *concern*.

- But in reality, we don't know the word groupings in advance.
- We have to try to apply the same idea in a very high dimensional space, where every word gets its own dimension (and most dimensions are irrelevant!)
- Or we have to automatically find a low-dimensional projection. More on that much later in the course.

Here's a related scenario:

- You look at thousands of news articles from today
- Plot them on a graph of *Miley* vs *Syria*

[3] example from Pedersen and Bruce (1997)

- Three clumps emerge (Miley, Syria, others)
- Those clumps correspond to natural document classes
- Again, in reality this is a hugely high-dimensional graph

So these examples show that we can find structure in data, even without labels.

5.2 K-means clustering

So you might know about clustering algorithms like K-means. These algorithms are iterative.

1. Guess the location of cluster centers.
2. Assign each point to the nearest center.
3. Re-estimate the centers as the mean of the assigned points.
4. Goto 2.

This is an algorithm for finding coherent "blobs" of documents.
There is a variant called "soft k-means."

- Instead of assigning each point $\boldsymbol{x}_n$ to a specific cluster z_n
- You assign it a distribution over clusters $q(z_n)$

We're now going to explore a more principled, statistical version of soft K-means, called EM clustering.

By understanding the statistical principles underlying the algorithm, we can extend it in a number of cool ways.

5.3 The Expectation-Maximization Algorithm

Let's go back to the Naive Bayes model:

$$\log P(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\phi}, \boldsymbol{\theta}) = \sum_i \log P(\boldsymbol{x}_i | y_i; \boldsymbol{\phi}) P(y_i; \boldsymbol{\theta})$$

For example, $\boldsymbol{x}$ can describe the documents that we see today, and $\boldsymbol{y}$ can correspond to their labels. But suppose we never observe y_i? Can we still do something?

Since we don't know $\boldsymbol{y}$, let's marginalize it:

$$\log P(\boldsymbol{x}) = \sum_i \log \sum_y P(\boldsymbol{x}_i, y; \boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_i \log \sum_y P(\boldsymbol{x}_i | y; \boldsymbol{\phi}) P(y; \boldsymbol{\theta}) \tag{17}$$

Now we introduce an auxiliary variable $\boldsymbol{q}_i$, for each y_i. We have:

- $\sum_y q_i(y) = 1$
- $\forall y, q_i(y) \geq 0$.

In other words, q_i defines a probability distribution over Y, for each instance i. Now since $\frac{q_i(y)}{q_i(y)} = 1$,

$$\begin{aligned}\log P(\boldsymbol{x}) &= \sum_i \log \sum_y P(\boldsymbol{x}|y; \boldsymbol{\phi}) P(y; \boldsymbol{\theta}) \frac{q_i(y)}{q_i(y)} \\ &= \sum_i \log E_q[\frac{P(\boldsymbol{x}_i|y; \boldsymbol{\phi}) P(y; \boldsymbol{\theta})}{q_i(y)}],\end{aligned}$$

by the definition of expectation. (Note that E_q just means the expectation under the distribution $q_i(y)$.)

Now we apply *Jensen's inequality*, which says that because log is concave, we can push it inside the sum and obtain a lower bound.

$$\begin{aligned}\log P(\boldsymbol{x}) &\geq \sum_i E_q[\log \frac{P(\boldsymbol{x}_i|y; \boldsymbol{\phi}) P(y_i; \boldsymbol{\theta})}{q_i(y)}] \\ \mathcal{J} &= \sum_i E_q[\log P(\boldsymbol{x}_i|y; \boldsymbol{\phi})] + E_q[\log P(y; \boldsymbol{\theta})] - E_q[q_i(y)]\end{aligned}$$

By maximizing $\mathcal{J}$, we are maximizing a lower bound on the joint likelihood $\log P(\boldsymbol{x})$.

Now, $\mathcal{J}$ is a function of two arguments:

- the distributions $q_i(\boldsymbol{y})$ for each i
- the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$

We'll optimize with respect to each of these in turn, holding the other one fixed.

5.4 The E-step

First, we expand the expectation in the lower bound as:

$$\begin{aligned}\mathcal{J} &= \sum_i E_q[\log P(\boldsymbol{x}_i|y;\boldsymbol{\phi})] + E_q[\log P(y;\boldsymbol{\theta})] - E_q[q_i(y)] \\ &= \sum_i \sum_y q_i(y)\left(\log P(\boldsymbol{x}_i|Y_i=y;\boldsymbol{\phi}) + \log P(Y_i=y;\boldsymbol{\theta}) - \log q_i(y)\right)\end{aligned}$$

As in relative frequency estimation of Naive Bayes, we need to add a Lagrange multiplier to ensure $\sum_y q_i(y) = 1$, so

$$\begin{aligned}\mathcal{J} &= \sum_i \sum_y q_i(y)\left(\log P(\boldsymbol{x}_i|Y_i=y;\boldsymbol{\phi}) + \log P(Y_i=y;\phi) - \log q_i(y)\right) + \lambda_i(1 - \sum_y q_i(y)) \\ \frac{\partial \mathcal{J}}{\partial q_i(y)} &= \log P(\boldsymbol{x}_i|Y_i=y;\boldsymbol{\phi}) + \log P(Y_i=y;\boldsymbol{\theta}) - \log q_i(y) - 1 - \lambda_i \\ \log q_i(y) &= \log P(\boldsymbol{x}_i|Y_i=y;\boldsymbol{\phi}) + \log P(Y_i=y;\boldsymbol{\theta}) - 1 - \lambda_i \\ q_i(y) &\propto P(\boldsymbol{x}_i|Y_i=y;\boldsymbol{\phi})P(Y_i=y;\boldsymbol{\theta}) \\ q_i(y) &= P(Y_i=y|\boldsymbol{x}_i;\boldsymbol{\theta},\phi)\end{aligned}$$

After normalizing, each $q_i(y)$ – which is the soft distribution over clusters for data $\boldsymbol{x}_i$ – is set to the conditional probability $P(y_i|\boldsymbol{x}_i)$ under the current parameters $\boldsymbol{\theta}, \phi$.

This is called the E-step, or "expectation step," because it is derived from updating the expected likelihood under $q(\boldsymbol{y})$.

5.5 The M-step

Next, we hold $q(\boldsymbol{y})$ fixed and update the parameters. Again, we start by adding Lagrange multipliers to the lower bound,

$$\begin{aligned}\mathcal{J} &= \sum_i \sum_y q_i(y)\left(\log P(\boldsymbol{x}_i|Y_i=y;\boldsymbol{\phi}) + \log P(Y_i=y;\boldsymbol{\theta}) - \log q_i(y)\right) + \sum_y \lambda_y(1 - \sum_j \phi_{y,j}) \\ \frac{\partial \mathcal{J}}{\partial \phi_{y,j}} &= \sum_i q_i(y)\frac{x_{i,j}}{\phi_{y,j}} - \lambda_y \\ \lambda_h \phi_{y,j} &= \sum_i q_i(y) x_{i,j} \\ \phi_{y,j} &= \frac{\sum_i q_i(y) x_{i,j}}{\sum_{j'}\sum_i q_i(y) x_{i,j'}} = \frac{E_q[\text{count}(y,j)]}{E_q[\text{count}(y)]}\end{aligned}$$

So $\boldsymbol{\phi}_y$ is now equal to the relative frequency estimate of the **expected counts** under the distribution $q(y)$.

- As before, we can apply smoothing to add α to all these counts
- The update for $\boldsymbol{\theta}$ is identical: $\boldsymbol{\theta}_y \propto \sum_i q_i(y)$, the expected proportion of cluster $Y = y$. Again, we can add smoothing here too.
- Everything in the M-step is just like Naive Bayes, except we used expected counts rather than observed counts.

5.6 Coordinate ascent

Algorithms that alternate between updating various subsets of the parameters are called "coordinate-ascent" algorithms.

The objective function $\mathcal{J}$ is **biconvex**, meaning that it is separately convex in $q(\boldsymbol{y})$ and $\langle \boldsymbol{\theta}, \boldsymbol{\phi} \rangle$, but it is not jointly convex.

- Each step is guaranteed not to decrease $\mathcal{J}$
- This is called hill-climbing: you never go down.
- But the overall procedure is **not** guaranteed to find a global maximum.
- This means that initialization is important: where you start can determine where you finish.