Random probability computation tricks

Kobus Barnard, University of Arizona

Trick 1 (normalizing log probabilities).

Often in fitting models you will want to use log values in probability computations to deal with the limited precision of computers. However, if the log values are simply *proportional* to probabilities, then, at some point you may need to normalize the probabilities to sum to one. This occurs, for example, in the EM algorithm. You get the joint density for each latent variable and a data point, but to compute responsibilities (probability of latent variable given the point) you need to normalize by the sum over the values for all latent variables. This cannot be achieved in log space.

The trick is to first subtract the max over the clusters from the vector. Let:

$$M(x) = \max_c \{\log(p(x,c)\}$$

and define:

$$\log(\tilde{p}(x,c)) = \log(p(x,c)) - M(x)$$

alternatively:

$$\tilde{p}(x,c) = \frac{p(x,c)}{e^M}$$

To compute the normalization you can use $\tilde{p}(x,c)$ instead of $p(x,c)$ because the e^M cancels. However, to compute log likelihood the e^M needs to be accounted for. We substitute

$$p(x,c) = e^M \tilde{p}(x,c)$$

into the log likelihood equation to get:

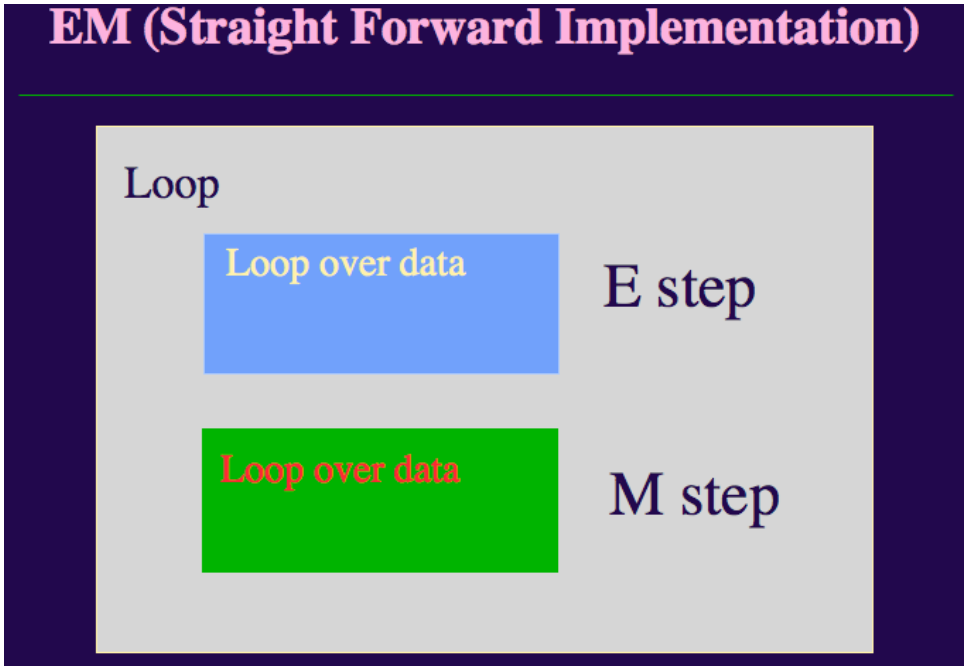
$$LL = \sum_{images} \left\{ \sum_{w \in image} \log(p(w \mid X)) + \sum_{\mathbf{x} \in image} \log(\tilde{p}(x)) + M(x) \right\}$$

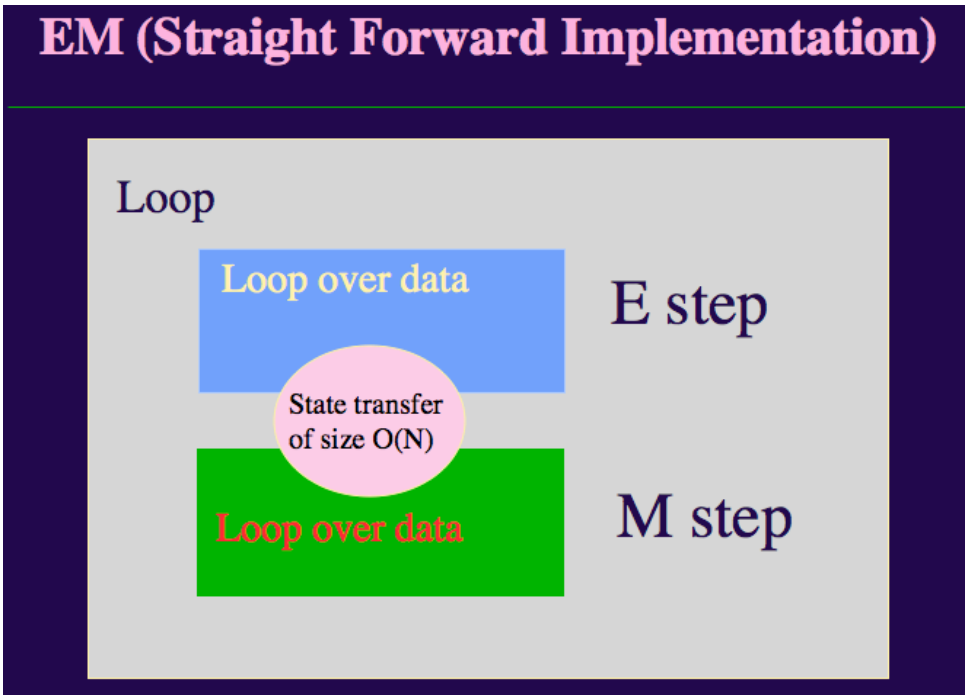
Note for users of the vision library: Available routines for facilitating this kind of normalization:

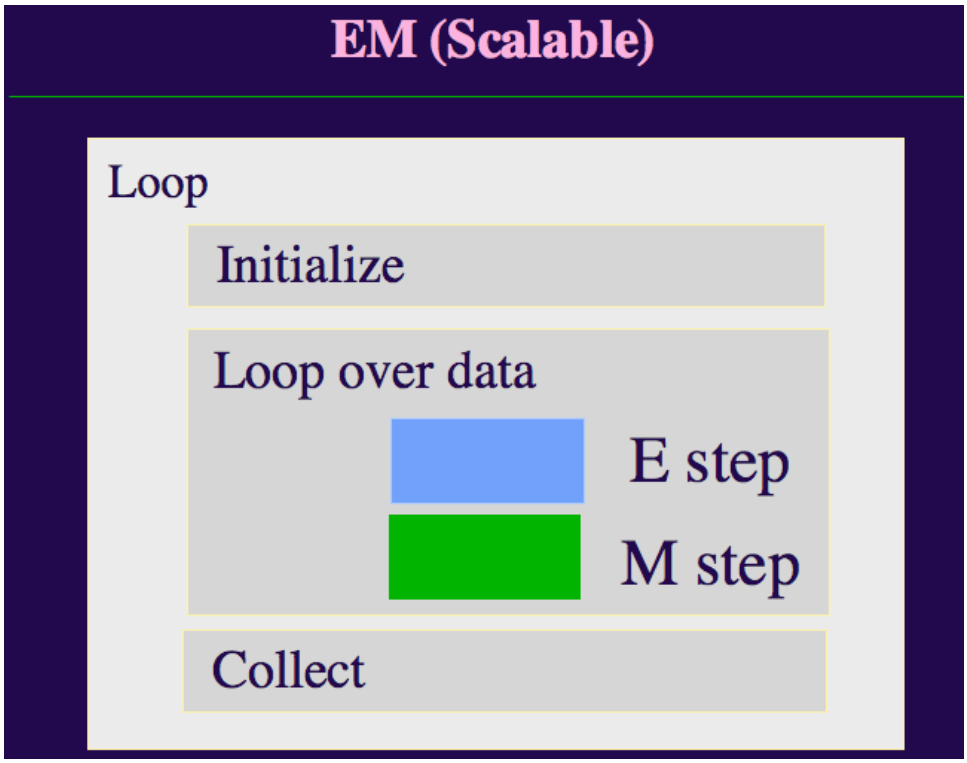
ow_normalize_log_prob_vp()
ow_exp_scale_by_sum_log_vector()

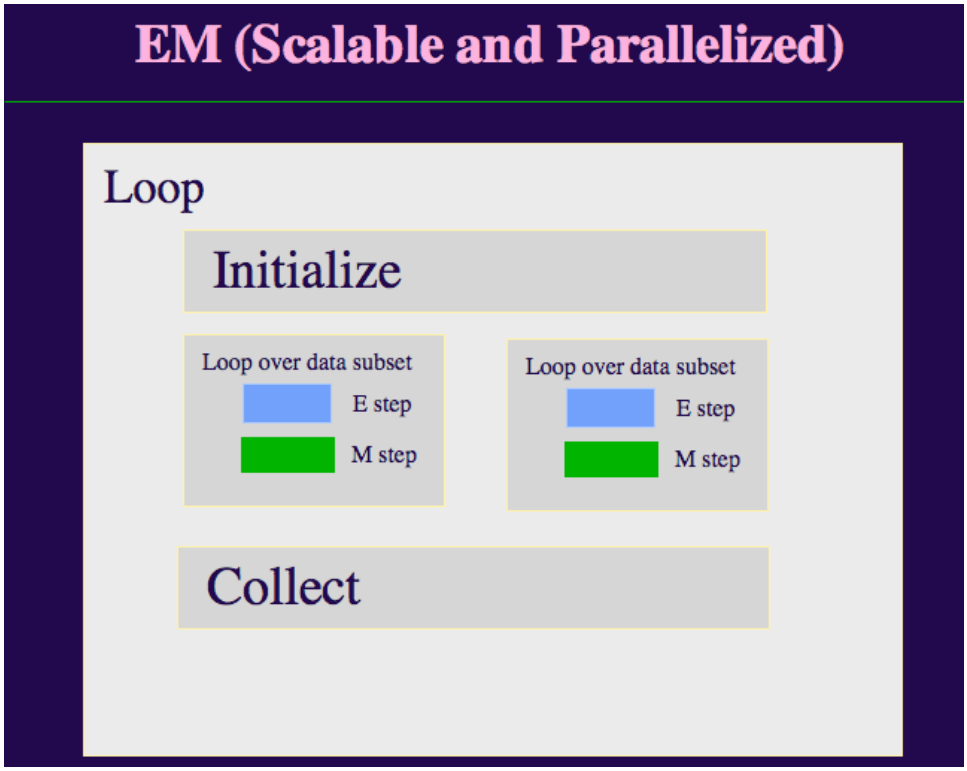
Trick 2 (doing EM one point at a time).

Depending on the model, the storage requirements for the responsibilities can be large. This is not generally the case for a simple model like the GMM, but note that regardless of the model, storage of the responsibilities goes up as the number of data points, which can be very large. The following trick keeps the storage requirements to the number of model parameters (which generally go up with more data, but the considerations are different). With a little though, you can always break EM up as suggested by the following pictures. Note that this also shows that EM is easy to parallelize.


EM (Straight Forward Implementation)
Loop
Loop over data
E step
Loop over data
M step


EM (Straight Forward Implementation)
Loop
Loop over data
E step
State transfer
of size O(N)
Loop over data
M step


EM (Scalable)
Loop
Initialize
Loop over data
E step
M step
Collect


EM (Scalable and Parallelized)
Loop
Initialize
Loop over data subset
E step
M step
Loop over data subset
E step
M step
Collect