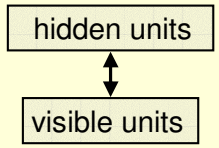
CSC2535: Computation in Neural Networks

Lecture 11: Conditional Random Fields

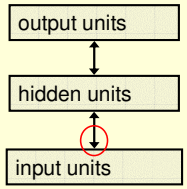
Geoffrey Hinton

Conditional Boltzmann Machines (1985)

- Standard BM: The hidden units are not clamped in either phase.
- The visible units are clamped in the positive phase and unclamped in the negative phase. The BM learns p(visible).

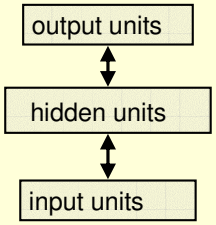


- Conditional BM: The visible units are divided into "input" units that are clamped in both phases and "output" units that are only clamped in the positive phase.
  - Because the input units are always clamped, the BM does not try to model their distribution. It learns p(output | input).

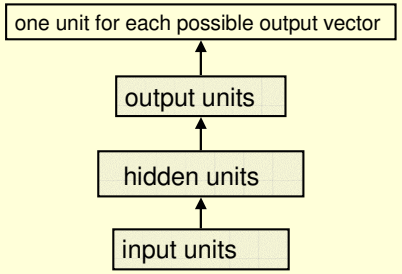


What can conditional Boltzmann machines do that backpropagation cannot do?

- If we put connections between the output units, the BM can learn that the output patterns have structure and it can use this structure to avoid giving silly answers.

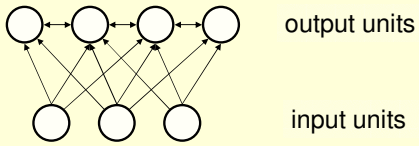


- To do this with backprop we need to consider all possible answers and this could be exponential.



Conditional BM's without hidden units

- These are still interesting if the output vectors have interesting structure.
  - The inference in the negative phase is non-trivial because there are connections between unclamped units.



Higher order Boltzmann machines

- The usual energy function is quadratic in the states:

$$E = bias\ terms\ -\sum_{i<j} s_i s_j\ w_{ij}$$

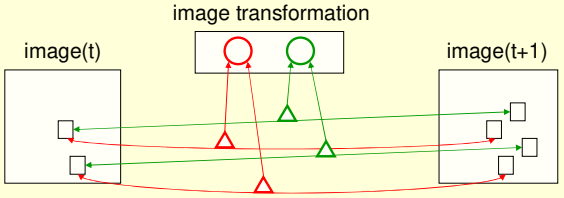
- But we could use higher order interactions:

$$E = bias\ terms\ -\sum_{i<j<k} s_i s_j s_k\ w_{ijk}$$

- Unit k acts as a switch. When unit k is on, it switches in the pairwise interaction between unit i and unit j.
  - Units i and j can also be viewed as switches that control the pairwise interactions between j and k or between i and k.

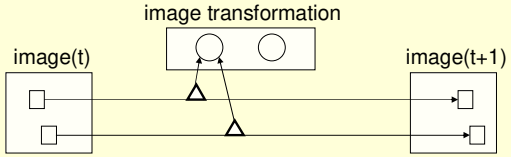
Using higher order Boltzmann machines to model transformations between images.

- A global transformation specifies which pixel goes to which other pixel.
- Conversely, each pair of similar intensity pixels, one in each image, votes for a particular global transformation.



Higher order conditional Boltzmann machines

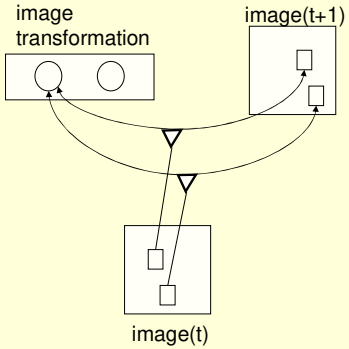
- Instead of modeling the density of image pairs, we could model the conditional density $p(\text{image(t+1)} \mid \text{image(t)})$



- Alternatively, if we are told the transformations for the training data, we could avoid using hidden units by modeling the conditional density $p(\text{image(t+1), transformation} \mid \text{image(t)})$
  - But we still need to use alternating Gibbs for the negative phase, so we do not avoid the need for Gibbs sampling by being told the transformations for the training data.

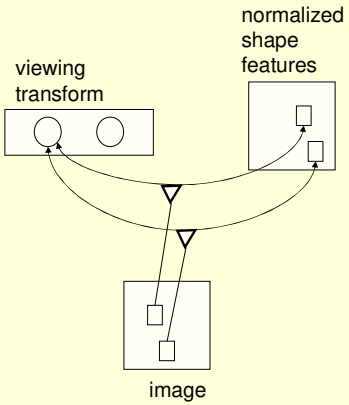
Another picture of a conditional Boltzmann machine

- We can view it as a Boltzmann machine in which the inputs create quadratic interactions between the other variables.



Another way to use a conditional Boltzmann machine

- Instead of using the network to model image transformations we could use it to produce viewpoint invariant shape representations.



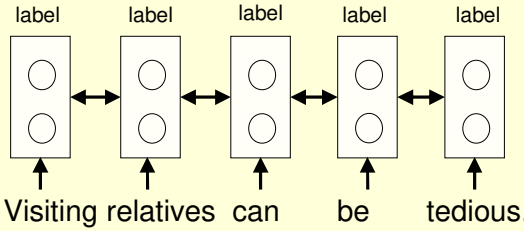
More general interactions

- The interactions need not be multiplicative. We can use arbitrary feature functions whose arguments are the states of some output units and also the input vector.

$$E = -\sum_k f_k(\mathbf{x}, \mathbf{y})\ w_k$$

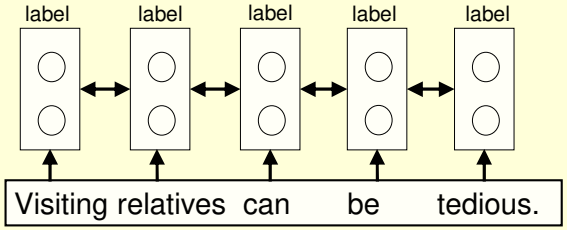
A conditional Boltzmann machine for word labeling

- Given a string of words, the part-of-speech labels cannot be decided independently.
  - Each word provides some evidence about what part of speech it is, but syntactic and semantic constraints must also be satisfied.
  - If we change "can be" to "is" we force one labeling of "visiting relatives". If we change "can be" to "are", we force a different labeling.



Conditional Random Fields

- This name was used by Lafferty et. al. as the name for a special kind of conditional Boltzmann machine that has:
  - No hidden units, but interactions between output units that may depend on the input in complicated ways.
  - Output interactions that form a one-dimensional chain which makes it possible to compute the partition function using a version of dynamic programming.



Doing without hidden units

- We can sometimes write down a large set of sensible features that involve several neighboring output labels (and also may depend on the input string).
- But we typically do not know how to weight each feature to ensure that the correct output labeling has high probability given the input.

$$\log p(\mathbf{y}\,|\,\mathbf{x}) = \sum_k w_k\ f_k(\mathbf{x},\mathbf{y}) - \log \sum_{y'} \exp\Big(\sum_k w_k\ f_k(\mathbf{x},\mathbf{y}')\Big)$$

goodness of output vector y

The partition function

Learning a CRF

- This is much easier than learning a general Boltzmann machine for two reasons:
- The objective function is convex.
  - It is just the sum of the objective functions for a large number of fully visible Boltzmann machines, one per input vector.
  - Each of the conditional objective functions is convex.
    - The learning is convex for a fully visible Boltzmann machine.
- The partition function can be computed exactly using dynamic programming.
  - Expectations under the model's distribution can also be computed exactly.

The pro's and con's of the convex objective function

- Its very nice to have a convex objective function:
  - We do not have to worry about local optima.
- But it comes at a price:
  - We cannot learn the features.
    - But we can use an outer loop that selects a subset of features from a larger set that is given.
- This is all very similar to way in which hand-coded features were used to make the learning easy for perceptrons in the 1960's.

The gradient for a CRF

$$\log p(\mathbf{y}_c\,|\,\mathbf{x}_c) = \sum_k w_k\ f_k(\mathbf{x}_c,\mathbf{y}_c) - \log \sum_d \exp\Big(\sum_k w_k\ f_k(\mathbf{x}_c,\mathbf{y}_d)\Big)$$

$$\frac{\partial\Big(\sum_{cases} \log p(\mathbf{y}_c\,|\,\mathbf{x}_c)\Big)}{\partial w_k} = \sum_{cases} f_k(\mathbf{x}_c,\mathbf{y}_c) - \sum_d p(\mathbf{y}'\,|\,\mathbf{x}_c)\ f_k(\mathbf{x}_c,\mathbf{y}_d)$$

expectation on data

expectation under model's distribution

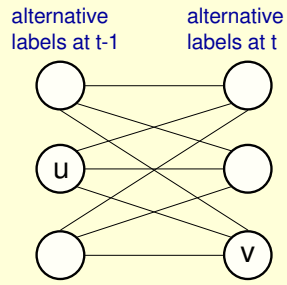
- The maximum of the log probability occurs when the expected values of features on the training data match their expected values in the distribution generated by the model.
  - This is the maximum entropy distribution if the expectations of the features on the data are treated as constraints on the model's distribution.

Learning a CRF

- The first method used for learning CRF's used an optimization technique called "iterative scaling" to make the expectations of features under the model's distribution match their expectations on training data.
  - Notice that the expectations on the training data do not depend on the parameters.
    - This is no longer true when features involve the states of hidden units.
- For big systems , iterative scaling does not work as well as preconditioned conjugate gradient (Sha and Pereira, 2003).

An efficient way to compute feature expectations under the model.

- Each transition between temporally adjacent labels has a goodness which is given by the sum of all the contributions made by the features that are satisfied for that transition given the input.
- We can define an unnormalized transition matrix with entries



$$M_{uv}^{t} = e^{\sum_i w_i f_i(y_{t-1}=u,\, y_t=v,\, \mathbf{x})}$$

Computing the partition function

- The partition function is the sum over all possible combinations of labels of exp(goodness).
- In a CRF, the goodness of a path through the label lattice can be written as a product over time steps

$$Z = \sum_{P \varepsilon\ paths} \prod_{t=1}^{t=T} e^{G_{P_{t-1},P_t}^{t-1,t}}$$

We can take the last exp(G) term outside the summation.

where P_t is the label in path P at time t

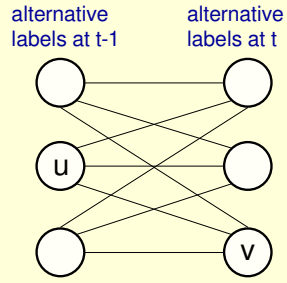
The recursive step

- Suppose we already knew, for each label, u, at time t-1, the sum of exp(goodness) for all paths ending at that label at that time. Call this quantity

$$\alpha_u^{t-1}$$

- There is an efficient way to compute the same quantity for the next time step:

$$\alpha_v^t = \sum_u \alpha_u^{t-1} M_{uv}^t$$



The backwards recursion

- To compute expectations of features under the model we also need to compute another quantity which can be done recursively in the reverse direction.
- Suppose we already knew, for each label, v, at time t+1, the sum of exp(goodness) for all paths starting at that label at that time and going to the end of the sequence.
- Call this quantity β_v^{t+1}

$$\beta_u^t = \sum_v \beta_v^{t+1} M_{uv}^{t+1}$$

Computing feature expectations

- Using the alphas and betas, we can compute the probability of having label u at time t-1 and label v at time t. Then we just add the feature value over all pairs of times.

$$< f_i(\mathbf{x}) > = \sum_t \frac{\sum_{uv} \alpha_u^{t-1} \beta_v^t M_{uv}^t \; f_i(y_{t-1}=u,\, y_t=v,\, \mathbf{x})}{Z(\mathbf{x})}$$

$$Z(\mathbf{x}) = \sum_u \alpha_u^T$$

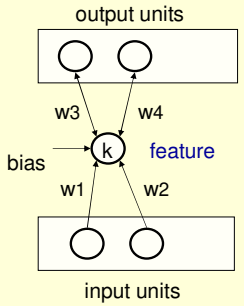
The partition function is found by summing the final alphas.

Feature selection versus feature discovery

- In a conditional Boltzmann machine with hidden units, we can learn new features by minimizing contrastive divergence.
- But the conditional log probability of the training data is non-convex, so we have to worry about local optima.
- Also, in domains where we know a lot about the constraints it is silly to try to learn everything from scratch.

Feature selection versus feature discovery

If we fix all the weights to the hidden units and just learn the hidden biases, is learning a convex problem?

Not if the bias has a non-linear effect on the activity of unit k.

To make learning convex, we need to make the bias scale the energy contribution from the state of unit k, but we must not allow the "bias" to influence the state of k.