PGMs week 4 Inference: Belief Propagation

- As you would have discovered from last week's work, determining a full joint probability might be an extremely expensive exercise when there are many variables. We are now going to study a number of *message passing* algorithms to do this more efficiently.
- Read Barber chapter 5 Section 5.1: Variable Elimination and Belief Propagation. This might seem short, but the Koller videos (below) will substantially expand on it.
  - Section 5.1 starts out with variable elimination and then shows how this leads to the Sum-Product algorithm in linear and tree-structured networks.
  - This is specifically developed here for *factor graphs*. This consists of nodes for the individual variables that are connected to factor nodes according to the potentials that these variables partake in. In these graphs there are two distinct types of *messages*:
    - From a variable node to a factor node the message is the product of all the *other* messages that lead into that variable.
    - From a factor node to a variable node this product (from the previous step) is multiplied with the factor potential, and then all the other variables (except the destination variable) gets marginalised out.
- **WATCH** the Koller videos on Inference, Variable Elimination and Message Passing:
  - *PGM31 - Knowledge Engineering*
  - *07.1 - Inference: Conditional Probability Queries*
  - *07.2 - Inference: MAP Inference*
  - *07.3 - Inference: Variable Elimination: Algorithm*
  - *07.4 - Inference: Variable Elimination: Complexity Analysis*
  - *07.5 - Inference: Variable Elimination: Graph-Based Perspective*
  - *07.6 - Inference: Variable Elimination: Finding Elimination Orders*
  - *PGM34 - Message Passing: Belief Propagation Algorithm*
  - *PGM35 - Message Passing: Cluster Graph Properties*
  - *PGM36 - Message Passing: Properties of BP Algorithm*
  - *PGM41 - Message Passing: BP in Practice*
  - *PGM42 - Message Passing: Loopy BP and Message Decoding*
- Notes on these videos:
  - PGM31 is for general interest, we have not discussed template and plate models as yet, so do not be overly worried when you encounter them.
  - 07.1 and 07.2 provide overviews of work still to be done. (Do note in 07.2 the example that shows that the max of the joint is not necessarily equal to the max of the marginals.)
  - The rest of the 07 series discusses Variable Elimination. This is an important topic which will also find use in later work.
  - PGM34-PGM36 introduces belief propagation (i.e. the Sum-Product / Schaefer-Shenoy algorithm). In contrast to Barber which approaches this from a factor graph viewpoint, Koller uses a more general formulation that paves the way for the Junction-Tree Algorithm (future work).
  - PGM41 Shows some variants and tricks to improve convergence in loopy belief propagation.
  - PGM42 Shows the historical link between loopy belief propagation and error correction coding.
  - Note that the factor graph approach translates to a specific case of this more general formulation – the Bethe cluster graph. The two types of messages that we saw in the factor graph is a simplification arising from the fact that in a Bethe cluster graph the sepset corresponds exactly to the particular variable on that specific link.

  - When our variable clusters are connected in loops, the algorithm might converge to the wrong solution, or might even oscillate. Quite often though, it converges to the correct solution.

- Make sure you understand the following:
  - A 'message' really is a *distribution/potential* describing what the originators of the message believes about the variables they have in common with the target cluster. The target cluster's own opinion about itself should not be included in this. (Otherwise you get the inflated potential runaway ego situation we so often observe in cults, be it political or religious.)
  - Initially messages are initialised to unity. This just means they still have no opinion about their variables, the potential of the cluster rules.
  - Numerical issues: With unnormalized potentials (as we typically use in MRF's) we run a risk of exhausting our numerical range with either under- or overflows. We basically have two avenues to cure this (see discussion in Barber at the end of section 5.1.2, and also 5.1.4):
    * Normalise the messages before passing them on. The works fine, but do note that it will no longer be possible to calculate the partition function value Z by finding the normalization constant at any arbitrary variable cluster. You will have to normalize all your clusters separately (resulting in a $Z = 1$), or alternatively find the full joint and calculate Z from that.
    * Use logarithmic potentials. With these we need to extend our inventory of factor operations somewhat:
      · Potential multiplication now change to potential summation.
      · Observing evidence/conditioning is done as before.
      · To do marginalisation you need to first convert your logarithmic potentials to linear form again. However, direct/naive linearisation does not work – you immediately fall prey to the original over/underflow problem again. The trick (known as the logsumexp trick) is to realise that:

$$\begin{aligned}\log(\sum_i e^{L_i}) &= \log(e^{\max\{L_j\}} \sum_i e^{L_i - \max\{L_j\}}) \\ &= \max\{L_j\} + \log(\sum_i e^{L_i - \max\{L_j\}})\end{aligned}$$

  - The *cluster belief* is the product of the cluster potential and all the messages entering it. It serves as a pseudo-marginal.
  - The *sepset belief* is the product of the two messages traveling in opposite directions through it. It serves as a pseudo-marginal.
  - Convergence implies *Calibration*: This simply means that on convergence we can choose any two of the beliefs available (cluster as well as sepset), and calculate marginals for the variable(s) they have in common. The distributions obtained in this manner must be identical. It should not happen that one part of the system has a different idea about the distribution of some variable(s) than what some other part has.
  - Note that the product of the cluster beliefs divided by the sepset beliefs is the same as the product of the original potentials (known as reparameterisation). While they therefore result in the same joint potential, the message passing procedure have at the same time refined them to provide the marginals for each cluster.
  - Those familiar with the use of the forward-backward algorithm in the context of HMMs, will see that the belief propagation algorithm generalises that algorithm.

- Exercise:

  Objectives:
  - Implement/use sum-product messages to calculate marginal probabilities without first calculating the full joint probability. I.e. we can calculate probabilities while avoiding the exponential blow-up of the full joint.
  - Verify that after convergence the system achieves calibration i.e. the various beliefs in the system agrees on the marginals of the variables it knows about.
  - Verify the re-parameterisation property, i.e. we can recover the full joint from the various cluster and sepset beliefs. Of course we normally would not want to do this, since it once again introduces the exponential factor size blow-up.
  - Compare the behaviour of (the more generic) cluster graphs with that of factor graphs.

Task 1: In the following, use only the basic factor operations we encountered previously – we first want to foster an understanding of the basics of message passing. We are going to re-implement the hard-decoding (BSC) version of the Hamming (7,4) code of last week, but this time using a special graph structure called a Cluster Graph. Construct it as follows:

- First construct all the factors as you did in week 3.
- The scope of each factor is the set of random variables it contains. In contrast to last week we are not going to now create a joint factor by multiplying all available factors out – instead we want to arrange them into a graph structure. However, if the scope of a factor is a subset of another factor, we can (optionally) simplify things by assimilating it in that other factor. We do this by simply multiplying them. (Note, when a factor is a subset of more than one other factor, we will only multiply it out with *one* of them. I.e. each factor occurs only once in the system.)
- The (in this case three) factors you are now left with, need to be linked up in a graph structure which obeys the RIP principle. RIP requires that for all variables X_i in the PGM, when considering only those nodes/clusters containing X_i (as well as the links between them), the resulting sub-graph must form a *tree*. (Study the Koller PGM35 video until you are clear about what RIP entails.) To form this graph involves two steps:
  - Decide which factors are to be linked and
  - specify their *sepset*, i.e. the set of variables those two factors will exchange information about. Note, the sepset is a *subset* of the random variables those two factors have in common.

  In particular, pay careful attention to which sepsets contain b_0 (the variable that is common to all three parity checking factors). After having done this yourself you can confirm that you end up with something similar to Figure 1. We represent the clusters inside ellipses, and their sepsets inside boxes.

Now we are ready to start passing messages around between the nodes in this graph:

- Since we have a loopy structure we have to place initial messages at all relevant places (consult the videos). Each message is a `pgmpy DiscreteFactor` (or `emdw DiscreteTable`) factor, the scope of which is being specified by the variables in the corresponding sepset. Each message is contained in a separate such variable, you might want to consider a coding scheme such as `m01` being the message that is sent from cluster 0 to cluster 1 etc.
- Now we repeatedly pass messages between the various nodes using the loopy belief propagation algorithm. You have to determine a schedule for passing messages. For this specific application a natural option is to go once in one direction around the loop, and then once in the other direction around the loop. You will also have to take precautions against numerical under/overflow. For this it probably is easiest to simply normalize each message.
- Iterate/repeat this message passing until *calibration* is achieved between all clusters (i.e. the sepset beliefs do not change any more). Consult the Koller PGM36 video for determining how a cluster belief and a sepset belief is calculated. You will have to decide how you are going to quantify such a *change* in belief. When a graph is calibrated, all beliefs are in agreement as to what the distribution for the variable(s) are.
- It might also be interesting to plot the convergence behaviour as the biggest change in sepset belief as a function of the iteration number.
- Decoding is complicated by the fact that picking the max of the marginals is not necessarily equivalent to picking the maximum of the joint assignment – an elegant solution for this dilemma forms part of the MAP/max-sum message passing algorithm which we will discuss at a later stage. For now we will have to content ourselves by interpreting the maximum values of the cluster beliefs – this should give us a fair idea as to what is happening. Alternatively we can form the full joint by making use of reparameterisation (see the last part of the Koller PGM36 video) and then picking the max of the joint. The division operator in the PGM toolbox will be useful here for dividing by the sepset potentials. This should return the most likely answer.
- How does this compare with your results from last week?
- Due to the small size of the problem we are considering, the computational load will be higher than what we previously encountered. This seems to negate our very reason for doing message passing. Confirm that you are clear why message passing can alleviate the computational load in bigger problems.
- Can you create situations with bad convergence behaviour?
- Investigate what happens when every pair of parity checking clusters do share b_0 i.e a loop of b_0 sepsets violates the uniqueness requirement of RIP.

Task 2: Repeat the above, but using a factor graph approach. Compare the results, especially in terms of convergence behaviour.

Further exploration: The Koller video PGM41 contains a number of very useful ideas to explore. Tree Reparameterization, Residual Belief Propagation, Synchronous vs Asynchronous propagation and Message Damping. All worth while to look at.

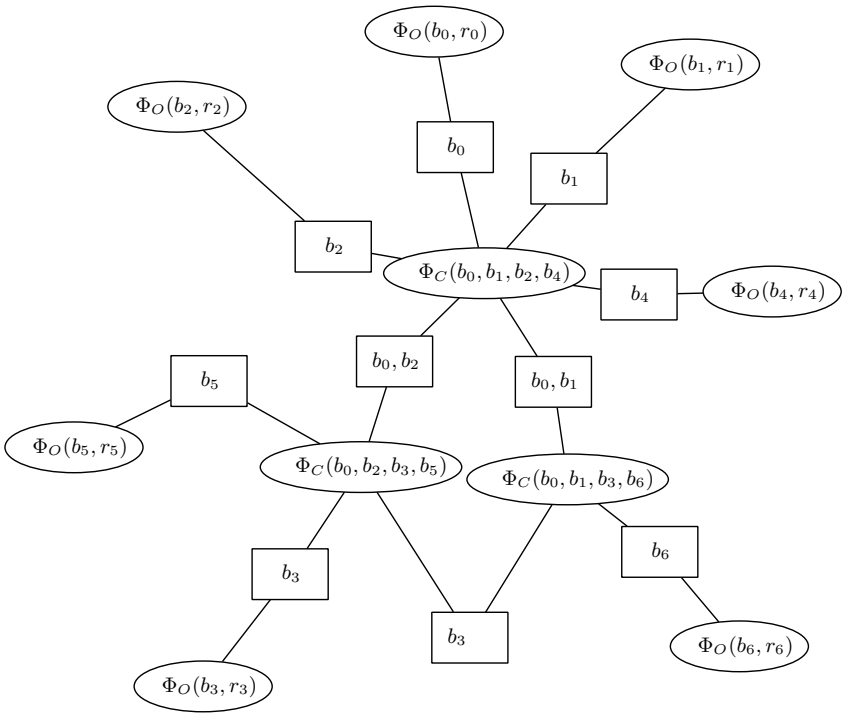

Figure 1: A cluster graph for the Hamming (7,4) error correcting code. Note the sepset in the bottom center where b_0 has been omitted to achieve valid RIP. With the r_i's observed, this graph can be simplified to only an inner loop of three clusters by assimilating each resulting $\Phi_o(b_i)$ into the Φ_C cluster it is connected to.