Day 3&4 (sickness day 3).

Lecture 1. Graphical Models, Dynamic Programming, and Binocular Stereo.

A lot of recent work formulates probability distributions on graphs (e.g., see <http://www.ipam.ucla.edu/programs/gss2007/>, <https://www.ipam.ucla.edu/schedule.aspx?pc=gss2011>). This has the advantages of knowledge representation, data reduction (learning), and computation. A full distribution over N binary valued variables has an exponential number of possible states, so there is usually not enough data to learn this distribution, it takes too much computation to find the most probable state (have to search over an exponential number of states), and it doesn’t give much insight into the data. Graphical models (by specifying nodes and edges) give knowledge by making it explicit which variables directly influence each other, and reduce the complexity of the distribution allowing for rapid inference and efficient learning. The edges define the variables which directly influence each other – the Markov structure. Markov Random Fields are graphical models, but there are other examples such as stochastic grammars, see tutorial by Griffiths and Yuille. A particular important class of graphical models are those for which the graph has no closed loops – in this case, dynamic programming (due to Bellman) enables efficient inference in polynomial time. Dynamic programming (DP) has a forward pass – to compute the lowest energy – and a backward pass to determine the configuration that minimizes the energy. There are several variants of DP – it can also be used to compute the marginal distributions efficiently and to compute the expectations of certain statistics. DP can be extended to some graphs with closed loops by the junction tree mechanism. Binocular stereo is the task of estimating the depth from two images (e.g., the left and right eye). It requires knowing the directions of gaze and the distance between the cameras (this is known as calibration). To obtain depth, after calibration, requires finding the corresponding points on the left and right images – called the correspondence, or matching problem. The geometry of stereo mean that points in one camera can only be matched to points on a line in the other camera (the epipolar line constraint). If you know the epipolar lines (which follow from the camera calibration) then stereo can be formulated as a one-dimensional matching problem in terms of the disparity, the displacement between the position of a point in one camera and the position of the corresponding point in the second (the depth can be computed from the disparity). Stereo can be formulated in terms of a Markov Random Field (MRF) which contains a unary term – for matching the local intensity properties (e.g., feature responses) between the two images – and a binary term (or higher order terms) to impose prior assumptions, such as weakly smooth disparity (e.g., the TV-norm). in some extreme cases, the local intensity properties are sufficient to give unambiguous matches – but is not true for almost all real images and the prior assumptions are necessary to give an unambiguous result. The most probable state of this MRF can be estimated using DP (exploiting the one-dimensional structure and the local interactions). More advanced models can be obtained which model the fact that some parts of the scene are visible to one camera only – called half-occlusion – and these unmatched points give clues for sharp changes in the depth (da Vinci stereopsis). Stereo can also be formulated as a two-dimensional matching problem which adds the prior assumption that the disparity is also weakly smooth across the epipolar lines (the earlier model assumed weak smoothness only along the epipolar line). This gives better results, but requires the use of belief propagation, or max-flow/min-cut, to estimate the most probable state (since this model has many closed loops).

Lecture 2. Review of Maximum Likelihood Learning with and without Hidden Variables.

Most of this material was summarized in the summary for day 2. But this lecture was added to full in some details, to give the connection between model selection and minimizing entropy, and to clarify aspects of feature pursuit. The lecture discussed the difficulty of performing the two steps of the EM algorithm -- and said that DP could be used to perform them if the graph had no closed loops. This shows that, for example, that EM can be applied to Hidden Markov Models using dynamic programming (only needing to know that HMMs have no closed loops – not needing to know anything else about HMMs).

Lecture 3. Hidden Markov Models (HMMs).

These models were designed for speech recognition (most practical systems are build using them). They are probability models without closed loops. There are hidden (unobserved) states which have probabilities for emitting observations. There are also transition probabilities between the unobserved states. The classic way to describe this is as follows – there are three urns (vases) which each contains a mixture of balls of different colors, so that one urn contains mostly blue balls, another mostly red balls. Only one urn is visible at each time step (the urns look the same, so you don’t know which one) and there is a hidden process which changes the urns between the different time steps. You, the observer, pick a ball from the visible urn at each time step. The learning task is to estimate the transition probability between hidden states (i.e. the probabilities of transitions between the urns) and the observation model (i.e. the proportion of colored balls in each urn). This is not easy. HMMs can be applied to vision (see Peng paper) where you have an HMM for four different baseball activity (e.g. hit home run, get caught). The HMM for each activity has a set states – the camera shots (e.g., pitch view) – and there can be observations which are image features which help identify which type of camera shot you are seeing. There are three tasks: (i) learn an HMM for each activity using a dataset of image sequences for each activity – this is done by EM using DP to compute the two steps of EM, (ii) decide which activity you have watched in one image sequence (e.g., home run, or catch) which is done by model selection (i.e. compute probability of data using the each HMM, and select the HMM which has highest probability) which also uses DP, (iii) estimate the most probable values of the hidden states for one sequence (also uses DP). The notes, not discussed in class, give details of the formulation of HMMs and how these three tasks are performed.

Lecture 4. Support Vector Machines.

This lecture introduces support vector machines (SVMs) for binary classification (SVMs can be generalized to do more). The input is training data -- a set of features x and a binary value y . The basic idea, dating back to Perceptrons, is to find a plane that separates the data so that positive examples (y=1) are above the plane and negative examples (y=-1) are below. You need to make sure that this decision rule (above plane or below plane) generalizes to new data that you havn’t see – you don’t want a rule that memorizes the data and gives perfect results on the training data (this is tested by cross-validation using a testing dataset, which you havn’t trained on). SVM uses a max-margin criterion. This attempts to maximize the margin separating the positive and negative examples, while paying penalty for slack variables (z’s) that enable misclassified data points to be moved to the correct sides of the margin. The max-margin criterion is quadratic in the primal variables (those that define the plane and the slack variables) and contains lagrange parameters which impose the constraints that the data points lie on the correct sides of the margins (after using slack variables if necessary) and that the slack variables are always positive or zero. Minimizing the criterion with respect to the plane variables shows that the solution is a weighted sum of the data points – the weights are zero unless the datapoints are on the margins, these are the slack variables. The weights depend on the values of the lagrange parameters. These can be obtained by solving the dual problem (a function of the lagrange parameters only) which can be obtained by solving the max margin criterion for the primal variables – in terms of the data and the dual variables – and using these solutions to eliminate them from the criterion. The kernel trick can be used to extend to non-linear ways to separate the data – popular kernels include polynomials and radial basis functions.

Lecture 5. AdaBoost and Face Detection.

AdaBoost is a technique for learning a strong classifier (highly accurate) by linear combined weighted weak classifiers (less accurate). There is a dictionary of weak classifiers (pre-specified) and so the learning task is to select weak classifiers from this dictionary and weight them (this can be reformulated as an approximation to regression – learning a parameterized conditional distribution P(W|I, lambda) which is of exponential form and is very similar to Maximum Likelihood learning – select lambda to maximize the product of P(W|lambda) for the training data, so can SVM). AdaBoost can be formulated in terms of minimizing Z which is a convex function of the weights lambda of the weak classifiers. Z is an upper bound of the error of the strong classifier and so we can make the error small by making Z small. We initialize the weights by setting lambda =0 for all weak classifiers, which means that none of them are selected. Then we perform coordinate descent by finding the lambda (weak classifier) which decreases Z most at each time step (the form of Z makes this easy to compute). This process selects and weights classifiers simultaneously. Viola and Jones applied AdaBoost to face detection and achieved excellent results in real time – they used thresholded Haar features as weak classifiers. AdaBoost selected intuitive Haar features (e.g., detecting symmetry, the change in intensity between the forehead and the eye region). They also used an attentional mechanism so that they don’t need to search all the image to find faces (easy to rule out some regions very simply). (Note: AdaBoost is an approximation to doing regression – learning a conditional distribution).