Machine Learning B

708.062 12W 1sst KU, WS 2012/13

Exercises

Problems marked with * are optional.

1 Comparison of Optimization Algorithms [5 P]

Apply the four optimization algorithms *gradient descent*, *genetic algorithms*, *simulated annealing*, and *particle swarm optimization* to minimize each of the following four 2-dimensional functions;

1) Rastrigin:

$$f_1(\mathbf{x}) = \sum_{i=1}^{2}(x_i^2 + 10 - 10\cos(2\pi x_i))$$

in the interval $\mathbf{x} \in [-5, 5]^2$,

2) Rosenbrock:

$$f_2(\mathbf{x}) = (x_2 - x_1^2)^2 + (x_1 - 1)/2 + 2(|x_1 - 1.5| + |x_2 - 1.5|)$$

in the interval $\mathbf{x} \in [-2, 2]^2$,

3) Ackley:

$$f_2(\mathbf{x}) = e + 20 - 20\exp(-0.2\sqrt{1/2\sum_{i=1}^{2} x_i^2}) - \exp(1/2\sum_{i=1}^{2}\cos(2\pi x_i))$$

in the interval $\mathbf{x} \in [-8, 8]^2$,

4) Chasm:

$$f_2(\mathbf{x}) = 1000|x_1|/(1000|x_1| + 1) + 0.01|x_2|$$

in the interval $\mathbf{x} \in [-5, 5]^2$

and compare their performance and run time.

a) Download the MATLAB code for the genetic algorithm toolbox (`gatoolbox`), simulated annealing (`anneal.m`) and the calculation of the gradient (`gradient.m`).[1]

[1] http://www.igi.tugraz.at/lehre/MLB/WS12/task1.zip

b) Implement the particle swarm optimization algorithm. Complete the MATLAB function `pso.m`.

c) Implement the gradient descent algorithm with adaptive learning rate by estimating the gradient with finite differences as provided in the MATLAB function `gradient.m`.

d) In order to carry out the optimization you can write your own MATLAB code or use the code template provided in `compare.m`. Missing code fragements that have to be completed are marked with "`... HOMEWORK ...`".

e) For each of the four optimization problems set the free parameters of each of the four optimization algorithm to appropriate values. The free parameters are marked with "`... HOMEWORK ...`" in the MATLAB code. Explain and justify your choice.

f) Repeat the minimization for each of the four functions with each of the four algorithms 10 times (with the parameters settings chosen in e)) and calculate the mean and the standard error of the mean (SEM) of the resulting minimum function values and the corresponding run times.

g) Compare the results obtained in f) and interpret possible strengths and weaknesses of each optimization algorithm with respect to the optimized functions. Hand in graphical illustrations of your results that support all your statements.

Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortless.

2 Cart-Pole Controller Optimization [5 P]

Optimize the weights of a neural network that controls the horizontal forces applied to a cart in order to swing up a pole that is mounted on it into the upright position. Use an optimization algorithm of your choice (one of the four algorithms used in homework assignment 1). The state $\mathbf{x}$ of the dynamical system is defined as a four dimensional vector with the elements $(xc, \dot{xc}, \varphi, \dot{\varphi})^T$, where xc is the position of the cart and φ is the pole angle.

a) Download the MATLAB code for the cart-pole.[2] The dynamics of the cart-pole for one time step is computed in the function `cp_dyn.m`. The state `x` of the cart-pole can be visualized with the function `cp_vis.m` (you don't have to change or call these two functions).

[2] http://www.igi.tugraz.at/lehre/MLB/WS12/task2.zip

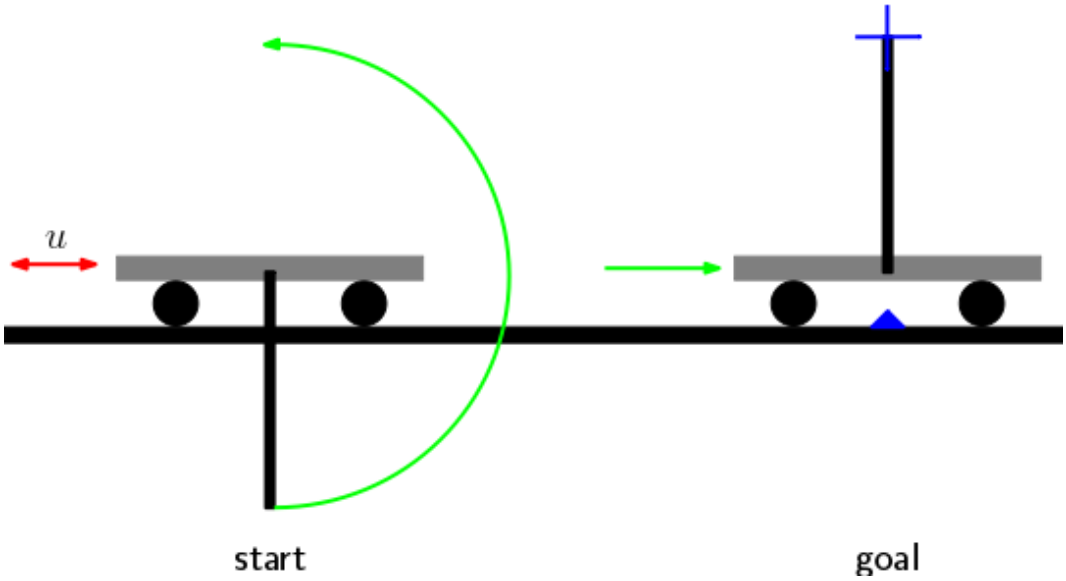

Figure 1: The cart-pole model.

b) Modify the file `learn_cp.m` that contains also the parameters for the cart-pole in the structure `model` (masses, length of the pole etc.) to implement an optimization algorithm that minimizes the error computed with the function `cost_function.m`.

c) Modify the file `cost_function.m` that simulates the cart for a duration of 4 seconds and assigns an error value to the dynamics of the cart. You have to i) complete the function `cost_function.m` to implement a neural network controller that generates the horizontal force u that is applied to the cart at each time step, and ii) define an appropriate error function e (that scores the cart-dynamics for each time step), which has to be minimized.

d) **Neural network controller:** Write the code to initialize and simulate a neural network without using the neural network toolbox. The input to the network for each time step should consist of the following 5 values: $(xc, \dot{xc}, \sin\varphi, \cos\varphi, \dot{\varphi})^T$. The network consists of a single hidden layer with 5 hidden units. The scalar output of the neural network, i.e. the force, should be bounded with values between -10 and 10.

Hints: Don't forget to optimize the bias values of the neurons. Use a tansig output neuron to obtained bounded output values.

e) **Error function:** Use the state variables to construct an error function e that outputs a value at each time step that assure that the pole remains in the upright positions after the upswing. Set the error to 10^6 if the cart

leaves the interval $[-1, 1]$. The total error (or cost) assigned to the cart movement is the sum of all e for all time steps.

Hints: The key to success is an appropriate error function.

f) Analyze the best solution found and state the reasons for all choices you made in the MATLAB code.

Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortless.

3 Genetic Algorithm [3* P]

Apply the genetic algorithm toolbox in MATLAB to an optimization problem of your choice for which you can show by simulation that the encoding of the variables (that should be optimized) in the genome has an effect on the convergence speed of the algorithm. Describe precisely the optimization problem and the key ideas behind the encoding scheme. Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortless.

4 Genetic Coding [4 P]

a) Choose 2 concepts from Lecture 3 that you found interesting, and provide more precise definitions and explanations to each of them (about 1/2 page for each), on the basis of online-sources (give references to these sources).

b) Choose one process from Lecture 3 that you found interesting, and explain this process in more detail (on 1-2 pages), on the basis of online-sources (give references to these sources).

5 RL theory I [3 P]

Prove **Corollary 1.3** (p. 9) from the script *Theory of Reinforcement Learning* [3]:

Every policy π for which V^π satisfies the Bellman optimality equations

$$V^\pi(s) = \max_{a \in A_s} Q^\pi(s, a) \ \forall s \in S$$

is optimal.

[3] http://www.igi.tugraz.at/lehre/intern/MDP_Theory.pdf

6 RL theory II [3 P]

Assume that for a given continuing MDP with discount factor $\gamma < 1$ we modify the reward signal by either

a) adding a constant d to all rewards

b) multiplying every reward with a constant $k > 0$

c) linearly transforming the reward signal to $k \cdot r + d$, $k > 0$

Can this change the optimal policy of the MDP? Express for all three cases the new state values in terms of $V(s), \gamma$ and the constants (where $V(s)$ is the optimal value of state s under the original reward function).

Now consider the following modifications for deterministic MDPs:

d) Let (s_{max}, a_{max}) be the state-action pair that leads to the highest possible immediate reward $r_{max} = max_{s,a} r(s, a)$ in the MDP. Set $r(s_{max}, a_{max}) \leftarrow r_{max} + d,\ d > 0$

e) Let (s_{min}, a_{min}) be the state-action pair that leads to the lowest possible immediate reward $r_{min} = min_{s,a} r(s, a)$ in the MDP. Set $r(s_{min}, a_{min}) \leftarrow r_{min} - d,\ d > 0$

For simplicity, you can assume in both cases that the minimum/maximum is unique, i.e. it is taken on exactly at one state-action pair. Can you guarantee for arbitrary MDPs that the optimal policy stays the same? If not, show a counterexample.

7 RL application: On- and off-policy learning [3 P]

Download the Reinforcement Learning (RL) MATLAB Toolbox and the example files[4]. Adapt the mountain car demo example and apply RL to the following learning task. Consider the gridworld shown in Figure 2. Implement this environment with the RL Toolbox as an undiscounted ($\gamma = 1$), episodic task with a start state at S and a goal state at G. The actions move the agent up, down, left and right, unless he bumps into a wall, in which case the position is not changed. The reward is -1 on all normal transitions, -10 for bumping into a wall, and 0 at the *bonus state* marked with B.

[4] http://www.igi.tugraz.at/lehre/intern/MountainCarDemo.zip

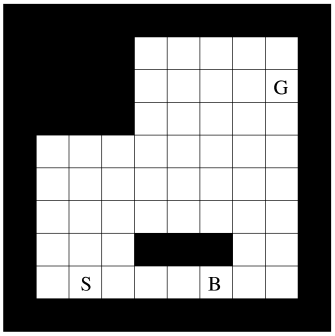

Figure 2: Gridworld with bonus state.

Use *Q-Learning* and *SARSA* without eligibility traces to learn policies for this task. Use ϵ-greedy action selection with a constant $\epsilon = 0.1$. Measure and plot the online performance of both learning algorithms (i.e. average reward per episode), and also sketch the policies that the algorithms find. Explain any differences in the performance of the algorithms. Are the learned policies optimal? Submit your MATLAB code.

Repeat the exercise again with $\epsilon \in \{0.01, 0.001, 0.0001, 0.00001\}$, respectively. Explain and interpret you results for both , i.e. *Q-Learning* and *SARSA*. Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortless.

8 RL game [3* P]

Consider the following game: You have a random number generator that produces in every round an integer number from 1 to 3 with equal probability. You play 3 rounds and have to decide at which position of a 3 digit number you want to place the random digit. Your goal is to form the largest possible (decimal) number. Formulate this game as a Markov decision process and find an optimal policy. Also analyze the case where the numbers are drawn *without replacement*, i.e. if the digit 3 appears in the first round, it cannot appear anymore in the remaining two rounds.

9 Policy Gradient Methods: Swimmer [4 P]

In this task you have to learn optimal policies for the swimmer (see Figure 3) using different policy gradient methods. You have to compare two algorithms to compute the gradient $\nabla_\theta J(\theta_h)$, namely Finite Differences and Likelihood Ratio. The robot is a 3-link (2-joints) snake-like robot swimming in the water. It has

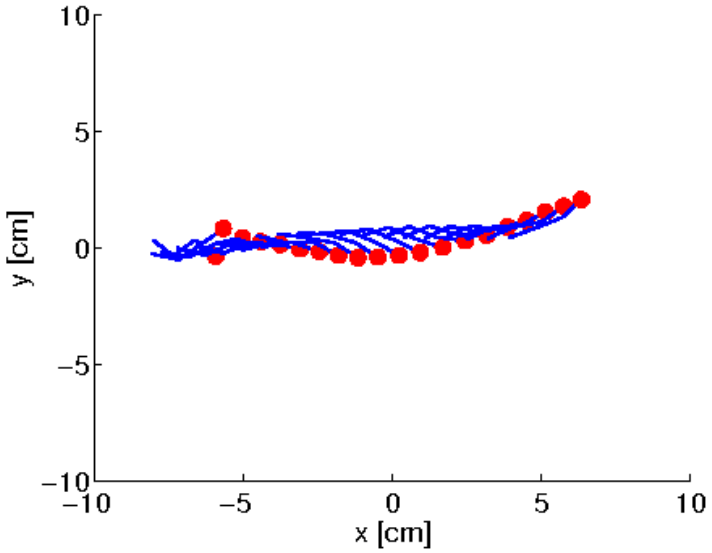

Figure 3: 3 link wwimmer task: The simulated snake-like robot should swim fast and energy efficient.

two actuators, you have to learn how to use these actuators to swim as fast as possible in a given direction. The model, the policy and the reward function are already given in the provided matlab package `swimmer.zip`[5]. The used policy is a stochastic Gaussian policy implemented by a Dynamic Movement Primitive (DMP). The DMP uses 6 centers per joint. As we deal with a periodic movement the phase variable x of the DMP is also periodic. The policy itself is a stochastic policy which adds noise to the velocity variable of the DMP, i.e.

$$\pi(\dot{\mathbf{y}}|\mathbf{z}, x; \mathbf{b}) = \mathcal{N}(\dot{\mathbf{y}}|\Phi(x)\mathbf{b} + \mathbf{z}, \sigma^2\mathbf{I}),$$

where $\mathbf{b}$ is the parameter vector (also denoted as θ in the further description) of the policy which we have to learn. The DMP itself is already implemented so you do not have to deal with that, the MATLAB package provides you with all information you need to calculate the gradients. The reward function (already implemented) is given by $r_t = 10^{-2}v_x - 10^{-6}\mathbf{u}^2$, where v_x is the velocity in x-direction and $\mathbf{u}$ is the used torque.

9.1 MATLAB package description

In order to work with the swimmer model add the folder *model* to your MATLAB path. The model environment is stored in a structure, which is created with the command $E = initE()$. For implementing the policy gradient methods, the most

[5]http://www.igi.tugraz.at/lehre/MLB/WS12/intern/swimmer.zip

important function is the function $E.J$ to perform a single rollout

$$[perf, \epsilon, \phi, rewards, ...] = E.J(E, \theta, \sigma)$$

[6]. The function takes the model structure E, the policy parameters θ (i.e. the linear parameters $\mathbf{b}$ in our case) and the variance of the stochastic policy σ as arguments. The function simulates the swimmer using θ as parameters of the policy for 200 time steps ($dt = 0.01s$ resulting in a simulation time of $2s$) and returns the summed reward (perf) for this episode ($\sum_t r_t$). In addition it returns the used noise vector ϵ for each time step (so ϵ is a 2×200 matrix) and the features $\phi(x_t)$ for each timestep (6×200 matrix). The single rewards for each time step can also be obtained (rewards). The E.J function has additional output values which return the visited trajectory, the performed torques and the state variables of the dmp (y and $\dot{y}$), see *evaluate.m* for further details. To visualize a policy use $plotPolicy(E, \theta)$.

Finally some general remarks: The policy θ has $(E.d - 1) \times E.NumRBFs$ parameters, where $d = 3$ denotes the number of links of the swimmer. The number of Gaussian kernel functions is given by the model and set to 6. Make sure, that the parameters are within the interval $[+4, -4]$.

9.2 Policy Gradient Methods

At least for the finite difference method, you may normalize the gradient as a unit vector before the weight update, i.e.

$$\delta\theta_{h+1} = \theta_h + \alpha \frac{\nabla_\theta J(\theta)}{|\nabla_\theta J(\theta)|}.$$

This usually improves the learning speed. For a more exact description of the methods and the equations see the lecture slides.

- Finite Differences [2 P]: This method perturbs the parameter vector itself to estimate the gradient. We can set the stochasticity of the policy to 0 for this method to improve the accuracy of the estimation. Generate I (try 24) rollouts by adding small Gaussian Noise to the parameter vector $\Delta\theta_i = \mathcal{N}(0, \sigma_{FD}^2 I)$, where σ_{FD} equals small values like 0.5. Try several learning rates α (Hint: Setting α to 2 is a good starting point).

- Likelihood ratio + Policy Gradient Theorem [2 P]: Here the noise directly acts on the action, therefore we use a stochastic policy (use $\sigma = 0.5$ which can be set in the E.J function). For both methods you need to be able to calculate the gradient of the log-likelihood of the policy for each time step, e.g. $\nabla_\theta \log \pi(\dot{\mathbf{y}}_t | \mathbf{z}_t, x_t; \mathbf{b})$. As we can see in the lecture slides the only two

[6] This function points to the file *evaluate.m*

quantities needed for this operation are the used noise ϵ_t and the feature representation $\Phi(x_t)$. Both quantities are provided by the function $E.J$ [7]. For the policy gradient theorem the reward r_t for each time step is needed instead of the summed reward signal. This vector is given by the 4-th output value of the $E.J$ function. 12 rollouts and a learning rate of $\alpha = 10$ are proper choices.

All policy gradient methods should be compared with respect to learning speed. Therefore, create a performance curve (x-axis : number of episodes seen by the algorithm, y-axis: summed reward of current parameter value) for each algorithm. In order to get a reliable estimate, use the average over at least 10 trials for each curve.

10 Bayesian networks [2+2* P]

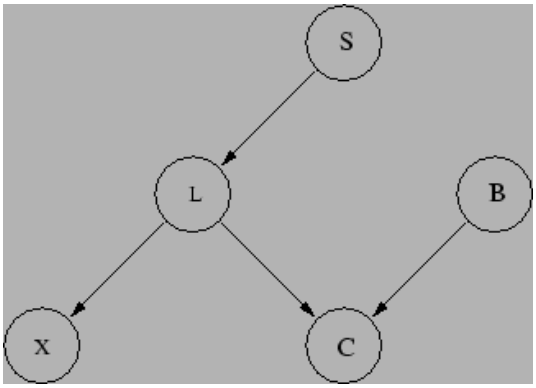

Figure 4: Graphical model for diagnosis example.

a) [2 P] Consider the graphical model in Figure 4 for a medical diagnosis example, where B = bronchitis, S = smoker, C = cough, X = positive X-ray, and L = lung cancer.

List the pairs of nodes that can be proven to be conditionally independent with the definition of d-separation, given the following evidence:

1) [1/2 P] No evidence for any of the nodes.

2) [1/2 P] The lung cancer node is set to true (and no other evidence).

[7] Note that in the lecture slides the gradient $\log \pi(\dot{\mathbf{y}}_t|\mathbf{z}_t, x_t; \mathbf{b})$ is given by $\propto \epsilon\Phi(x)^T$, which is typically a matrix not a vector. The gradient vector can be obtained by transforming the matrix to a vector $[\epsilon_t(1)\Phi(x_t); \epsilon_t(2)\Phi(x_t)]$

3) [1/2 P] The smoker node is set to true (and no other evidence).

4) [1/2 P] The cough node is set to true (and no other evidence).

b) [2* P] In your local nuclear power plant station, there is an alarm that senses when a temperature gauge exceeds a given threshold. The gauge measures the temperature of the core. Consider the Boolean variables A (alarm sounds), F_A (alarm is faulty), and F_G (gauge is faulty) and the multivalued nodes G (gauge reading) and T (actual core temperature).

1. Draw a Bayesian network for this domain, given that the gauge is more likely to fail when the core temperature gets too high.
2. Suppose there are just two possible actual and measured temperatures, normal and high; the probability that the gauge gives the correct temperature is x when it is working, but y when it is faulty. Give the conditional probability table associated with G.
3. Suppose the alarm works correctly unless it is faulty, in which case it never sounds. Give the conditional probability table associated with A.

11 Approximate inference in Bayesian networks [4 P]

Apply Gibbs sampling to carry out approximate inference in Bayesian networks. You should estimate the (marginal) probability distribution of several variables in a Bayesian network, given the settings of a subset of the other variables (evidence). Implement the Gibbs algorithm in MATLAB based on the code provided `Gibbs.zip`[8] and test it on the three Bayesian networks shown below. Your code should run Gibbs sampling a specified number of iterations in order to estimate the required probability distributions. Since Gibbs sampling is a randomized, iterative algorithm, the actual number of iterations needed to estimate probabilities is an important issue, that will be explored as part of this assignment.

a) Download the MATLAB code and modify the file `gibbs.m` to implement Gibbs sampling. The file loads one of six predefined Bayesian networks that are stored in the files `alarm.mat, insurance.mat, leader-follower_05.mat, ... leader-follower_20.mat`. Each of these data files contains the two variables `var` and `p` that specify the random variables and the factors of the joint probability distribution for each Bayesian network, respectively.

[8] http://www.igi.tugraz.at/lehre/MLB/WS12/intern/Gibbs.zip

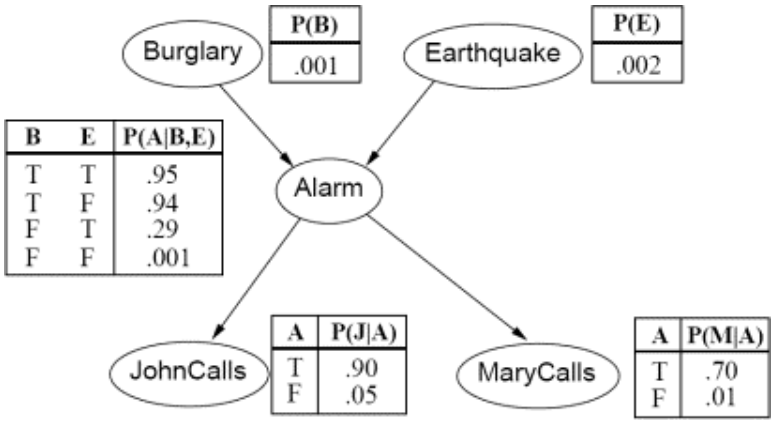

Figure 5: A Bayesian network establishing relations between events on the burglary-earthquake-alarm domain, together with complete specifications of all probability distributions.

b) As a sanity check, to be sure that your code is working, start by running Gibbs sampling on the "earthquakes and burglar alarms" Bayesian network (Fig. 5) provided in the file `alarm.mat`.

   1. Run Gibbs sampling 1000 times for 2000 steps starting each time from random intial values for the hidden variables (drawn from a uniform distribution) and verify that the conditional probability of *burglary* given that both *John* and *Mary* call is approximately 0.284. In order to estimate this conditional probability calculate the mean and the SEM (standard error of the mean) for the 1000 values of the binary random variable that is one if *burglary* = *true* and 0 otherwise (after 2000 steps).
   2. Suppose now that you learn that there was an earthquake in the area. Run Gibbs sampling to estimate the probability of a burglary (given that John and Mary both called, and that there was an earthquake). Interpret the result.

c) The Bayesian network illustrated in Fig. 6 attempts to estimate the risk posed by an individual seeking car insurance. In other words, the network attempts to relate variables like the age and driving history of the individual to the cost and likelihood of property damage or bodily injury.

   The network has 27 variables (usually the 12 shaded variables are considered hidden or unobservable, while the other 15 are observable) and over 1400 parameters. An insurance company would be interested in predicting the bottom three "cost" variables, given all or a subset of the other observable variables. The network has been provided in the file `insurance.mat`.

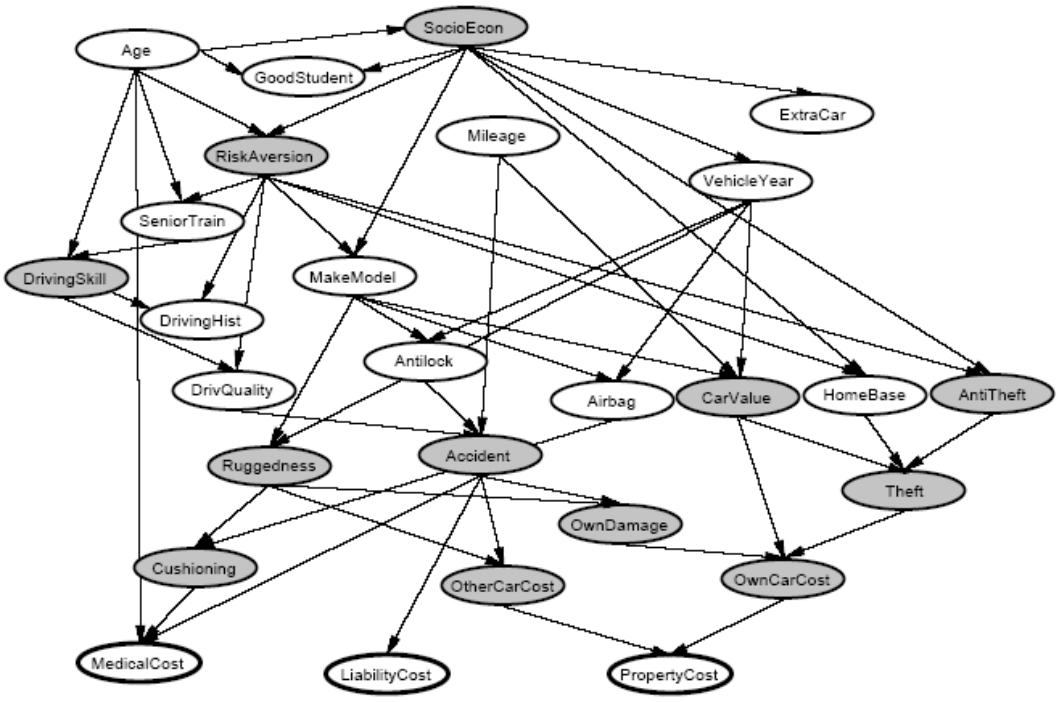

Figure 6: A Bayesian network establishing relations between events on the automobile insurance domain.

1. You should estimate the cost of property damage, i.e. the probabilities of the values of the variable *PropertyCost*, for an adolescent driving a car (*Age* = *Adolescent*) with about 50,000 miles (*Mileage* = *FiftyThou*) and no anti-lock brakes (*Antilock* = *False*).

   Run Gibbs sampling for a 1000 times for 1, 10, 100, and 1000 steps starting from random intial values for the hidden variables (drawn from a uniform distribution) and plot the dependence of the resulting probability distribution (obtained for the 1000 trials) on the number of steps. Because some of the factors of the joint probability distribution are 0 you have to make sure that the probability of the intial state of the random variables is larger than zero (if not choose another initial state).

   How long ist the burn-in time (number of steps) that is required to sample from the stationary probability distribution?

2. Try out at least one other query of your own invention and report the results. Comment on why your results did or did not make sense.

d) Explore the convergence properties of Gibbs sampling on a very simple family of Bayesian networks. Theoretically, Gibbs sampling will converge

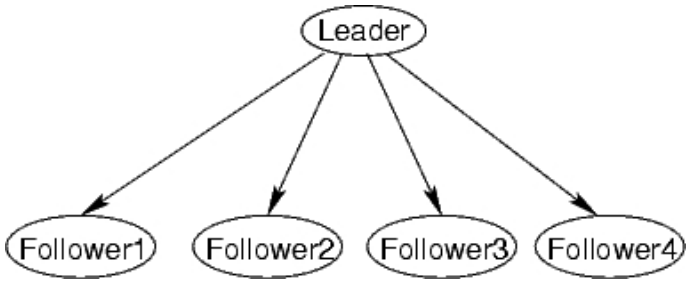

Figure 7: A Bayesian network establishing relations between events on the leader-follower domain.

to the correct stationary probability distribution asymptotically, that is, as the number of iterations becomes very large. However, in practice, it is not always clear how many iterations actually suffice, as this example demonstrates.

Each network in this family has a single "leader" variable, and some number of "follower" variables as shown in Fig. 7. The idea is that the leader selects a bit at random (0 or 1 with equal probability), and then all of the followers choose their own bits, each one copying the leader's bit with 90% probability, or choosing the opposite of the leader's bit with 10% probability. Leader-follower networks with k followers have been provided in the files `leader-follower-k.bn` for $k = 5, 10, 15, 20$.

What is the marginal probability distribution for the leader variable, in the absence of any evidence (i.e., none of the other variables have been set to specified values)?

1. What is the correct answer for the query given above? (This should be easy to answer.)
2. Run Gibbs sampling on a network with 5 followers using the given query. Run for at least 1000000 steps, printing the partial results every 1000 steps. Make a plot of the estimated probability (as estimated by Gibbs sampling) of the leader choosing the bit 0 as a function of the number of steps. In other words, the x-axis gives the number of steps, and the y-axis gives the probability of the leader bit being 0 as estimated after t steps of Gibbs sampling. Does Gibbs sampling seem to converge to the right answer? If so, roughly how long does it seem to take?
3. Now repeat the last question on networks with 10, 15 and 20 followers. Run for at least 1000000 steps, printing partial results at least every 1000 steps.

4. Can you explain what you observed? Why does Gibbs sampling have such a hard time with these networks? What do these experiments say about the difficulty of detecting convergence?

Submit your MATLAB code. Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortless.

12 Learning overhypotheses [3* P]

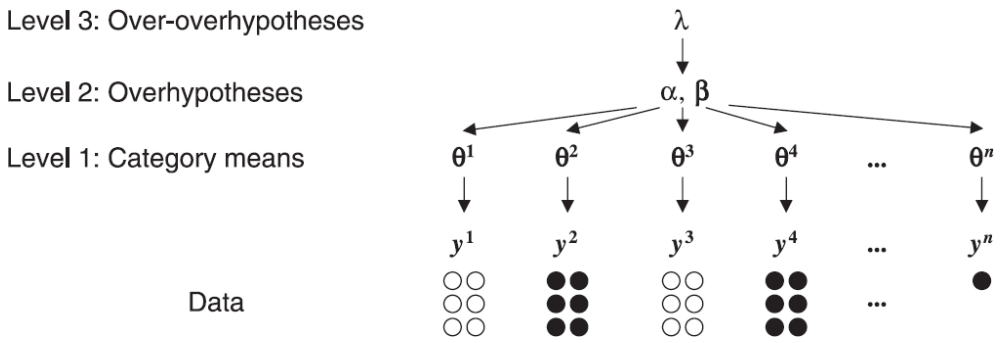

Figure 8: A hierarchical Bayesian model. Each setting of (α, β) is an overhypothesis: β represents the color distribution across all categories, and α represents the variability in color within each category.

Apply Gibbs sampling to acquire overhypotheses about the feature variability for the bags of marbles model illustrated in Fig. 8: Suppose that S is a stack containing many bags of marbles. We empty several bags and discover that the marbles within the same bag have certain features in common: For instance some bags may contain black marbles, others may contain white marbles, but that the marbles in each bag are uniform in color. Given a new bag - bag n - and a single marble (e.g. a black marble) drawn from this bag we are interested in the probability of the colors of all other marbles within this bag. On its own, a single draw would provide little information about the contents of the new bag, but experience with previous bags may lead us to endorse certain hypothesis (e.g. all marbles in a bag have uniform colors).

Learning overhypothesis: The term overhypothesis is used to refer to any form of abstract knowledge that sets up a hypothesis space at a less abstract level. By this criterion, an overhypothesis sets up a space of hypotheses about the marbles in bag n: they could be uniformly black, uniformly white, and so on. Hierarchical Bayesian models capture the notion of overhypothesis by allowing hypothesis spaces at several levels of abstraction. In this example we wish to explain how a certain kind of inference can be drawn from a given body of data.

In this case, the data are observations of several bags and we are working with a set of 2 colors.

Bags of marbles model: Let $\mathbf{y}^i$ indicate a set of observations of the marbles in bag i. If we have drawn five marbles from bag 7 and all but one are black, then $\mathbf{y}^7 = [4, 1]$. We are interested in the ability to predict the color of the next marble to be drawn from bag n. The first step is to identify a kind of knowledge (level 1 knowledge) that explains the data and that supports the ability of interest. In this case, level 1 knowledge is knowledge about the color distribution of each bag. Let θ^i indicate the true color distribution for the ith bag in the stack. We assume that $\mathbf{y}^i$ is drawn from a binomial distribution with parameter θ^i: in other words, the marbles responsible for the observations in $\mathbf{y}^i$ are drawn independently at random from the ith bag, and the color of each depends on the color distribution θ^i for that bag. If 60% of the marbles in bag 7 are black, then $\theta^7 = [0.6, 0.4]$.

For the marbles scenario, level 2 knowledge is knowledge about the distribution of the θ variables. This knowledge is represented using two parameters, α and β. The vectors θ^i are drawn from a Beta distribution parameterized by a scalar α and a vector $\beta = (\beta_1, \beta_2)$ with $\beta_1 + \beta_2 = 1$. The parameter α determines the extent to which the colors in each bag tend to be uniform, and β represents the distribution of colors across the entire collection of bags. We need to formalize our a priori expectations about the values of these variables.

Level 2 knowledge is acquired by relying on a body of knowledge at an even higher level, level 3. We use a uniform distribution on β_1 and an exponential distribution on α, which captures a weak prior expectation that the marbles in any bag will tend to be uniform in color. The mean of the exponential distribution is $\lambda = 1$, i.e. $P(\alpha) = exp(-\alpha)$. The parameter λ and the pair (α, β) are both overhypotheses, since each sets up a hypothesis space at the next level down. Since the level 3 knowledge is specified in advance (λ), you should analyze how an overhypothesis can be learned at level 2.

The joint probability distribution for this model is therefore given by

$$P(\mathbf{y}^1, ..., \mathbf{y}^n, \theta^1, ..., , \theta^n, \alpha, \beta|\lambda) = \prod_{i=1}^{n} P(\mathbf{y}^i|\theta^i)P(\theta^i|\alpha, \beta)P(\alpha|\lambda)P(\beta) \tag{1}$$

with

$$\begin{aligned} \alpha &\sim Exponential(\lambda) && (2)\\ \beta_1 &\sim Beta(1, 1) && (3)\\ \theta^i &\sim Beta(\alpha\beta_1, \alpha\beta_2) && (4)\\ \mathbf{y}^i|n^i &\sim Binomial(\theta^i) && (5) \end{aligned}$$

where n^i is the number of observations for bag i.

Task: You should perform Gibbs sampling for this joint distribution to estimate the marginal distributions for each single θ^i, α and β (the overhypotheses) where the observation $\mathbf{y}^i$ are kept fixed for the following scenarious:

1. After observing 10 all-white bags, 10 all-black bag and a single black marble in the last bag.
2. After observing 20 mixed bags, where half of the marbles are white and half of the marbles are black, and a single black marble in the last bag.
3. Same as in 1 but with fixed $\alpha = 1$ and $\beta_1 = \beta_2 = 0.5$.
4. Same as in 2 but with fixed $\alpha = 1$ and $\beta_1 = \beta_2 = 0.5$.

Calculate average distributions across 50 Markov chains, each of which was run for 100000 iterations (discard the first 10000 samples as burn-in). Hand in plots for all estimated distributions and interpret your results.

Hints:

1. For the resampling of θ^i the factor $P(\mathbf{y}^i|\theta^i)P(\theta^i|\alpha,\beta)$ is again a Beta distribution that can be directly sampled in MATLAB with the command `random` and the argument `beta`.
2. For the resampling of α and β apply sampling-importance-resampling sampling (with 100 samples drawn from a proposal distribution that is identical to the prior distribution) from a distribution proportional to the factor $P(\theta^i|\alpha,\beta)P(\alpha|\lambda)P(\beta)$.
3. You should adapt your own MATLAB code from the previous Gibbs sampling homework example to solve this assignment.

Submit your MATLAB code. Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortless.

13 Planning with Approximate Inference [3 P]

In this example we consider a 2-link planar arm. In this task you have to plan an optimal path from an initial joint-position to a given endeffector-position [9] by Gibbs sampling.

The length of the links l_1 and l_2 is given by 0.5 meters. We will restrict the first joint position of our arm model to be in the range of $[0; \pi/2]$ and the second to be in the range of $[-\pi/2; \pi/2]$.

[9] The endeffector position is given by $[x, y]^T = [l_1 sin(q_1) + l_2 sin(q_1 + q_2), l_1 cos(q_1) + l_2 cos(q_1 + q_2)]$, where q_i denotes the joint angle of the ith joint.

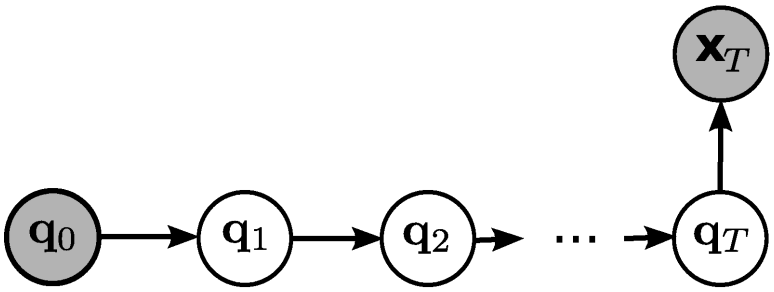

Figure 9: Dynamic Bayesian Network for task-space planning. The task-space is defined as cartesian coordinates of the endeffector position.

We will first discuss how to construct a Bayesian Network where we can sample from. We want to reach our target within $T = 10$ time steps, i.e. we get a dynamic Bayesian Network with one node per time step (11 nodes), see Figure 9.

Each node t represents the joint positions $\mathbf{q}_t$ at time t. For simplicity, we will use a discrete representation of the joint positions. Therefore we use a uniform 11×11 grid to discretize the joint space. We will use a Gaussian motion prior in order to define the transition probabilities of $P(q_t^{(j)}|q_{t-1}^{(i)})$ from the ith discrete joint position at time $t-1$ to the jth joint position at time t. Let $\mathbf{q}^(j)$ be the j joint position vector (in radians), then $P(\mathbf{q}_t^{(j)}|\mathbf{q}_{t-1}^{(i)}) \propto \mathcal{N}(\mathbf{q}_t^{(j)}|\mathbf{q}_{t-1}^{(i)}, \mathbf{W})$, where W equals $\text{diag}([0.0125, 0.05])$. The motion prior encodes our laziness, meaning that, if not necessary, we do not want to move away from $\mathbf{q}_{t-1}$.

In order to plan a trajectory to a certain end-effector position we still need to define our kinematic task space mapping. We will also use a discrete representation for task space (Cartesian coordinates of the hand). Here, we use again a 11×11 uniform grid over the range $[0; 1]$ for x and y. The probability of reaching the jth discrete task space position when being in the ith discrete joint space position is given by $P(x_t^{(j)}|q_t^{(i)}) \propto \mathcal{N}(x_t^{(j)}|\Phi(q_t^{(i)}), \mathbf{C})$, where Φ is the non-linear mapping from the joint positions to the endeffector coordinates. The covariance matrix $\mathbf{C}$ is set to $\text{diag}([0.004, 0.004])$.

13.1 Task Space Planning

We will first start with task space planning. Therefore we add an 'mental observation' of reaching the jth discrete position in task space $x_T = x^{(j)}$ at time $T = 10$ to our Bayesian network (see Figure 9). We will set our desired target position to be $[0.2, 0.2]$, the index j denotes the discrete index of this position. In addition, we also observe our current state $\mathbf{q}_0$ which is $[\pi/4, 0]^T$. Our task is to estimate a trajectory $\mathbf{q}_{1:T}$ using Gibbs-sampling.

- Your main task is to generate the state transition probabilities P_{qq} as well

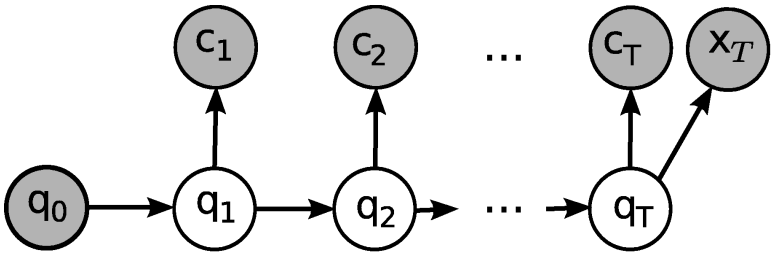

Figure 10: Dynamic Bayesian Network for planning with obstacles.

as the kinematic task mapping P_{qx} as described in the text.

- Initialize your trajectory $\mathbf{q}_{1:T}$ with random discrete indices. Use 5000 Gibbs sampling steps to estimate the true distribution. Plot 5 independent samples of $\mathbf{q}_{1:T}$ from the sampling process. We assume that two samples are independent at least after 100 Gibbs Sampling steps.

- Now we want to calculate the marginals $P(\mathbf{q}_t = q^{(i)})$ for all i and t. Therefore count the number of of times in which the variable $\mathbf{q}_t$ equals $q^{(i)}$ during the last 3000 Gibbs sampling steps. Plot the marginals for each time step, use a visualization which you find appropriate. How does the estimated solution look like?

- Repeat the experiment at least 10 times using different initial positions for the Gibbs sampler. Are the marginals always similar (up to a certain noise level...)? If not, why not?

13.2 Task Space Planning with Obstacle Avoidance

Now we want to add an obstacle to our environment. The obstacle is located at $[0.5, 0.5]$ in Cartesian space. For simplicity, we assume that only the end-effector can collide with the obstacle. The collision probability $P(c_t = 1|q_t^{(j)})$ for joint position $\mathbf{q}^{(j)}$ is given by $\exp(-1/2||\Phi(\mathbf{q}^{(j)}) - [0.5, 0.5]^T||^2/0.15^2)$. We want our robot to avoid the obstacle for the whole trajectory, therefore we add the observation of not colliding with the obstacle $P(c_t = 0|q_t^{(j)})$ for each time step t to our Bayesian network (see Figure 10, c_t notes).

- Generate the collision probability P_{qc}

- Use Gibbs sampling the same ways as before, visualize the marginals $P(\mathbf{q}_t = q^{(i)})$. How has the estimated solution changed?

13.3 Pseudo-Dynamic Planning with Obstacle Avoidance (Optional Task)

Try the approach also on a dynamic task.

Now we also want to add the velocities $\dot{\mathbf{q}}$ of the joints to our planning scenario. Therefore, we will also incorporate controls $\mathbf{u}$ of the robot in our model. The controls $\mathbf{u}$ directly represent the accelerations of the joints. The control-dependent state transitions are now given by

$$P(\mathbf{q}_t, \dot{\mathbf{q}}_t | \mathbf{q}_{t-1}, \dot{\mathbf{q}}_{t-1}, \mathbf{u}_{t-1}) = \mathcal{N}([\mathbf{q}_t; \dot{\mathbf{q}}_t] | [\mathbf{q}_{t-1} + 0.1\dot{\mathbf{q}}_{t-1}; \dot{\mathbf{q}}_{t-1} + 0.1\mathbf{u}_{t-1}], \mathbf{W}),$$

where $\mathbf{W}$ is set to $\text{diag}([10^{-5}, 10^{-5}, 10^{-3}, 10^{-3}])$. Now, in difference to the previous tasks we incorporated controls to our model. For each dimension we will use 5 discrete actions $u_{1,2} \in [-4, -2, 0, 2, 4]$, resulting in a action space of 25 actions. The actions are unknown, and hence, like every unknown hidden variable, they can be integrated out : $P(\mathbf{q}_t, \dot{\mathbf{q}}_t | \mathbf{q}_{t-1}, \dot{\mathbf{q}}_{t-1}) = \sum_{i=1}^{25} P(\mathbf{q}_t, \dot{\mathbf{q}}_t | \mathbf{q}_{t-1}, \dot{\mathbf{q}}_{t-1}, \mathbf{u}_{t-1}^{(i)}) P(\mathbf{u}_{t-1}^{(i)})$. The term $P(\mathbf{u}_{t-1}^{(i)})$ denotes the action prior, similarly to the previous example we again use it to code our laziness, i.e. we prefer doing no action at all $P(\mathbf{u}_{t-1}^{(i)}) = \mathcal{N}(\mathbf{u}_{t-1}^{(i)} | 0, \mathbf{H})$, where $\mathbf{H}$ is set to $16\mathbf{I}$.

As we can see the controls are excluded from the inference process, however, they can be easily calculated from an estimated trajectory $[\mathbf{q}_{1:T}, \dot{\mathbf{q}}_{1:T}]$. We will again use a discretization of the state space with a $11 \times 11 \times 11 \times 11$ uniform grid. Valid velocities are in the range of $[-1; 1]$

- Create the state transition probabilities P_{qq} for the dynamic case. Also create the task space mapping P_{qx} and the collision mapping (both do not depend on the velocities). Use the same intial state and target end-effector position as before, but set the velocity to zero.

- Again use Gibbs sampling to sample from valid trajectories. Now use 50000 Gibbs sampling steps and 1000 steps between two independent samples. Again calculate the marginals and visualize the marginals of the positions $\mathbf{q}_t$ for each time step.

- In addition, plot the expected positions and velocities for each joint over time.