2D1431 Machine Learning
Exam 2002-02-14 8:00-13:00 L21-22

Frank Hoffmann
e-mail: hoffmann@nada.kth.se

December 20, 2002

No books or other study aids allowed!

Grading: U : 0-22p, 3 : 23-28p, 4 : 29-34p, 5 : 35-40p

Question 1 (2p)
What is meant by the term *over-fitting* in the context of inductive learning? What can cause over-fitting when training an artificial neural network on training data? How can one avoid over-fitting?

Over-fitting means that an hypothesis h has a larger true error over unseen instances than another hypothesis h', even though its training error is lower than that of h'.
In ANN over-fitting is usually caused by a combination of the following factors

- noisy data (including outliers and irrelevant attributes)
- large number of hidden neurons
- large number of training cycles

Over-fitting can be avoided by means of cross-validation, namely to partition the data into training and validation set and to pick the hypothesis that has the lowest error on the validation data.

Question 2 (3p)
Backpropagation and instance based learning are two learning methods that were used in lab 3. Explain the key concepts, objectives and principles of both learning algorithms. Describe how both methods approximates a real-valued target function. In what way are their mechanisms for regression similar and in what way do they differ?

BP is used to train multi-layer perceptrons in finding a set of weights that minimize the error between target value and network output. Instance based learners store training examples and predict the value of an unseen instance based on its similarity to stored training instances.
BP and instance based learning are similar in that they minimize the quadratic error between the target value and the output of the learner. BP tries to globally approximate the target function, by means of gradient descent on the error function. Instanced based learners locally approximate the target function, in that it weights the error of a training instances by its distance to the query point. Instance based learner form a hypothesis that is locally optimal for the query point. Instance based learning is a lazy learner in that it delays building an hypothesis until a query is made, whereas BP is an eager learner with an explicit hypothesis representation. Usually the multi-layer neural network used by BP is a non-linear model, whereas instance based learner often (not necessarily though) use a constant or linear model. Instance based learner classify or predict the value of an unseen instance based on the target values of similar instances in the training set. Instance based learning is suitable for online learning, BP is usually an off-line learning algorithm. Radial basis function networks are a mixture between conventional multi-layer perceptrons and locally weighted regression.

Question 3 (2p)
What is the difference between pattern recognition/classification and function approximation/regression? Describe a concrete example of each learning task and explain which learning algorithm you would use to solve the task.

Pattern recognition is concerned with discrete target function. An instance x is assigned to a discrete class c_i, the goal is to correctly classify instances, the error is either 0 or 1. Function approximation is concerned with learning a continuous function, that is minimize the error, namely the absolute dif-

ference $|\hat{f}(x) - f(x)|$ between the output of the learner $\hat{f}(x)$ and the target value $f(x)$. Each instance x is associated with a real value $f(x)$. Example for classification, use a decision tree to accept or reject a credit card application, possible classes (accept, reject). Or object recognition, assign an image of an object to a set of possible classes (e.g. apple, peach, mango, banana). Classify the MONK dataset with a decision tree (lab 1). Example for function approximation, real estate value appraisal, given attributes of a real estate (size of estate, number of rooms, location, etc.) predict the market value of this house. Or approximate Mexican hat function from a set of training instances (lab 3).

Question 4 (2p)
What does Bayes Theorem state? Describe the meaning of Bayes Theorem in words. What is the *maximum a posteriori* (MAP) hypothesis and how can you calculate it using Bayes theorem? What is the *maximum likelihood* (ML) hypothesis and under what condition is it identical with the MAP hypothesis?

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{1}$$

P(D) is the prior probability of the data (evidence)
P(h) prior probability of hypothesis (prior)
P(D—h) probability of the data D given h, (likelihood)
P(h—D) posterior probability of the hypothesis given the data (posterior)

The MAP hypothesis is the one that maximizes $P(h|D)$ it can be computed by

$$h_{MAP} = argmax_h P(h|D) = argmax_h P(D|h)P(h) \tag{2}$$

as the normalization factor $P(D)$ is the same for all h.
The maximum likelihood hypothesis is given by

$$h_{ML} = argmax_h P(D|h) \tag{3}$$

the MAP and ML hypothesis are identical if all priors $P(h)$ are equal, and assumption that is often made if priors are not available.

Question 5 (5p)
Construct the decision tree for the set of training instances in the left table with attributes x_1, x_2, x_3 and classes $+, -$. Ignore the attribute x_3 for making splits and only build a decision tree that includes nodes for attributes x_1 and x_2. Which attribute should be used at the root node and why? Show how the examples $D_1, \ldots, D_{10}$ are sorted down the tree and how they are classified. Use the right table to look up the entropy for a subset of training instances. Take the entry that is closest to the decimal number for which you want to calculate the entropy (e.g. if you need to calculate the entropy of 0.33, take the entry for 0.3 in the table which is 0.9). It is sufficient to make approximate calculations rounded to one digit behind the decimal point.

No.	x_1	x_2	x_3	$c(x_1, x_2)$
D_1	0	0	1	+
D_2	0	0	2	+
D_3	0	0	3	+
D_4	0	0	4	+
D_5	0	1	1	-
D_6	0	1	2	-
D_7	0	1	3	-
D_8	1	0	4	-
D_9	1	1	1	+
D_{10}	1	1	2	+

Training examples

p	entropy(p)
0.0	0.0
0.1	0.5
0.2	0.7
0.3	0.9
0.4	1.0
0.5	1.0
0.6	1.0
0.7	0.9
0.8	0.7
0.9	0.5
1.0	0.0

First compute the entropy of the overall data set D (6+,4-), so p=0.6, E(D)=1.0.
When we split on x_1 we obtain two subsets $D_{10}\text{for}(\text{x}_1 = 0) = (4+, 3-)$ and $D_{11}\text{for}(\text{x}_1 = 1) = (2+, 1-)$ Their entropy is $E(D_{10}) = E(p = 4/7) = 1.0$ and $E(D_{11}) = E(p = 2/3) = 0.9$
The information gain for attribute x_1 is computed by

$$G(x_1) = E(D) - \frac{|D_{10}|}{D}E(D_{10}) + \frac{|D_{11}|}{D}E(D_{11}) \quad (4)$$

$$= 1.0 - 7/10 * 1.0 - 3/10 * 0.9 = 0.03 \quad (5)$$

The same calculation applies for attribute x_2. When we split on x_2 we obtain two subsets $D_{20}\text{for}(\text{x}_2 = 0) = (4+, 1-)$ and $D_{21}\text{for}(\text{x}_2 = 1) = (2+, 3-)$ Their entropy is $E(D_{20}) = E(p = 4/5) = 0.7$ and $E(D_{21}) = E(p = 2/5) = 1.0$

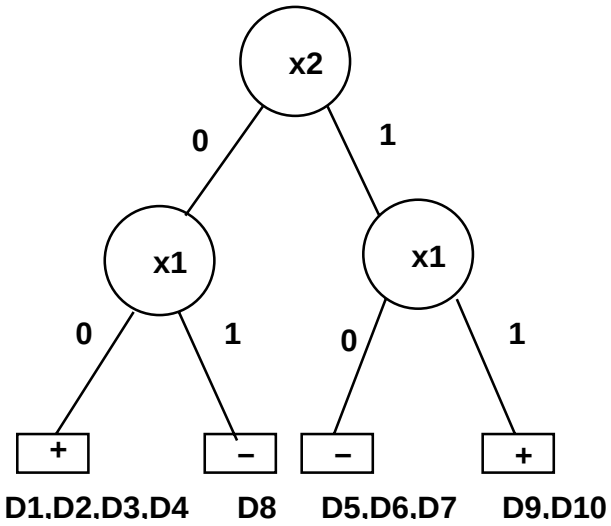

Figure 1: Decision tree

$$G(x_2) = E(D) - \frac{|D_{20}|}{D}E(D_{20}) + \frac{|D_{21}|}{D}E(D_{21}) \quad (6)$$

$$= 1.0 - 5/10 * 0.7 - 5/10 * 1.0 = 0.15 \quad (7)$$

Therefore attribute x_2 should be used at the root node. The examples are sorted down the tree as shown in figure 1

Question 6 (2p)
Explain what is meant by PAC-learnable. What is the definition of PAC-learnability for a concept class? What is the sample complexity of a learning problem and how does it relate to the notion of PAC-learnability?

PAC stands for probably approximately correct. Probable means that the learner finds with probability (1-δ) a hypothesis that is approximately correct, which means it has a true error smaller than ϵ or smaller than training error plus ϵ. The learner should be able to find such a hypothesis in time polynomial in $1/\epsilon$, $1/\delta$ and the number of training instances. Sample complexity describes the number of examples needed to learn an approximately correct hypothesis with high probability. The higher the probability and the lower the true error the more training examples are needed.

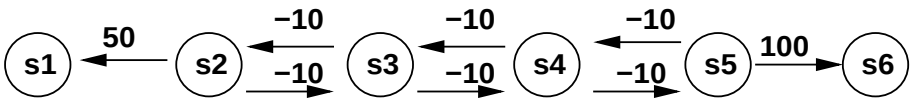

Figure 2: Environment for finite horizon problem: s_1 and s_6 are terminal states . The arrows indicate the possible actions $\leftarrow, \rightarrow$ in each of the non-terminal states $s_2, \ldots, s_5$. The numbers above the arrows show the rewards $r(s_i, \leftarrow)$, $r(s_i, \rightarrow)$ associated with states s_i and actions $\leftarrow$, $\rightarrow$.

Question 7 (2p)
Assume the following scenario. You are witness of an accident at night time involving a taxi in Athens. In Athens all taxis are either green or blue. You swear under oath that the taxi that you saw was blue. Extensive testing shows that under the dim lighting conditions, discrimination between blue and green is reliable 75% of the time. What is the most likely color of the taxi given no other information? What is the probability that the taxi is blue if you would know in addition that 80% of the taxis in Athen are green and 20% of the taxis are blue.

Given no other information, we have to assume that the prior probabilities for observing a green or blue taxi are equal, so we have to take the maximum likelihood hypothesis $h_{ML} = argmax_h P(D|h)$ that generates the highest likelihood of the data. P(taxi is green—witness sees blue)=0.25, P(taxi is blue—witness sees blue)=0.75, therefore the ML hypothesis is blue. If we know the priors P(h) we take them into account to compute the MAP hypothesis $h_{MAP} == argmax_h P(h|D) = argmax_h P(D|h)P(h)$. P(taxi is green—witness sees blue) * P(green=0.25*0.8=0.2, P(taxi is blue—witness sees blue) * P(blue)=0.75*0.2=0.15. So the MAP hypothesis is green. If we normalize accordingly the probability that the taxi is blue is $P = 0.15/(0.2 + 0.15) = 3/7$.

Question 8 (5p)
Consider the environment of a chain of six states $s_1, \ldots, s_6$ as shown in figure 2. The states s_1 and s_6 are absorbing terminal states. In each non-terminal state, the agent can choose between two alternative actions $\leftarrow, \rightarrow$ that bring the agent to the neighboring state. The deterministic transition function for the non-terminal states $s_2, \ldots, s_5$ is given by

$$\delta(s_i, \rightarrow) \quad = \quad s_{i+1} \text{ for } i \in \{2, 3, 4, 5\}$$

$$\delta(s_i, \leftarrow) \quad = \quad s_{i-1} \text{ for } i \in \{2,3,4,5\} \tag{8}$$

The deterministic reward function is indicated by the numbers on the arrows corresponding to action $\leftarrow, \rightarrow$.

$$\begin{aligned} r(s_i, \rightarrow) &= -10 \text{ for } i \in \{2,3,4\} \\ r(s_i, \leftarrow) &= -10 \text{ for } i \in \{3,4,5\} \\ r(s_2, \leftarrow) &= 50 \\ r(s_5, \rightarrow) &= 100 \end{aligned} \tag{9}$$

Compute the value function $V^\pi(s)$ of the finite horizon problem for the equiprobable policy $\pi(s_i, \rightarrow) = \pi(s_i, \leftarrow) = 0.5$ for $i \in \{2,3,4,5\}$ using two iterations of policy evaluation. Assume a discount factor of $\gamma = 0.8$ and an initial value function of $V(s_i) = 0$. Calculate with integer precision, that means ignore all digits behind the decimal point, for example 78.4 is rounded to 78.
For the same environment, compute the optimal value function $V^*(s)$ and policy $\pi^*(s)$ using three complete cycles of policy iteration. You can use either synchronous or asynchronous updates to calculate $V^\pi(s)$ and $V^*(s)$.
in general

$$V(s_i) = \sum_a \pi(s_i, a)(r(s_i, a) + \gamma V(\delta(s_i, a))) \tag{10}$$

where in our case $\pi(s_i, a) = 0.5$ the reward r and transition function δ are given above. Notice that s_1 and s_6 are terminal states, so there state value remains zero.
policy evaluation
equiprobable policy first iteration:
$V(s_2) = 0.5(50 + 0.8 * 0 - 10 + 0.8 * 0) = 20$
$V(s_3) = 0.5(-10 + 0.8 * 0 - 10 + 0.8 * 0) = -10$
$V(s_4) = 0.5(-10 + 0.8 * 0 - 10 + 0.8 * 0) = -10$
$V(s_5) = 0.5(-10 + 0.8 * 0 + 100 + 0.8 * 0) = 45$

equiprobable policy second iteration (synchronous update):
$V(s_2) = 0.5(50 + 0.8 * 0 - 10 + 0.8 * (-10)) = 16$
$V(s_3) = 0.5(-10 + 0.8 * 20 - 10 + 0.8 * (-10)) = -6$
$V(s_4) = 0.5(-10 + 0.8 * (-10) - 10 + 0.8 * 45) = 4$
$V(s_5) = 0.5(-10 + 0.8 * (-10) + 100 + 0.8 * 0) = 41$

policy iteration
Now $\pi(s, a)$ is a policy that always picks the best action namely the one

that greedily maximizes $r(s,a)+\gamma V(\delta(s,a))$. As in the first scenario s_1 and s_6 are terminal states, so there state value remains zero.
In the first iteration one goes left in state s_2, right in state s_5 and picks an arbitrary action s_3 and s_4. The general update rule becomes

$$V(s_i) = \max_a(r(s_i,a) + \gamma V(\delta(s_i,a))) \tag{11}$$

first iteration:
$V(s_2) = 50 + 0.8 * 0 = 50$ (left)
$V(s_3) = -10 + 0.8 * 0 = -10$ (left or right)
$V(s_4) = -10 + 0.8 * 0 = -10$ (left or right)
$V(s_5) = 100 + 0.8 * 0 = 100$ (right)

second iteration:
$V(s_2) = 50 + 0.8 * 0 = 50$ (left)
$V(s_3) = -10 + 0.8 * 50 = 30$ (left)
$V(s_4) = -10 + 0.8 * 100 = 70$ (right)
$V(s_5) = 100 + 0.8 * 0 = 100$ (right)

third iteration:
$V(s_2) = 50 + 0.8 * 0 = 50$ (left)
$V(s_3) = -10 + 0.8 * 70 = 46$ (right is better than previous action left)
$V(s_4) = -10 + 0.8 * 100 = 70$ (right)
$V(s_5) = 100 + 0.8 * 0 = 100$ (right)

This is also the final value function. The optimal policy is therefore to go left in state s_2 and to go right in all the other states s_3, s_4, s_5.

Question 9 (2p)
Explain the principles and objectives of Boosting? For which learning problems and learners is Boosting likely to improve the performance of the learner.

The idea of boosting is to repeatedly invoke a weak learning algorithm on different distributions of the training data. The goal is to obtain a committee of classifiers that has a smaller error than each individual classifier alone. The overall classification is obtained from the aggregated votes of the individual classifiers. In each iteration of the basic learner, the error on the current distribution of training instances is observed, and the weight of

correctly classified instances is reduced, whereas the weight of incorrectly classified instances is increased. Boosting thereby is able to generate a more complex decision surface than the basic learning alone. Boosting is likely to improve the performance, when the hypothesis space of the basic learner does not match the target concept and when the learning algorithm is unstable, that means the hypothesis is sensitive with respect to the distribution of training instances.

Question 10 (3p)
What is meant by the term inductive bias of a learner? What are the inductive biases of the Find-S algorithm, candidate elimination algorithm and decision tree learning? What is the difference between *restriction bias* and *preference bias*? Give an example of each.

The inductive bias of a learner is the minimal set of assumptions such that for any target concept and corresponding training data the correct classification can be deductively proven. In other words (bias + training data + instance) entail that the classification of the learner for this new instance. The inductive bias of Find-S is that the most specific hypothesis (all instances are negative unless the dataset entails the opposite) consistent with the training data holds and that the target concept is in the hypothesis space, candidate elimination assumes that the target concept is in the hypothesis space and decision tree learning prefers trees of smaller size with high information gain attributes at the top nodes. Restriction bias means that the hypothesis space only contains a subset of all possible hypothesis, so —H— is smaller than —C—. A learner with a preference bias searches a complete hypothesis albeit in an order determined by its preference for some hypothesis over others. Find-S has a restriction bias, decision tree learners have a preference bias.

Question 11 (3p)
Usually feed-forward neural networks (multi-layer perceptrons) are trained with the backpropagation algorithm. How could one train the same type of neural network with a genetic algorithm. In particular describe, how the genotype, phenotype and fitness function would look like.

Instead of using gradient descent in the backpropagation algorithm the

genetic algorithm does optimizes the weight vector with respect to the quadratic error. The fitness of an individual (weight vector) would be the inverse of the sum of squared error between network output and target value.

$$f(w) = \frac{1}{E} = \frac{1}{sum_i(\hat{f}(x_i) - f(x_i))^2} \tag{12}$$

The genotype would encode the synaptic weight vector as a binary string, using Gray code or normal binary coding, each weight for example would be represented by N-bits, the entire string has a length of #weights * N. The phenotype is the entire multi-layer perceptron described by the neurons, weights and activation functions. The optimization is similar to the batch gradient descent and not suitable for stochastic gradient descent or online learning. In addition, the GA could encode general parameters of the network architecture, such as the number of hidden neurons and the presence or absence of synaptic connections. In that case the genotype would be of variable length. One problem in this context is the so-called competing conventions problem, which occurs due to the fact that two networks with swapped hidden neurons and weight vectors, represent the same network function, although their genotypes are different.

Question 12 (3p)
Consider the difficulties that a machine learning algorithm faces in learning to play poker successfully. Discuss the notion of temporal and structural credit assignment in the context of this game. Motivate which learning algorithm you would apply and describe how you would apply it to learn playing poker. What would be the goal that the learner tries to achieve? Which information would be used for learning? How would you train the learner?

The first problem is that of temporal credit assignment, namely that the outcome of the game depends on a sequence of actions like bidding, exchanging cards, taken during one round of play. The second problem is that the game is stochastic by nature, due to the dealing of cards and the uncertainty about the other players hands and bidding behavior, for example bluffing. As a result, the same set of actions might result in a completely different outcome. Another problem is that poker is a game of incomplete information, as the learner does not know the hand of the other players. Another question is from which experience one tries to learn, one could imitate

an expert human poker player (supervised learning), which would require a large database of poker games. One could learn by letting the learner play against itself, but that might result in inferior performance when the learner plays against other strategies. One could train the learner by letting it play against human players or a standard poker program. Another problem is the need for a strategy that is somewhat randomizes, as with a fixed policy the opponent might be able to predict my own behavior. For example, if the learner always bids the maximum bid when it has a full-house, it would reveal too much information to its opponent. The learner could also try to learn a model of the opponents strategy, for example classifying an opponent as a conservative player or a risky player that often bluffs. This information could then be used to optimize its own policy. The most suitable learning algorithm would be reinforcement learning. The current game state would be described by the hand of cards one has, the current bids of the learner and other players, the number of new cards dealt to other players. The possible actions and decisions of the reinforcement learner are pass, bid, raise bid, get new cards (how many and which cards to drop), hold. The reward would be the money won or lost at the end of each game. The goal is to maximize the amount of money won by the program in the long run. It is easiest to regard each round of play separately, however in practice subsequent rounds of play are not totally independent as they influence the behavior of my opponents, for example if the opponent detects that the learner always bluffs, he will not pass on a high bid of the learner. It is also a good idea to separate the decision on bidding, raising and passing from the decision on which cards to drop. The later part could be learned separately, or simply computed directly based on the probabilities of achieving certain combinations of cards. What is still needed is a utility for a particular hand, for example a full-house is worth 50 points, a pair is worth 10 points. This utility function could be learned by means of reinforcement learning. The decision on which cards to drop is then taken according to the maximum expected utility. The reinforcement learning part is then only concerned with learning the optimal policy for the bidding process and uses the fixed policy for exchanging cards.

Question 13 (3p)
What is the *version space*? How does the candidate elimination algorithm operate and how does it represent the version space? Under which condition, is the candidate elimination algorithm able to safely classify an unseen instance as positive or negative? Under which condition, is the candidate

elimination algorithm able to safely classify **all** instances as either positive or negative?

The version space is the subset of hypothesis in the hypothesis space that are consistent with the training data, in other words for every hypothesis $h\ in VS$ and each instance $x \in D\ h(x) = c(x)$. The candidate elimination algorithm works sequentially, for each new instance it removes those hypothesis from the version space that are inconsistent with the training example. Instead maintaining a complete list of all hypotheses in the VS it describes the VS by its general and specific boundary. It then updates this boundary, for example for a positive instance it might generalize the specific boundary and might remove an hypothesis from the general boundary that is inconsistent with the training example. Assuming the target concept is in the hypothesis space Candidate elimination can safely classify an unseen instance if all hypotheses in the VS agree on the classification, it is actually sufficient to check that all hypotheses in the specific and general boundary agree on this instance. This condition is fulfilled for all positive instances inside the S-boundary and all negative instance outside the G-boundary. Obviously, the candidate elimination algorithm can safely classify all instances if the VS has been reduced to a single hypothesis, which then must coincide with the target concept.

Question 14 (3p)
Describe the temporal difference (TD) learning algorithm (Q-learning). In particular, how does the agent represent its knowledge about the world? Which information does the agent use to update its current beliefs/model? In what way is temporal difference learning different from Monte-Carlo methods? How can one unite Monte-Carlo methods and temporal difference learning?

The goal of temporal difference learning is to learn an optimal policy that maximizes the expected, accumulated, future discounted rewards. In order to find such a policy, the agent learns the action value function $Q(s, a)$, namely the expected value of the accumulated rewards, when applying action a in state s. The agent obtains no direct training examples of $Q(s, a)$, instead it forms an estimate of the action value by comparing its current estimate with the sum of immediate reward and the discounted action value of the next state action pair (s',a'), namely the temporal difference of the action

value function between two successive state action pairs.

$$Q(s,a) = Q(s,a) + \alpha(r(s,a) + \gamma Q(s',a') - Q(s,a)) \tag{13}$$

The agent uses the observed sequence of state, action, reward, state action (SARSA) for updating its current estimate of $Q(s,a)$. Given the action value function the optimal policy $\pi(s)$ follows from taking the action that maximizes $\pi(s) = \text{argmax}_\text{a}\text{Q(s, a)}$. In order to explore the state space the agent has to take non-optimal actions every now and then, for example using an ϵ-greedy policy. in Monte-Carlo methods the agent updates its state value function only at the end of an episode, but then updates the entire history of previously visited states. No learning takes place until the end of an episode. Temporal difference learning uses a one step look-ahead and immediately updates the action value function based on its own current estimate of $Q(s,a)$. Monte Carlo method use an N-step look-ahead for updating $Q(s,a)$. Both methods can be combined with $TD(\lambda)$ which updates its action value function based on a mixture of all k-step look ahead $k \in 1, \ldots, N$.