Module – 3 Artificial Neural Networks and Bayes Theorem

1.(a)Explain appropriate problems for Neural Network Learning with its characteristics.

1.*Instances are represented by many attribute-value pairs.*

The target function to be learned is defined over instances that can be described by a vector of predefined features. These input attributes may be highly correlated or independent of one another. Input values can be any real values.

2.The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.

3.The training examples may contain errors.

ANN learning methods are quite robust to noise in the training data.

4.Long training times are acceptable.

Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

5.Fast evaluation of the learned target function may be required.

Though ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast.

6. ***The ability of humans to understand the learned target function is not important.***

The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are less easily communicated to humans than learned rules.

Examples:

- Speech recognition
- Image Classification
- Financial Predictions

b). Consider two perceptrons defined by the threshold expression $w_0+w_1x_1+w_2x_2>0$.

Perceptron A has weight values $w_0=1$, $w_1=2$, $w_2=1$, and

Perceptron B has weight values $w_0=0$, $w_1=2$, $w_2=1$

True or False? Perceptron A is more general than Perceptron B.

Solution:

We will say that h_j is (strictly) more-general than h_k (written $h_j >_g h_k$) if and only if $(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$. Finally, we will sometimes find the inverse useful and will say that ***hj*** is ***more specific than h_k*** when h_k is ***more_general-than h_j.***

X_1	X_2	$w_0+w_1x_1+w_2x_2$ **Perceptron A**	$w_0+w_1x_1+w_2x_2$ **Perceptron B**	**A more general than B ($A \geq_g B$)**
0	0	1+2*0+1*0=1	0+2*0+1*0=0	1
0	1	1+2*0+1*1=2	0+2*0+1*1=1	1
1	0	1+2*1+1*0=3	0+2*1+1*0=2	1
1	1	1+2*1+1*1=4	0+2*1+1*1=3	1

B(<x1,x2>) = 1→ 2x1+x2 > 0 → 1+2x1+x2 > 0 → A(<x1,x2>)) = 1

True.

2. a). Explain Gradient Descent algorithm along with derivation

If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept. The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples. This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm.

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

The delta training rule is best understood by considering the task of training an ***unthresholded*** perceptron; that is, a ***linear unit*** for which the output $\boldsymbol{o}$ is given by

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.

In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples. Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where D is the set of training examples, t_d is the target output for training example *d,* and o_d is the output of the linear unit for training example d.

By this definition, $E(\vec{w})$ is simply half the squared difference between the target output t_d and the hear unit output o_d, summed over all training examples. Here we characterize ***E*** as a function of $\vec{w}$, because the linear unit output ***o*** depends on this weight vector. Of course ***E*** also depends on the particular set of training examples, but we assume these are fixed during training, so we do not bother to write E as an explicit function of these . In particular, there we show that under certain conditions the hypothesis that minimizes E is also the most probable hypothesis in H given the training data.

DERIVATION OF THE GRADIENT DESCENT RULE

We can calculate the direction of steepest descent along the error surface.This direction can be found by computing the derivative of ***E*** with respect to each component of the vector $\vec{w}$. This vector derivative is called the ***gradient*** of ***E*** with respect to $\vec{w}$ written $\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

Notice $\nabla E(\vec{w})$ is itself a vector, whose components are the partial derivatives of ***E*** with respect to each of the w_i. ***When interpreted as a vector in weight space, the gradient specijies the direction that produces the steepest increase in E.*** The negative of this vector therefore gives the direction of steepest decrease.

Since the gradient specifies the direction of steepest increase of ***E,*** the training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

Where

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

Here η is a positive constant called the learning rate, which determines the step size in the gradient descent search. The negative sign is present because we want to move the weight vector in the direction that ***decreases*** E. This training rule can also be written in its component form

$$w_i \leftarrow w_i + \Delta w_i$$

Where

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

----→ (4.5)

differentiating Error function E from

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

---→ (4.6)

where x_{id} denotes the single input component x_i for training example d. We now have an equation that gives in terms of the linear unit inputs x_{id}, outputs O_d, and target values td associated with the training examples. Substituting Equation (4.6) into Equation (4.5) yields the weight update rule for gradient descent

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

2. b) Write the differences between standard and stochastic gradient descent approach.

The key differences between standard gradient descent and stochastic gradient descent are:
1. In standard gradient descent, the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example.
2.Summing over multiple examples in standard gradient descent requires more computation per weight update step. On the other hand, because it uses the true gradient, standard gradient descent is often used with a larger step size per weight update than stochastic gradient descent.
3. In cases where there are multiple local minima with respect to $E(\vec{w})$, stochastic gradient descent can sometimes avoid falling into these local minima because it uses the various $\nabla E_d(\vec{w})$ rather than $\nabla E(\vec{w})$ to guide its search.

3.Explain the following components of artificial neural networks

i) Perceptrons ii) Representational power of Perceptrons
iii) Perceptron training rule

i) Perceptron :

The *perceptron* is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons. Associated with each input, *xj* connection weight $x_j \in \Re$ $j = 1, \ldots, d$, is a *connection weight*,or *synaptic weight wj* synaptic weight $w_j \in \Re$, and the output, *y*, in the simplest case is a weighted sum of the inputs

$$y = \sum_{j=1}^{d} w_j x_j + w_0$$

w_0 is the intercept value to make the model more general; it is generally modeled as the weight coming from an extra *bias unit.*

A perceptron takes a vector of real-valued inputs, calculates a linearcombination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. More precisely, given inputs ***xl*** through ***x,,*** the output ***o(x1, . . .*** , ***x,)*** computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

where each ***wi*** is a real-valued constant, or weight, that determines the contribution of input ***xi*** to the perceptron output. Notice the quantity ***(-wO)*** is a threshold that the weighted combination of inputs $w_1 x_1 + \ldots + w_n x_n$ must surpass in order for the perceptron to output a 1.To simplify notation, we imagine an additional constant input ***xo*** = 1, allowing us to write the above inequality as $\sum_{i=0}^{n} w_i x_i > 0$, or in vector form as $\vec{w} \cdot \vec{x} > 0.$ We will sometimes write the perceptron function as

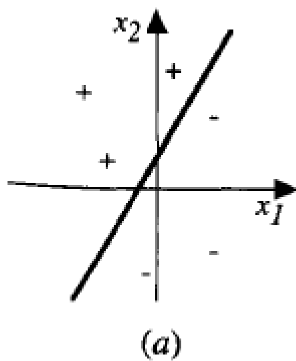
$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

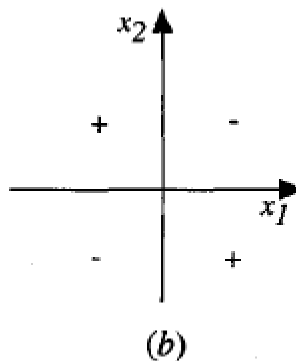
where

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

ii) Representational power of Perceptrons

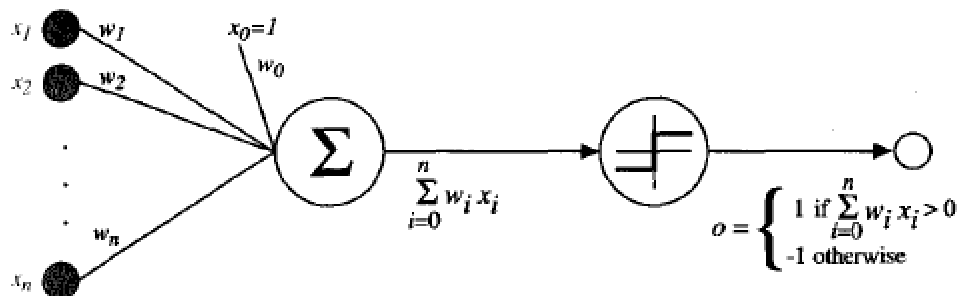
We can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points). The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in Figure. The equation for this decision hyperplane is $\vec{w} \cdot \vec{x} = 0$. Of course, some sets of positive and negative examples cannot be separated by any hyperplane. Those that can be separated are called linearly separable sets of examples.





a)A set of training examples that are linearly separable **b) A set of training examples that are not linearly separable**

A single perceptron can be used to represent many boolean functions. For example, if we assume boolean values of 1 (true) and -1 (false), then one way to use a two-input perceptron to implement the AND function is to set the weights w_o = ***-0.3,*** and $w2 = w_2$ =0 *.5*. This perceptron can be made to represent the OR function instead by altering the threshold to w_o = -0.3. AND and OR can be viewed as special cases of m-of-n functions: that is, functions where at least m of the n inputs to the perceptron must be true. The OR function corresponds to m = 1 and the AND function to m = n. Any m-of-n function is easily represented using a perceptron by setting all input weights to the same value (e.g., 0.5) and then setting the threshold w_o accordingly.
Perceptrons can represent all of the primitive boolean functions AND, OR, NAND (1 AND), and NOR (1 OR). Unfortunately, however, some boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if $\mathbf{x_1 \neq x_2}$. Note the set of linearly nonseparable training examples corresponds to this XOR function.



A Perceptron

iii) Perceptron training rule

Neural networks has many interconnected perceptron units. We need to understand the weights for a single perceptron.

The precise learning problem is to determine a weight vector that causes the perceptron to produce the correct ± 1 output for each of the given training examples. There are two algorithms : the perceptron rule and the delta rule (These two algorithms are guaranteed to converge to somewhat different acceptable hypotheses, under somewhat different conditions. They are important to ANNs because they provide the basis for learning networks of many units.

One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the ***perceptron training rule,*** which revises the weight w_i associated with input x_i according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Here ***t*** is the target output for the current training example, ***o*** is the output generated by the perceptron, and *q* is a positive constant called the ***learning rate.***

The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

4.When will you go for multi layer neural networks? Give the derivation of the back propagation rule.

Single perceptrons can only express linear decision surfaces. When we want to learning non linear training examples we need to go for using multilayer neural networks.

BACKPROPAGATION Weight - Tuning rule:

Back propagation algorithm uses stochastic gradient descent rule for iterating through the training examples one at a time, for each training example d descending the gradient of the error E_d with respect to this single example. For each training example d every weight ***wji*** is updated by adding to it Δw_{ji} .

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

where E_d is the error on training example d, summed over all output units in the network

Here ***outputs*** is the set of output units in the network, t_k is the target value of unit k for training example d, and o_k is the output of unit k given training example d.

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

By using chain rule,

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$
$$= \frac{\partial E_d}{\partial net_j} x_{ji}$$

Case 1: Training Rule for Output Unit Weights. Just as w_{ji} can influence the rest of the network only through net_j, net_j can influence the network only through o_j. Therefore, we can invoke the chain rule again to write

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \tag{4.23}$$

To begin, consider just the first term in Equation (4.23)

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

The derivatives $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ will be zero for all output units k except when $k = j$. We therefore drop the summation over output units and simply set $k = j$.

$$\begin{aligned} \frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j} \frac{1}{2}(t_j - o_j)^2 \\ &= \frac{1}{2} 2(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j) \qquad (4.24) \end{aligned}$$

Next consider the second term in Equation (4.23). Since $o_j = \sigma(net_j)$, the derivative $\frac{\partial o_j}{\partial net_j}$ is just the derivative of the sigmoid function, which we have already noted is equal to $\sigma(net_j)(1 - \sigma(net_j))$. Therefore,

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$
$$= o_j(1 - o_j) \tag{4.25}$$

Substituting expressions (4.24) and (4.25) into (4.23), we obtain

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\ o_j(1 - o_j) \tag{4.26}$$

and combining this with Equations (4.21) and (4.22), we have the stochastic gradient descent rule for output units

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\ (t_j - o_j)\ o_j(1 - o_j) x_{ji} \tag{4.27}$$

Case 2: Training Rule for Hidden Unit Weights. In the case where j is an internal, or hidden unit in the network, the derivation of the training rule for w_{ji} must take into account the indirect ways in which w_{ji} can influence the network outputs and hence E_d. For this reason, we will find it useful to refer to the set of all units immediately downstream of unit j in the network (i.e., all units whose direct inputs include the output of unit j). We denote this set of units by $Downstream(j)$. Notice that net_j can influence the network outputs (and therefore E_d) only through the units in $Downstream(j)$. Therefore, we can write

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k\ w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k\ w_{kj}\ o_j(1 - o_j)
\end{aligned} \tag{4.28}$$

Rearranging terms and using δ_j to denote $-\frac{\partial E_d}{\partial net_j}$, we have

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj}$$

and

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

which is precisely the general rule from Equation (4.20) for updating internal unit weights in arbitrary acyclic directed graphs. Notice Equation (T4.4) from Table 4.2 is just a special case of this rule, in which $Downstream(j) = outputs$.

5.Explain the back propagation algorithm. Mention its limitations. Why is it not likely to be trapped in local minima?

The BACKPROPAGATIlON algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these

outputs. Because we are considering networks with multiple output units rather than single units.

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Limitations :

The error surface of multi layer network can have multiple local minima, in contrast to the single-minimum parabolic error surface. This means that gradient descent is guaranteed only to converge toward some local minimum, and not necessarily the global minimum error.
Why is it not likely to be trapped in local minima?

The BACKPROPAGATION algorithm implements a gradient descent search through the space of possible network weights, iteratively reducing the error E between the training example target values and the network outputs. Because the error surface for multilayer networks may contain many different local minima, gradient descent can become trapped in any of these. As a result, BACKPROPAGATION over multilayer networks is only guaranteed to converge toward some local minimum in E and not necessarily to the global minimum error.

6.Prove that the posterior probability of hypothesis H(H is consistent with D)is inversely proportionate to version space of H with respect to D by using bayes theorem.

assumptions

1. training data D is noise free (i.e., $d_i = c(x_i)$)
2. target concept c is contained in H (i.e. $(\exists h \in H)[(\forall x \in X)[h(x) = c(x)]]$)
3. no reason to believe that any hypothesis is more probable than any other

$$\Rightarrow P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

$$\Rightarrow P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

Bayes theorem:

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

- The problem for the learning algorithms is fully-defined
- in a first step, we have to determine the probabilities for *P(h|D)*

h is **inconsistent** with training data D

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

h is **consistent** with training data D

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

- this analysis implies that, under these assumptions, each consistent hypothesis is a MAP hypothesis, because for each consistent hypothesis

$$P(h|D) = \frac{1}{|VS_{H,D}|}$$

Hence , the posterior probability of hypothesis H(H is consistent with D)is inversely proportionate to version space of H with respect to D.

7.a)What is Bayes' Theorem? How is it useful in a machine learning context?

Bayes' theorem incorporates prior knowledge while calculating the probability of occurrence of the same in future. So, the bayesian probability of some event B occurring provided the prior knowledge of another event(s) A, given that B is dependent on event A (even partially).

- In machine learning we are interested in determining the best hypothesis from some space H, given the observed training data D.
- One way to specify what we mean by the ***best*** hypothesis is to say that we demand the ***most probable*** hypothesis, given the data ***D*** plus any initial knowledge about the prior probabilities of the various hypotheses in H.
- Bayes theorem provides a direct method for calculating such probabilities. More precisely, Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Bayes theorem:

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

P(h) - ***priorprobability*** of ***h***
P(D)- prior probability that training data D
P(D/ h) - the probability of observing data ***D*** given some world in which hypothesis ***h*** holds
P (h/ D) - ***posterior probability*** of ***h***

The approach allows for learning from experience, and it combines the best of classical AI and neural nets.

7.b). Write the features of Bayesian Learning methods.

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct.
- This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example.
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- standard of optimal decision making

8. A patient takes a lab test and the result comes back positive. It is known that the test returns a correct positive result in only 98% of the cases and a correct negative result in only 97% of the cases. Furthermore, only 0.008 of the entire population has this disease.

i). What is the probability that this patient has cancer?

ii). What is the probability that the patient does not have cancer?

iii) What is the diagnosis?

Solution:

$$P(cancer) = .008, \qquad P(\neg cancer) = .992$$
$$P(\oplus|cancer) = .98, \qquad P(\ominus|cancer) = .02$$
$$P(\oplus|\neg cancer) = .03, \qquad P(\ominus|\neg cancer) = .97$$

The maximum posteriori hypothesis can be

$$P(\oplus|cancer)P(cancer) = (.98).008 = .0078$$
$$P(\oplus|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$

Notice that while the posterior probability of cancer is significantly higher than its prior probability, the most probable hypothesis is still that **the patient does not have cancer.**