6.891 Machine learning and neural networks

Mid-term exam: SOLUTIONS

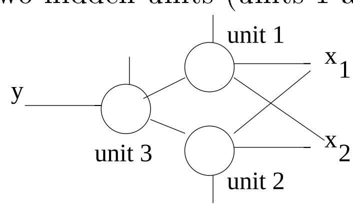
October 31, 2000

(2 points) Your name and MIT ID:

No Body, MIT ID # 0000 0000

Problem 1

1. **(6 points)** Consider a two-layer neural network with two inputs x_1 and x_2, one output unit (unit 3) and two hidden units (units 1 and 2).



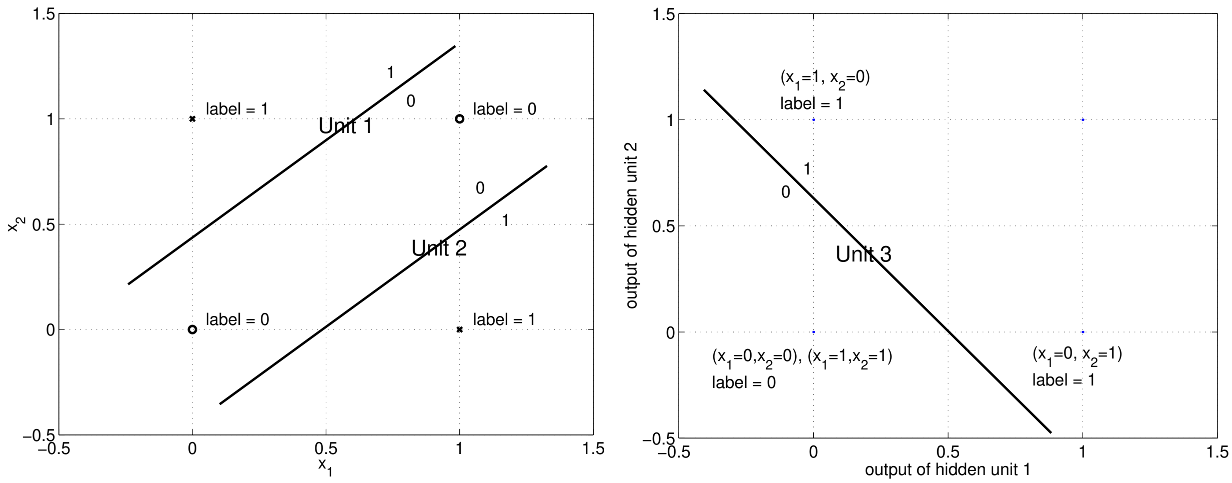
We assume that the transfer functions in the network are threshold functions. For example, unit 1 with two inputs x_1 and x_2 produces an output

$$y = \text{step}(\, w_{10} + w_{11}x_1 + w_{12}x_2 \,) \tag{1}$$

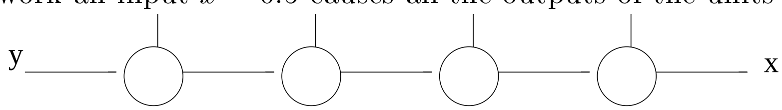
where $\text{step}(z) = 1$ if $z \geq 0$ and zero otherwise.

Show that such a network can solve the XOR problem in the left figure below.

In the left figure, mark the decision boundaries for the two hidden units corresponding to the solution. Also mark on each side of the boundary what the resulting output is for the corresponding unit. Map the outputs to the figure on the right and indicate which of the four input examples the outputs correspond to. Finally, draw the decision boundary corresponding to the output unit 3.



2. **(6 points)** Suppose we are trying to train the following chain like neural network with back-propagation. Assume that the transfer functions are logistic functions and that all the weights are initially set to 1 and all the biases are set to -0.5. Giving such a network an input $x = 0.5$ causes all the outputs of the units to become 0.5.



Now, given a single input $x = 0.5$ and corresponding target output $y = 1$, what can you say about the order of magnitude of the gradient updates for weights in this network? We are looking for a qualitative (not numerical) answer comparing the magnitude of the updates across the different units.

Back-propagation relies on the δ's. They are propagated in the above network as

$$\delta_i = g'(z_i) w_i \delta_{i+1}$$

where larger index indicates a unit closer to the output. In the given setting, $z_i = 0$ *for all* i *and weights are equal to one. Thus* $\delta_i = 1/4\, \delta_{i+1}$ *and the magnitude of the weight updates decreases exponentially as we move away from the output unit.*

3. **(T/F – 2 points)** Applying back-propagation to train a neural network is guaranteed to find the globally optimal solution

F

Back-propagation is a gradient ascent method and thus is susceptible to getting stuck in a locally optimal solution.

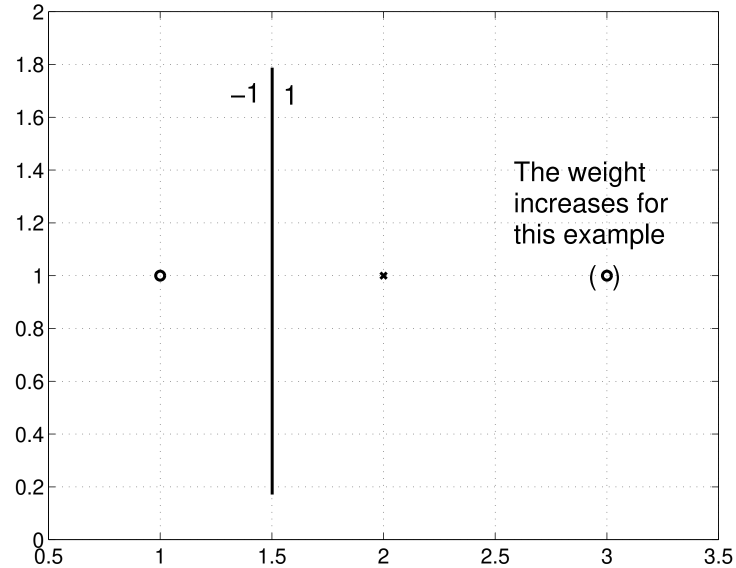
4. **(T/F – 2 points)** Regardless of the choice of the transfer function, setting all the weights close to zero in a neural network makes the network function like a linear mapping from inputs to outputs

   F

   This is true only for smooth transfer functions. For example, it does not hold for the threshold function used in the first problem.

Problem 2

1. **(4 points)** We want to solve the following classification problem with boosting where the component classifiers are decision stumps. In the figure below, a) mark the decision boundary for the first decision stump and b) circle the points whose weight will *increase* as a result. Indicate the positive/negative side of the decision boundary. 'o's in the figure correspond to -1 labels and 'x's denote $+1$ labels.



2. **(4 points)** Give us the new weights on the three training examples after the first boosting iteration (you should not need a calculator for this)

Initially the weights are uniform, i.e., $p_1(i) = 1/3$ *for all* i*. The weights are updated according to*

$$p_2(i) \leftarrow c\, p_1(i)\ \exp(-\alpha_1 y_i h(\mathbf{x}; \mathbf{w}_i))$$

where c *is the normalization constant and* $\alpha_1 = 1/2 \log(\epsilon_1/(1-\epsilon_1))$*. The weighted training error* ϵ_1 *in this case is* $0 + 0 + 1/3 = 1/3$*. Thus*

$$\alpha_1 = 1/2\ \log((1/3)/(2/3)) = 1/2 \log(2)$$

The new weights are (enumerated from left to right in the figure)

$$c\,[\frac{1}{3}\exp(-\alpha_1), \frac{1}{3}\exp(-\alpha_1), \frac{1}{3}\exp(+\alpha_1)] = c\,[\frac{1}{3\sqrt{2}}, \frac{1}{3\sqrt{2}}, \frac{\sqrt{2}}{3}]$$

(naturally you won't lose any points if you used the incorrect expression for α *that was in the lecture notes)*

3. **(6 points)** How many boosting iterations would we need in the previous example so that the combined classifier separates the examples perfectly? Please answer either a) 1 iteration b) 2 iterations, c) at least 3 iterations or d) boosting cannot separate the examples. Briefly explain why.

You need at least three iterations. The component classifiers output binary ± 1 *values and have weights* α_1 α_2 *and* α_3*. If the second hypothesis is shifted one example to the right with an inverted sign and the last one is shifted all the way to the left with a negative sign towards the examples, then the combined classifier outputs the signs of*

$$[-\alpha_1 + \alpha_2 - \alpha_3, \alpha_1 + \alpha_2 - \alpha_3, \alpha_1 - \alpha_2 - \alpha_3]$$

The signs of these terms can be set to $[-1, +1, -1]$ *as desired. So boosting has a chance to separate the examples after 3 iterations.*

4. **(4 points)** Explain briefly why we might ever want to use boosting as opposed to some other way of combining simple classifiers into strong classifiers (such as adapting the forward-fitting algorithm to classification problems)

We know forward-fitting generates a powerful combination in terms of separating the training examples. However, there are no guarantees that the solution will generalize well. Boosting on the other hand is geared towards generalization and tries to maximize the classfication margin on the training examples.

5. **(T/F – 2 points)** The weighted error of each of the component classifiers (error relative to the current weights on the training examples) always goes down as a function of the number of boosting iterations

   F

   In fact the opposite is often true. Since we concentrate the weights on the "hard" examples, the error rate of the component classifiers on the weighted examples typically gets worse.

6. **(T/F – 2 points)** The training error of the combined classifier decreases monotonically as a function of boosting iterations

   F

   There are no guarantees that this will happen. Boosting is not directly minimizing the number of missclassified training examples. Similarly, a gradient ascent training of logistic regression models wouldn't be guaranteed to monotonically decrease the number of missclassified training examples but it would increase the likelihood monotonically.

7. **(T/F – 2 points)** After some k boosting iterations, the next component classifier may achieve only weighted training error close to 0.5 (relative to the current weights on the examples) even if the combined classifier perfectly separates the training examples

   T

   We can continue boosting after the combined classifier has a zero training error. The performance of subsequent component classifiers on the weighted training set typically gets worse (closer to 0.5) as we increase boosting iterations. They are after all trying to separate the hardest examples (by themselves).

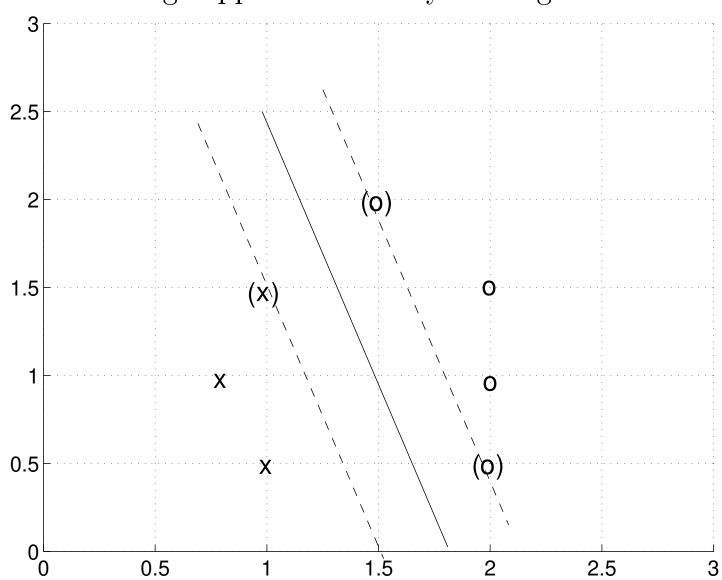
Problem 3

1. **(6 points)** Give us the simplest kernel function that permits support vector machines to represent all the decision boundaries that a mixture of two Gaussian model can

(without any constraints). Also provide a brief justification. We assume that the two components in the mixture model correspond to the binary classification labels.

> *The decision boundary between two Gaussians with arbitrary covariance matrices has a quadratic form. The simplest kernel that is equally powerful is the second degree polynomial kernel, i.e.,*
>
> $$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^2$$
>
> *(if this isn't clear, recall the associated feature vectors).*

2. **(4 points)** In the figure below, draw the maximum margin linear decision boundary and indicate the resulting support vectors by circling them.



3. **(4 points)** Suppose we are given a sequence of training examples $\mathbf{x}_1, \mathbf{x}_2, \ldots$ as well as the corresponding binary ± 1 labels $y_1, y_2, \ldots$. We predict the label for the n^{th} example by training our binary classifier, say $h(\mathbf{x}; \mathbf{w})$, with the first $n-1$ examples and labels and, subsequently, predicting the n^{th} label as the output of $h(\mathbf{x}; \hat{\mathbf{w}}_{n-1})$ ($\hat{\mathbf{w}}_{n-1}$ here denote the parameters that we find on the basis of the first $n-1$ training examples). Such a sitution might arise, for example, in predicting changes in stock prices, where $\mathbf{x}$ captures our knowledge of the current situation.

We wish to use support vector machines for this classification task. Give two reasons for why our ability to predict the n^{th} label might not improve or might even get worse as the number of training examples or n increases (up to say $n = 1000$). Your reasons may pertain to assumptions or properties of support vector machines.

There are at least two possible reasons here. If the sequence of examples and labels are not drawn independently from some fixed underlying distribution then we have little guarantees of generalizing well. The other reason could be "overfitting" in the sense that we might be using too complex family of classifiers whose VC-dimension is close to or greater than 1000. Such a classifier could essentially store the previous training examples without any attempt to model the relation between labels and examples.

4. **(4 points)** Give one reason for and against maximum margin separation of the training examples

For: good generalization. For example, if the test examples are noisy versions of the training examples, then the maximum margin boundary would still assigns them to the appropriate categories with high probability.

Against: any outliers in the training set can radically change the boundary.

5. **(4 points)** Briefly explain how we should set the constant C (level of regularization) in a regularized logistic regression:

$$J(\mathbf{w}) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{C}{2}\|\mathbf{w}\|^2 \tag{2}$$

We should use cross-validation to set the regularization coefficient C. The performance measure for cross-validation should in this case be the log-probability of the held out examples. This is what we seem to be wanting to optimize here. Finding the coefficient C that minimizes the cross-validation "error" involves searching for the minimum value of a function (cross-validation error) that we cannot express in closed form but can nevertheless evaluate for any specific value of C.

6. **(4 points)** Suppose we use the simple mutual information criterion $I(y; x_j)$ to select components x_j of the input vectors $\mathbf{x}$. We assume that we have a large number of training examples and the probabilities in $I(y; x_j)$ can be estimated accurately. Based on the resulting values for $I(y; x_j)$ can we ever definitely include/exclude a feature? Briefly explain why or why not.

We cannot really include or exclude any features. Even if $I(y; x_j) = 1$, i.e., that the feature perfectly predicts the label, we wouldn't necessarily want to include all such features since they may be redundant. Even if $I(y; x_j) = 0$, we cannot exclude the feature since this feature, if combined with another feature, might still perfectly predict the label. For example, let $y = x_1 x_2$ where all the variables are binary ± 1 and x_1 and x_2 are drawn at random. In this case $I(y; x_1) = I(y; x_2) = 0$ but x_1 and x_2 jointly perfectly predict the label.

7. **(T/F – 2 points)** The computational cost of estimating support vector machines increases linearly with the number of training examples — F

Solving the quadratic programming problem requires computational resources somewhere between $O(n^2)$ and $O(n^3)$.

8. **(T/F – 2 points)** A polynomial kernel function of degree one can solve the XOR-problem discussed in problem 1 — F

A linear kernel means a linear separator in the input space. A linear separator cannot solve the XOR problem.

9. **(T/F – 2 points)** As far as generalization error is concerned, the choice of the kernel function matters only in terms of what the resulting VC-dimension is

   F

   VC dimension is a rough combinatorial measure. We can put a lot of prior knowledge into the choice of the kernel function and considerably affect the generalization performance.

Problem 4

1. **(4 points)** Give one reason for why we would want to use the EM- algorithm for training mixture models rather than trying to maximize the log-likelihood of the training data via gradient ascent.

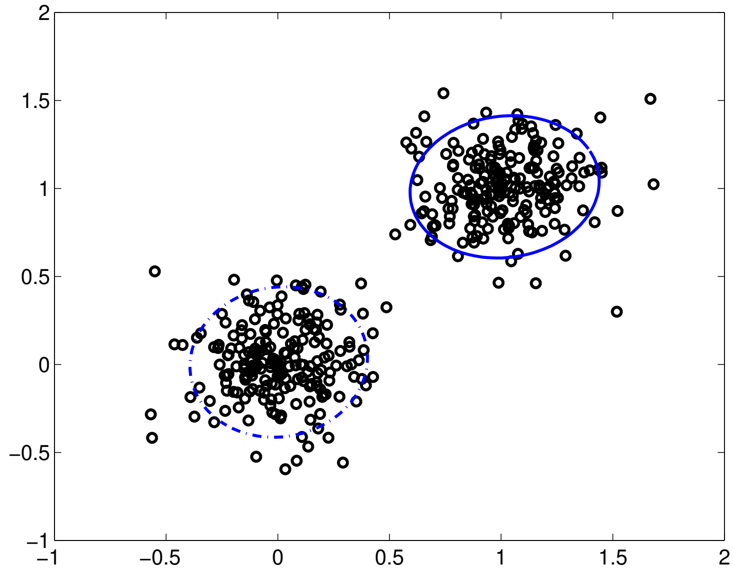
   In gradient ascent we have to be careful about not taking too long steps as the likelihood may decrease as a result. EM takes at times very long steps but it is still guaranteed to monotonically increase the likelihood. So the main reason is speed. The fact that EM is monotonic also serves as a great diagnostic tool. If the training likelihood does not increase monotonically you know there's a bug.

2. **(3 points)** Why doesn't the EM-algorithm converge after one iteration?

EM is inherently an iterative algorithm in the sense that the parameter updates in the M-step depend on our previous or initial setting of the parameters. In the context of mixture of Gaussians models, for example, the means in the M-step are computed as weighted means of the training examples. The weights, however, are the posterior probabilities over the components based on our previous parameter values. Thus the new means after the M-step are still dependent on our initial parameters (which may have been quite bad). This dependence on the initial parameter values decreases as we continue iterating.

3. **(3 points)** In the figure below, what is *not explicit* in the representation of the mixture of two Gaussians? In other words, what cannot I get even approximately from the figure? These are the type of figures that you have seen in lectures in the context of the EM algorithm.

The mixing proportions (prior probabilities of selecting mixture components) are not represented at all in the figure. The mean of the two Gaussians can be found as the center points and the covariance matrices could be in principle reconstructed from the ellipsoids.



4. **(4 points)** Suppose the mixture model given in the previous figure serves as our initial guess for the EM-algorithm. Could the EM- algorithm under any initial choice of the remaining parameters, those that are not specified in the figure, move the "solid" Gaussian over both clusters after one iteration? Briefly explain why or why not.

Yes it could. Examples are assigned to the Gaussians on the basis of the posterior probabilities measuring which Gaussian better explains each point. A part of this posterior probability comes from the prior probability of selecting the Gaussians, i.e., the mixing proportions. If the "solid" Gaussian has a prior probability very close to 1, then the posterior probability that it generated all the examples can be rclose to 1. In other words, the other Gaussian model would have to have a much higher likelihood of generating the examples to overcome a strong prior probability favoring the other Gaussian.

5. **(T/F – 2 points)** When the EM-algorithm converges for a mixture of Gaussians model, the mixing proportions $p_1, \ldots, p_k$ become equal to the posterior component probabilities $P(y = j|\mathbf{x}_i)$ for all training examples

F

The mixing proportions converge to the average of the posterior probabilities. The posterior probabilities vary from one training example to another, however.

6. **(T/F – 2 points)** Since Parzen windows is a non-parametric density estimation method the properties of the kernel bumps we assign over each training example are irrelevant.

   F

   Kernel width, for example, is a major consideration in non-parametric density estimation.