# CSE 569 Homework #5 Solutions

**Question 1 Answer:**

The "argument" presented in the question might be due to the following misunderstanding: if the output does not change much with respect to the input (i.e., *small f'*(net)), the network might not be learning, and thus we would want to update the weights *more*, to make the network learn. But this is a misunderstanding since whether or not the network needs to learn more should be assessed by whether the error/loss $J$ is still large (but not by whether the output is not changing much). If $J$ is still large, that means (on average) the output is still far from the target and needs to be changed more. Using the typical sigmoid $f$ as an example, where the output is far from the target is where $f'$ is large. So intuitively, wherever $f'$(net) is large, we will want to update the weights more. (The precise relation is in the update equations, based on computing the gradients.)

**Question 2 Solution:**

Please refer to Eqns. (26)-(29) in the textbook for more detailed steps of the derivation. You will still need to remember these equalities to understand all steps of Eqn. (26):

$$p(\mathbf{x}, \omega_k) = P(\omega_k|\mathbf{x})p(\mathbf{x})$$

$$P(\omega_k|\mathbf{x}) + P(\omega_{i\neq k}|\mathbf{x}) = 1$$

**Question 4 Solution:**

(a) Given only $p(x)$, without knowing the variances of the components and the mixing parameter, we cannot uniquely determine the parameters. For example, with the given mixture density, if $\sigma_1 = \sigma_2$ (but otherwise arbitrary), $P(\omega_1)$ can take any value in the range [0, 1], and still results in the same mixture density. Thus the density is completely unidentifiable.

(b) When $P(\omega_1)$ is fixed and known, consider the following four cases. (i) If $P(\omega_1) = 0$, the first component cannot be identified (i.e., $\sigma_1$ cannot be recovered); (ii) If $P(\omega_1)=1$, the second component cannot be identified (i.e., $\sigma_2$ cannot be recovered); (iii) If $P(\omega_1)=0.5$, $\sigma_1$ and $\sigma_2$ are interchangeable (i.e., not identifiable); (iv) For other general values of $P(\omega_1)$, the model is identifiable (i.e., by definition, if you are given two different sets of parameters ($\sigma_1$, $\sigma_2$) and ($\sigma_1'$, $\sigma_2'$), you will be able to show at least for some $x$, the resultant $p(x)$ will be different).

(c) Yes, if $\sigma_1 \neq \sigma_2$; No, if $\sigma_1 = \sigma_2$.

**Question 5 Solution:**

(1) Strictly from the pseudocode of the algorithm, which says "classify $n$ samples according to nearest $\boldsymbol{\mu}_i$ ; recompute $\boldsymbol{\mu}_i$", it appears that, if we initialize all $\boldsymbol{\mu}_i$ , $i$=1,…,$C$, to the same initial value, all $\boldsymbol{\mu}_i$ will be of equal distance to any given sample, and thus all $\boldsymbol{\mu}_i$ will be updated (in "recomputing") exactly the same way, and thus the algorithm will not work.

However, in implementation, the samples will be assigned to only a particular $\boldsymbol{\mu}_i$ ("nearest" will be determined by comparing distances using "less or equal to", and thus even if the distances are the same, one will be chosen) and thus after the first iteration, one $\boldsymbol{\mu}_i$ will be pulled apart from other $\boldsymbol{\mu}_i$'s, and in subsequently iterations, we may gradually make the $\boldsymbol{\mu}_i$'s different from each other. And thus the algorithm can still work (but of course it suffers from typical drawbacks of such iterative approaches: a poor initialization may cause slower convergence or convergence to a poorer local minimum).

(Further discussion: the above explanation will not help if all $\boldsymbol{\mu}_i$'s are initialized to be the mean of the data, since they will always remain at that value; but in practice, most good implementations would re-initialize a given cluster center if it does not receive any sample in one iteration. Hence you may still find that, e.g., the MATLAB kmean algorithm will also work even if you set all initial cluster centers to be the mean of the data.)

(2) In terms of $T$, $C$ and $N$, the basic algorithm as given in the slide will have a complexity of $O(TCN)$ since for each iteration we will need to computer the distance of any sample to any centroid. (If we consider the dimensionality $d$ of feature space as well, the complexity would be $O(TCNd)$.)

(3) That is something we would hope for, but unfortunately, as alluded in Part (1) by the underlined sentence, there is no guarantee that the resultant cluster centers of the $k$-Means algorithm will happen to give us the $\boldsymbol{\mu}_i$'s. In general, for a given dataset, there might be many local minima that the basic $k$-Means algorithm may converge to, depending on the initialization.

**Question 6 Solution** (For Part (a) only; Part (b) can be done similarly.)

For Part (a), the question basically asks, based on the $J_{ee}$ criterion, if we put $y_1$ at 0, where should we put $y_2$ and $y_3$ (assuming $y_1<y_2<y_3$)? So we will first find out $J_{ee}$ as a function of $y_2$ and $y_3$.

$$J_{ee} = \frac{\sum_{i<j}(d_{ij} - \delta_{ij})^2}{\sum_{i<j} \delta_{ij}{}^2}$$

Based on what is given (for $\mathbf{x}_i$ and $y_i$), we have $\delta_{12}=1$, $\delta_{13}=\sqrt{2}$, $\delta_{23}=1$, $d_{12}=y_2$, $d_{13}=y_3$, $d_{23}=y_3-y_2$. Plug all these into $J_{ee}$ , we will get

$$J_{ee} = \frac{(y_2 - 1)^2 + (y_3 - \sqrt{2})^2 + (y_3 - y_2 - 1)^2}{1 + 2 + 1}$$

Now, taking the partial derivative with respect to $y_2$ and $y_3$, and setting to zero, we will have two linear equations of $y_2$ and $y_3$. Solve the equations for $y_2$ and $y_3$, we will get

$y_2 = (1+\sqrt{2})/3, \quad y_3 = (2+2\sqrt{2})/3.$

**Question 3 Solution:**
(Partial solution given in the scanned pages.)

(Partial solution only)

The point of doing this exercise is < softmax is widely used;
learn to derive learning rule under different loss functions.

(In research, you may have a network with your own loss function.)

But the full solution is quite lengthy, if I have to write down every step. [And thus you can imagine I will not ask you to do this in the exam.]

So, in the following, I'll walk you through only the first few steps of Part (a) only.

— Some basics first: For a function $z_k = f(net_k) = \frac{e^{net_k}}{\sum_{m=1}^{c} e^{net_m}}$,

if you want to find out $\frac{\partial z_k}{\partial net_k}$, you'll need to use the quotient rule of differentiation

$$\frac{d}{dx}\left(\frac{g(x)}{f(x)}\right) = \frac{g'(x)f(x) - f'(x)g(x)}{(f(x))^2},$$

And you will get $\frac{\partial z_k}{\partial net_k} = \cdots = z_k(1 - z_k)$.

But if we want to find out $\frac{\partial z_s}{\partial net_k}$, note the difference here, $s \neq k$.

you will get $\frac{\partial z_s}{\partial net_k} = -z_s z_k$, $(s \neq k)$

(The difference is important here, since in the network of Fig 6-4, $z_k$ is related to only $net_k$, but in softmax, $z_k$

is made dependent of other $net_j$ for $j \neq K$. Hence when consider $\frac{\partial z_s}{\partial net_K}$, depending on $s=K$ or $s \neq K$, you'll have different results after using the quotient rule of differentiation.)

— Now, we'll follow the notations of Section 6.3 of the textbook for updating $w_{kj}$ (the hidden-to-output weights), we need to compute $\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_K} \frac{\partial net_K}{\partial w_{kj}}$, $\frac{\partial net_K}{\partial w_{kj}} = y_j$, so this term is done.

Let's compute the first term:

$$\frac{\partial J}{\partial net_K} = \frac{\partial}{\partial net_K}\left(\frac{1}{2}\sum_{m=1}^{c}(t_m - z_m)^2\right)$$

$$\vdots$$

$$= \frac{1}{2} \times 2 \sum_{m=1}^{c}(t_m - z_m)(-1)\frac{\partial z_m}{\partial net_K}$$

Note: as said above, this is easy for the network of Fig. 6.4 since only the $k$th term in the sum is related to $net_K$. But now every $m$th term is related to $net_K$, since $z_m = \frac{e^{net_m}}{\sum_{l=1}^{c} e^{net_l}}$ ← Here you'll have $net_K$.

Now, you need to use the results from the previous page on $\frac{\partial z_s}{\partial net_K}$

And after some simplification, we can have

$$\frac{\partial J}{\partial net_K} = \sum_{m \neq K}^{c}(-1)(t_m - z_m)(-z_m z_K) - (t_K - z_K) z_K (1 - z_K)$$

$$\therefore \frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_K} \cdot \frac{\partial net_K}{\partial w_{kj}} = (\downarrow) \cdot y_j.$$

Now, with $\frac{\partial J}{\partial w_{kj}}$ ready, we can update weight $w_{kj}$, according to $\eta \frac{\partial J}{w_{kj}}$, where $\eta$ is the learning rate.

Following the same procedure in the textbook for $\frac{\partial J}{\partial w_{ji}}$, and keeping in mind the caveat we had earlier on $\frac{\partial z_s}{\partial net_k}$, we can get the update rule for $w_{ji}$ too.

---

For part (b), if you follow the same path, you will have the answer. Note that the only key difference is in the step ★ (in the previous page), the first level of differentiation is now different since $J_{ce}$ is a different function now. Otherwise, all the rest derivation is pretty much the same.