CS260, Winter 2017
Problem Set 6: Clustering and Neural Networks
Due 3/10/2017

Ray Zhang

March 15, 2017

1 Q1

(a) Problem 1a

Solution: Solution to 1a

This is our original formula.

$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||_2^2$

The above can also be expressed in terms of:

$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} (x_n - \mu_k)^T (x_n - \mu_k)$

Taking the gradients of μ_k, we can find that it is equal to:

$\nabla_{\mu_k} D = \sum_{n=1}^{N} r_{nk} 2(x_n - \mu_k) = 0$

$\sum_{n:r_{nk}=1}^{N_k} (x_n - \mu_k) = 0$

$N_k \mu_k = \sum_{n:r_{nk}=1}^{N_k} x_n$

$\mu_k = \frac{\sum_{n:r_{nk}=1}^{N_k} x_n}{N_k}$

(b) Problem 1b

Solution: Solution to 1b

This is our original formula.

$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||_1$

The above can also be expressed in terms of:

$D = \sum_{n=1}^{N} \sum_{d=1}^{D} \sum_{k=1}^{K} r_{nk} |x_{nd} - \mu_{kd}|$

where d is the d-th dimension of the vectors.

We can then represent the absolute value with some indicator functions:

$\sum_{n=1}^{N} \sum_{d=1}^{D} \sum_{k=1}^{K} r_{nk}[1_{x_{nd}>\mu_{kd}}(x_{nd} - \mu_{kd}) - 1_{x_{nd}\leq\mu_{kd}}(\mu_{kd} - x_{nd})]$

Taking derivatives with respect to μ_{kd}, and maximizing it, we get:

$\frac{\partial D}{\partial \mu_k d} = \sum_{n=1}^{N} r_{nk}[1_{x_{nd}>\mu_{kd}} - 1_{x_{nd}\leq\mu_{kd}}] = 0$

This minimizer is satisfied when:

$\sum_{n=1}^{N} r_{nk} 1_{x_{nd}>\mu_{kd}} = \sum_{n=1}^{N} r_{nk} 1_{x_{nd}\leq\mu_{kd}}$

Or concisely:

$\sum_{n:r_{nk}=1} 1_{x_{nd}>\mu_{kd}} = \sum_{n:r_{nk}=1} 1_{x_{nd}\leq\mu_{kd}}$

This gives us that μ_{kd} is minimized when the points in the class k has as many points less than μ_{kd} as the number of points greater than μ_{kd}. This is the definition of a empirical median, and gives us $\mu_{kd}^* = M(X_{kd})$ where the function M gives the median of the data in the k-th class, in the d-th dimension.

The entire vector of μ_k, is then expressed as:

$$\begin{pmatrix} M(X_{k1}) \\ M(X_{k2}) \\ ... \\ M(X_{kn}) \end{pmatrix}$$

Which is the median vector.

2 Q2

(a) Problem 2.2

Solution: Gradient calculations

For the full update, we need to treat the 3 layers differently because they are different.

Some facts:

$\nabla_x \sigma(x) = \sigma(x)(1 - \sigma(x))$

$\nabla_x tanh(x) = 1 - tanh^2(x)$

$\nabla_x x^T y = y$

very important: consider the situation AB = C, where A, B, C are all matrices. We can't take the gradient of a function that outputs multiple numbers. Thus, we must assume there comes an incoming gradient of ∇_C in the same shape as C.

For some function $f(C)$, we have that $\nabla_{C_{ij}} = \frac{\partial f}{\partial C_{ij}}$.

For the derivative of some $\frac{\partial f}{\partial A_{ab}}$, we have that $C_{ij} = \sum_k A_{ik} B_{kj}$.

Case 1: $i \neq a$, then we have no terms involving A_{ab}. This is 0.

Case 2: $i = a$, the following decomposition: $\sum_{k \neq b} A_{ik} B_{kj} + A_{ab} B_{bj}$ leads to:

$\frac{\partial C_{ij}}{\partial A_{ab}} = B_{bj}$

By the additive property, we know that:

$\frac{\partial f}{\partial A_{ab}} = \frac{\partial f}{\partial C_{a1}} \frac{\partial C_{a1}}{\partial A_{ab}} + \frac{\partial f}{\partial C_{a2}} \frac{\partial C_{a2}}{\partial A_{ab}} + \cdots \frac{\partial f}{\partial C_{an}} \frac{\partial C_{an}}{\partial A_{ab}}$

Which is also equal to:

$\frac{\partial f}{\partial C_{a1}} B_{b1} + \frac{\partial f}{\partial C_{a2}} B_{b2} + \cdots \frac{\partial f}{\partial C_{an}} B_{bn}$, substituting from the results above.

We can see that this is equal to: $\sum_i \frac{\partial f}{\partial C_{ai}} B_{bi}$

Recall that ∇_C is composed of these partials. After tiling it, we can see that:

$\nabla_A = \nabla_C B^T$

We then have these forward equations. Define X as the design matrix $A^1 \in \Re^{nxd+1}$, $W^1 \in \Re^{d+1xh}$, $W^2 \in \Re^{h+1xc}$, $y \in \Re^{nxc}$.

$Z^2 = A^1 W^1$

$A^2 = tanh(Z^2)$

$Z^3 = A^2 W^2$

$A^3 = \sigma(Z^3)$

$L(A^3, y) = -\sum_{n=1}^{N} \sum_{c=1}^{C} y_{nc} log(A^3_{nc}) + (1 - y_{nc}) log(1 - A^3_{nc})$

The entire forward equation can be expressed as:

$y - \sigma(tanh(A^1W^1)W^2)$

Taking derivatives, we can recover the derivatives for ∇_{W^1} and ∇_{W^2}.

Starting with $L(A^3, y)$, we can get the derivative:

$\nabla_{A^3}L(A^3, y) = y \oslash A^3 - (1-y) \oslash (1-A^3)$ where $\oslash$ is a hadamard divisor.

$\nabla_{Z^3} = \nabla_{A^3} \otimes \sigma(Z^3) \otimes (1 - \sigma(Z^3))$

$\nabla_{A^2}Z^3$ is a little harder. This is a matrix multiplication instead of a hadamard operator. We use the identity $\nabla_A = \nabla_C B^T$ in this case to get that

$\nabla_{A^2} = \nabla_{Z^3}W^{2T}$

And similarly,

$\nabla_{W^2} = A^{2T}\nabla_{Z^3}$ from similar reasoning.

The rest is a bit easier, since it's similar in computation.

$\nabla_{Z^2} = \nabla_{A^2} \otimes (1 - (A^2)^2)$

$\nabla_{W^1} = A^{1T}\nabla_{Z^2}$.

To expand everything, we see that

$\nabla_{W^2} = A^{2T}((y \oslash A^3 - (1-y) \oslash (1-A^3)) \otimes \sigma(Z^3) \otimes (1 - \sigma(Z^3)))$

$\nabla_{W^1} = A^{1T}((y \oslash A^3 - (1-y) \oslash (1-A^3) \otimes \sigma(Z^3) \otimes (1-\sigma(Z^3)))W^{2T} \otimes (1 - (A^2)^2)$

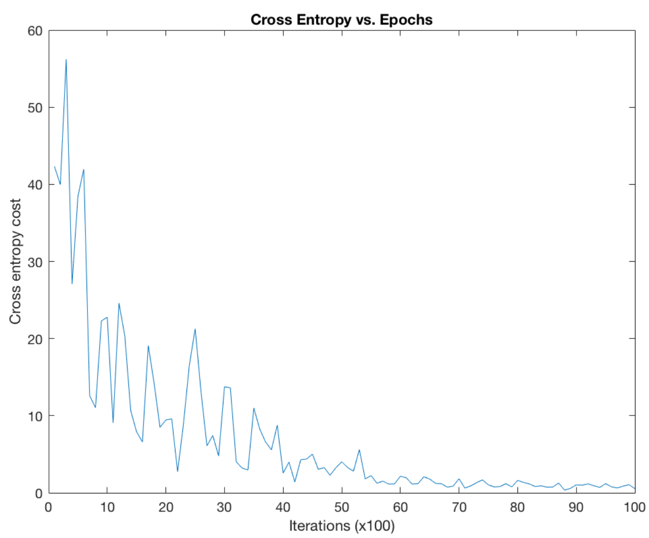
which is our answer.

Solution: Hyperparameter tuning effects

• With regards to learning rate, it appears that the solution converges slower when the learning rate is set to an initial lower value, e.g. 1e-4 instead of 1e-3. The decay function of the learning rate is exponential Le^-kt. The decay rate, when set to a large value, like 1e-3, causes the learning rate to become so low that it does not make any progress. Conversely, too small of a value, like 1e-5, causes the learning rate to not change, and thus effectively not decay. The learning rate and decay rate should thus be in proportion of each other - when the learning rate increases in size, the decay rate should also increase.

• Large initial weights imply that we assume larger patterns about the data. Using randn to generate the weights essentially means that we don't understand the data, which is a contradiction. Naturally, we would not want to set the initial weights to be too large, like randn*1000. The better method is to decrease the magnitude to a degree where floating point issues don't come up and the values are relatively small and dissimilar as to break symmetry. I found that randn*1e-2 was a good initialization for this.

Solution: Training log



Solution: Training Accuracy

The training accuracy was about:

accuracy of training data on final classifier is : 9.996833e-01

Solution: Testing Accuracy

The testing accuracy was about:

accuracy of testing data on final classifier is : 9.716000e-01