HW 6

1 Q)

Now, let $x = \log[r(y)/q(y)]$ Then we have:

$$E[x] = \int \log[r(y)/q(y)] \cdot P(y)dy$$

where $P(y)$ is Probability density fn.

Using jensen's inequality with convex fn.

$$\phi(a) = e^a$$

$$E[x] \leq E[e^x]$$

Substituting $x = \log[r(y)/q(y)]$

$$e^{E[\log(r(y)/q(y)]} \leq E[e^{\log[r(y)/q(y)]}$$

Simplifying right hand side.

$$E[e^{\log[r(y)/q(y)]} = E[r(y)/q(y)] = \int r(y)/q(y) \cdot P(y)dy.$$

Since $\log(x)$ and e^a are inverse functions, we have

$e^{\log(a)} = a$, Substituting this in left hand side & Simplifying

$$r(y)/q(y) \leq E[r(y)/q(y)]$$

Taking natural logarithm of both sides.

$$\log[r(y)/q(y)] \leq \log[E[r(y)/q(y)]$$

∴, we shown that

$$\log[r(y)/q(y)] \leq \log E[\log[r(y)/q(y)]$$

(Q2)

$$\hat{Pr}_{U_k}(u) = \frac{1}{(M-m+1)} \sum_{t=m}^{M} Pr(u | U_l^{(t)}, l \neq k)$$

To justify the estimate given by eq(8.50), first we need to understand the context of problem. We are given a statistical problem where we have training data $U_l^{(t)}$, $l \neq k$ with l representing class labels and t representing data points. We are interested in estimating probability $Pr(u | U_l^{(t)}, l \neq k)$ for a given class u and some data points $U_u^{(t)}$.

We can use relationship provided in eq(1) to compute the probability $Pr(u | U_l^{(t)}, l \neq k)$ by marginalizing over distribution of data points $U_l^{(t)}$

$$Pr(u | U_l^{(t)}, l \neq k) = \int Pr(u | U_l^{(t)}, B, l \neq k) \, d(Pr(B))$$

Now, let's divide range of data points into $M-m+1$ equally spaced intervals and let the midpoints of these intervals be denoted as u_l^m, u_l^{m+1}, ... We can approximate the integral by summing midpoints.

$$Pr(u | U_l^{(t)}, l \neq k) \approx \frac{1}{M-m+1} \sum_{t=m}^{M} Pr(u | U_l^{(t)}, B, l \neq k) \, d(Pr(B))$$

Now as summation does not depend on the measure $d(Pr(B))$, we can remove it from sum:

$$Pr(u | U_l^{(t)}, l \neq k) \approx \frac{1}{M-m+1} \sum_{t=m}^{M} Pr(u | U_l^{(t)}, l \neq k)$$

which justify the estimate given by eq(8.5)

(Q3) To show that bagging estimate $f_{bag}(x)$ converges to the original estimate $f(x)$ as $B \to \infty$, we need to show that the expected value of $f_{bag}(x)$ converges to $f(x)$ as $B \to \infty$.

we can write bagging estimate as:

$$f_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} f^{*b}(x)$$

where $f^{*b}(x)$ is estimate of $f(x)$ from b^{th} bootstrap. Using Parametric bootstrap.

$$y_i^{*b} = \hat{\mu}(x_i) + \hat{\varepsilon}_i^{*b}$$

where $\hat{\mu}(x_i)$ is estimate of $\mu(x)$ from B-spline smoothe

we can then obtain estimate $f^{*b}(x)$ by fitting B-Spline smoothes to bootstrap sample y_i^{*b}.

Now, let's compute expected value $f_{bag}(x)$

$$E[f_{bag}(x)] = E\left[\frac{1}{B} \sum_{b=1}^{B} f^{*b}(x)\right]$$

$$= \frac{1}{B} \sum_{b=1}^{B} E[f^{*b}(x)]$$

we know that B-spline smoothes $\hat{\mu}(x)$ is a consistent estimator of $\mu(x)$, meaning that $\hat{\mu}(x) \to \mu(x)$ as $N \to \infty$ (N is sample size)

$\therefore B \to \infty$, we can assume that $\hat{\mu}(x) \to \mu(x)$

$$\therefore E[f_{bag}(x)] = \frac{1}{B} \sum_{b=1}^{B} E[f^{*b}(x)]$$

$$\to \frac{1}{B} \sum_{b=}^{B} f(x) = f(x) \text{ as } B \to \infty.$$

(4) The purpose of the exercise is to investigate the behaviour of a neural network using sigmoid activation function or more generally, any activation function with property of being nearly around the origin, we will take for granted in this exercise that the neural network uses sigmoid activation fn. with property that its almost linear near origin. let's first recall sigmoid activation fn.

$$\sigma(v) = \frac{1}{1+e^{-v}}$$

now, let's examine the taylor expansion for activation fn. $\sigma(v)$ in the context of neural network

$$\sigma(\alpha_0^{(m)} + \alpha_m^T x) = 1 / 1 + \exp(-\alpha_0^{(m)} - \alpha_m^T x)$$

$$= \left(\frac{\exp(\alpha_0^{(m)})}{1+\exp(\alpha_0^{(m)})}\right)\left(\frac{1}{\exp(-\alpha_m^T x) - 1}{1+\exp(\alpha_0^{(m)})+1}\right)$$

we can analyze behaviour of activation fn. of x approach zero:

$$\exp(-\alpha_m^T x) - 1 \rightarrow -\alpha_m^T x.$$

which leads to:

$$\sigma(\alpha_0^{(m)} + \alpha_m^T x) \rightarrow \left(\frac{\exp(\alpha_0^{(m)})}{1+\exp(\alpha_0^{(m)})}\right)\left(1 + \frac{\alpha_m^T x}{1+\exp(\alpha_0^{(m)})}\right)$$

from this, it becomes clear that fn. $f(x)$ is essentially a linear fn. of x.

now, let's define $u(x) = \frac{1}{2}(1+x)$

$$\lim_{x \to 0} \frac{u(x)}{\sigma(x)} = \lim_{x \to 0} \frac{1}{2}(1+x)(1+e^{-x})$$

$$= \lim_{x \to 0} \frac{1}{2}(1+e^{-x}+x+xe^{-x}) = 1$$

Considering the case when α_m values are close to zero, the values of $\alpha_m^T x$ will also be small. If we incorporate a bias term and expand σ, we obtain $z_m = \sigma(\alpha_m^T \alpha) \approx \frac{1}{2}(1 + \alpha_m^T x)$. As a result, the model exhibits as nearly linear behaviour in x since g_k is an identity fn.

In conclusion, we have demonstrated that when employing a sigmoid activation fn. exhibiting near linear properties close to origin, the neural network model tends to approximate a linear behaviour in x. This observation is particularly relevant when the α_m values are close to zero, further emphasizing the linear characteristics of model in such cases.

(Q5) we will use cross-entropy loss fn. which is defined as

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

let's derive the forward & Backward Propogation equations.

Forward Propogation: This is process of computing the output of neural network given an input x. we calculate output of each layer based on output of Previous layers.

initialize $a^{[0]} = x$. for $l = 1, 2, \ldots L$: a. calculate $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$. b calculate $a^{[l]} = g^{[l]}(z^{[l]})$

The final output of network is $\hat{y} = a^{[L]}$

Backward Propogation: This is Process of computing the gradients of loss fu. with respect to weights and biases of neural network. we use the chain rule to compute these gradients.

calculate error in output layer.

$$d^{[L]} = -\frac{dL}{dz^{[L]}} = \frac{dL}{dz^{[L]}} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dz^{[L]}}$$

$$= (a^{[L]} - y) \odot g^{[L]}(z^{[L]})$$

For $l = L-1, L-2, \ldots 1$: a calculate the error in layer l $d^{[L]} = (w^{[l+1]T} d^{[l+1]}) \odot (z^{[l]})$. for $l = 1, 2, \ldots L$: a Calculate gradients with respect to weights

$$\frac{dL}{dw^{[l]}} = d^{[l]} a^{[l-1]^T}$$. b. Calculate gradients with respect to baises: $\frac{dL}{db^{[l]}} = d^{[l]}$.

Once gradient are computed, we can update the weights and biases using a suitable optimization algorithm, such as gradient descent (or any of its variants.

In other words, the context of Cross-entropy loss is defined as

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log (f_k(x_i))$$

where y_{ik} is true label of the i^{th} data point for class k, $f_k(x_i)$ is the Predicated Probability for class k given input x_i. The corresponding classifier for this objective fu. is given by.

$$G(x) = \arg \max_k f_k(x).$$

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ as defined in eq (11.5) in text. Let $z_i = (z_{1i}, z_{2i} \cdots z_{mi})$

Objective fn: $R(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} (-y_{ik} \log(f_k(x_i))$

$$\frac{\partial R_i}{\partial \beta_{km}} = \frac{-y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i) z_{mi},$$

$$\frac{\partial R_i}{\partial \beta_{km}} = -\sum_{k=1}^{K} \frac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_{0m} + \alpha_m^T x_i) x_{i,l}.$$

Based on these derivates, a gradient descent update can be performed at the $(r+1)^{th}$ iteration as follows:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} = -\gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \alpha_{ml}^{(r+1)}$$

$$= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}.$$

γ_r is learning rate.

By Partial derivatives as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi} \cdot \frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}.$$

From their definitions we have back Propagation eq.

$$s_{mi} = \sigma'(\alpha_{0m} + \alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

To summary, forward Propagation calculates the Predicted Probabilities for each class given an input, while backward Propagation computes the gradient of loss fn. with respect to model Parameters. These gradients are then used to update model Parameters iteratively refining the model's prediction by minimizing cross-entropy loss over training Data