Problem 1.

[2pts] Derive the forward and backward propagation equations for the cross-entropy loss function.

Hint: The gradient update equations for the sum of squares loss can be found in p396 of Elements of Statistical Learning, *which is used for regression tasks. Here we consider the classification task and replace the sum of squares loss by the cross-entropy loss. You should highlight in your solution what parts have changed.*

For cross-entropy deviance, we have

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

and the corresponding classifier is $G(x) = \arg\max_k f_k(x)$

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$

$z_i = (z_{1i}, z_{2i}, \cdots, z_{Mi})$, then we have

$$\begin{aligned} R(\theta) &= \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} (-y_{ik}\log(f_k(x_i))), \end{aligned}$$

with derivatives

$$\begin{cases} \dfrac{dR_i}{d\beta_{km}} = -\dfrac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i) z_{mi}, \\ \dfrac{dR_i}{d\alpha_{ml}} = -\sum\limits_{k=1}^{K} \dfrac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_{0m} + \alpha_m^T x_i) x_{il} \end{cases} \quad (1)$$

Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

$$\begin{cases} \beta_{km}^{(r+1)} = \beta_{km}^{r} - \gamma_r \sum_{i=1}^{N} \frac{dR_i}{d\beta_{km}^{r}} \\ \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{dR_i}{d\alpha_{ml}^{(r)}} \end{cases}$$

where γ_r is the learning rate, we write (1) as

$$\begin{cases} \frac{dR_i}{d\beta_{km}} = g_{ki} z_{mi} \\ \frac{dR_i}{d\alpha_{ml}} = s_{mi} x_{il} \end{cases}$$

From their definations, we have

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

known as back-propagation equations.

Problem 2.

[2pts] In a K-class classification problem, suppose that for given neural network parameter (weights) θ and input x, the output of the network are functions $f_k(x, \theta)$, $k = 1, \ldots,$ which are nonnegative numbers that sum to 1. Show that minimizing the cross-entropy loss is equivalent to maximum likelihood estimation, if $f_k(x, \theta)$ is interpreted as the likelihood $P(Y = k|x, \theta)$.

In a K-class classification problem, we are given a neural-network that takes an input x and produces K non-negative numbers as outputs.

These outputs can be interpreted as probabilities of the input belonging to each of the K classes. Let $f(x, 0)$ denote the output of the network when its weights are set to 0, i.e. the network is not trained.

Now suppose we have a training set of inputs x_i and their corresponding true labels y_i, where y_i is an integer in the range 1 to k indicating the true classes of x_i.

Let $p(y_i \mid x_i, w) = \dfrac{f(x_i, w)_{y_i}}{\sum_{k=1}^{K} f(x_i, w)_k}$

where $f(x_i, w)_k$ denotes the kth output of the network for input x_i when its weights are set to w.

The maximum-likelihood estimate of the weights w is obtained by maximizing the likelihood of the training set given the weights.

Assuming the training examples are independent and identically distributed, the likelihood is given by the product of the probabilities assigned by the network to the true labels:

$$L(w) = \prod_{i=1}^{N} p(y_i \mid x_i, w)$$

where N is the size of the training set.

Maximizing the likelihood is equivalent to minimizing the negative log-likelihood:

$$J(w) = -\sum_{i=1}^{N} (\log (p(y_i \mid x_i, w))$$

The negative log-likelihood is equivalent to the cross entropy loss, which is defined as follows:

$$J(w) = -\sum_{i=1}^{N} (y_i \times \log(p(y_i \mid X_i, w))$$

where y_i is a one-hot vector representing the true label of X_i, i.e. y_i has a 1 in the position corresponding to the true class of X_i and 0s elsewhere.

Problem 3.

Train a one hidden layer neural network with scalar input and output $(X_i, Y_i)_{i=1}^n$. Consider a toy problem setting where the activation function degenerates to a linear (identity) function[1], so that for any input x, the network output is calculated as

$$z_j := \alpha_j x; \tag{1}$$

$$f(x) := \frac{1}{\sqrt{p}} \sum_{j=1}^{p} \beta_j z_j \tag{2}$$

where z_j's are neurons in the hidden layer and $(\alpha_j)_{j=1}^p$ and $(\beta_j)_{j=1}^p$ are the weights. Define the loss function $E := \sum_{i=1}^n (Y_i - f(X_i))^2$. Suppose that the gradient descent step size is small, so that we focus on the continuous-time version of backpropagation:

$$\frac{d\alpha_j}{dt} = -\frac{\partial E}{\partial \alpha_j}; \tag{3}$$

$$\frac{d\beta_j}{dt} = -\frac{\partial E}{\partial \beta_j}. \tag{4}$$

1. [2pt] Express the right sides of (3) and (4) in terms of X_i, α_j, β_j, and the residue $R_i := Y_i - f(X_i)$ $(i = 1, \ldots, n)$.
2. [1pt] Show that when training completes (reaches a stationary point), the neural net function $\hat{f}(x) = \frac{1}{\sqrt{p}} \sum_{j=1} \hat{\alpha}_j \hat{\beta}_j x$ coincides with the least squares estimator. *Hint: set the right sides of (3) and (4) to zero.*
3. [1pt bonus] Show that for each j, $\alpha_j^2 - \beta_j^2$ remains a constant during training. *Hint: take the ratio of (3) to (4) and use part 1).*

1. $$\frac{dE}{d\alpha_j} = \frac{d}{d\alpha_j}\sum_{i=1}^{n}(Y_i - f(X_i))^2 = -2\sum_{i=1}^{n}(Y_i - f(X_i))\frac{df(X_i)}{d\alpha_j}$$

$$= -\frac{2}{Jp}\sum_{i=1}^{n}(Y_i - f(X_i))\beta_j X_i = -\frac{2}{Jp}\sum_{i=1}^{n} R_i \beta_j X_i$$

Similarity,

$$\frac{dE}{d\beta_j} = \frac{d}{d\beta_j}\sum_{i=1}^{n}(Y_i - f(X_i))^2 = -2\sum_{i=1}^{n}(Y_i - f(X_i))\frac{df(X_i)}{d\beta_j}$$

$$= -\frac{2}{Jp}\sum_{i=1}^{n}(Y_i - f(X_i))Z_j = -\frac{2}{Jp}\sum_{i=1}^{n}(Y_i - f(X_i))\alpha_j X_i$$

$$= -\frac{2}{Jp}\sum_{i=1}^{n} R_i \alpha_j X_i$$

Summary:

$$\begin{cases} \dfrac{d\alpha_j}{dt} = -\dfrac{dE}{d\alpha_j} = \dfrac{2}{Jp}\displaystyle\sum_{i=1}^{n} R_i \beta_j X_i \\ \dfrac{d\beta_j}{dt} = -\dfrac{dE}{d\beta_j} = \dfrac{2}{Jp}\displaystyle\sum_{i=1}^{n} R_i \alpha_j X_i \end{cases} *$$

2. To show that when the training completes, the neural net function

$$\hat{f}(x) = \frac{1}{Jp}\sum_{j=1} \hat{\alpha}_j \hat{\beta}_j X$$

coincides with the least square estimators, we need to show that, at the stationary point, the weight $\hat{\alpha}_j$ and $\hat{\beta}_j$ have converged to the values that minimize the least square loss function.

At the stationary point, the gradient descent ③ and ④ reduce to

$$\begin{cases} \dfrac{d\hat{\alpha}_j}{dt} = -\dfrac{dz}{d\alpha_j} = \dfrac{2}{J_p}\sum_{i=1}^{n} \hat{\beta}_j x_i R_i \\ \dfrac{d\hat{\beta}_j}{dt} = -\dfrac{dz}{d\beta_j} = \dfrac{2}{J_p}\sum_{i=1}^{n} \hat{\alpha}_j x_i R_i \end{cases}$$

Setting these two equations to zero, we get

$$\begin{cases} \sum_{i=1}^{n} x_i R_i \hat{\beta}_j = 0 \\ \sum_{i=1}^{n} x_i R_i \hat{\alpha}_j = 0 \end{cases} \Rightarrow \begin{cases} \dfrac{dz}{d\alpha_j} = 0 \\ \dfrac{dz}{d\beta_j} = 0 \end{cases} \quad - \Phi$$

These equations are equivalent to the equations for the least square problem. To see this, let's rewrite $f(x)$

$$\hat{f}(x) = \frac{1}{J_p}\sum_{j=1}^{p} \hat{\alpha}_j \hat{\beta}_j x$$

As for least square method,

$$\begin{cases} \frac{d}{d\alpha_j} \sum_{i=1}^{n} (Y_i - f(x_i)) = \frac{dE}{d\alpha_j} = 0 \\ \frac{d}{d\beta_j} \sum_{i=1}^{n} (Y_i - f(x_i)) = \frac{dE}{d\beta_j} = 0 \end{cases} \quad - (2)$$

(1) = (2). Hence, the solution $\hat{\alpha}_j$, $\hat{\beta}_j$ for the least square estimator are the same with neural net function. Therefore, the neural net function

$\hat{f}(x) = \frac{1}{J_p} \sum_{j=1}^{p} \hat{\alpha}_j \hat{\beta}_j x$ coincides with the least square estimator.

Where $\alpha_j^{(0)}$ and $\beta_j^{(0)}$ are the initial values of α_j and β_j, simplify the LHS, we get

$$\ln\left(\frac{\alpha_j}{\alpha_j^{(0)}}\right) = \frac{1}{J_p} \sum_{i} \frac{\beta_j^{(0)} x_{ij} R_i}{\beta_j} - \frac{1}{J_p} \sum_{i} \frac{\beta_j x_{ij} R_i}{\beta_j^{(0)}}$$

Expand β_j and $\beta_j^{(0)}$ in terms of α_j and $\alpha_j^{(0)}$ using the expressions derived in prob 1, we obtain

$$\ln\left(\frac{\alpha_j}{\alpha_j^{(0)}}\right) = \frac{1}{J_p} \sum_{i} \frac{\alpha_j^{(0)} x_{ij} R_i}{\alpha_j} - \frac{1}{J_p} \sum_{i} \frac{\alpha_j x_{ij} R_i}{\alpha_j^{(0)}}$$

Simplify further, we get

$$\ln\left(\frac{\alpha_j}{\alpha_j^{(0)}}\right) = \frac{2}{Jp}\sum_i \frac{\alpha_j^{(0)} x_{ij} R_i}{\alpha_j} - \frac{2}{Jp}\sum_i \frac{\alpha_j x_{ij} R_i}{\alpha_j^{(0)}}$$

Using the fact that $\alpha_j - \beta_j$ is a constant during training, we can write

$$\alpha_j - \beta_j = \frac{4}{p}\sum_i \alpha_j^{(0)} \alpha_j x_{ij} R_i - \frac{4}{p}\sum_i \sqrt{\alpha_j^{(0)} \alpha_j}\, x_{ij} R_i$$

Simplify further, we get

$$(\alpha_j - \beta_j) - (\alpha_j^{(0)} - \beta_j^{(0)}) = \frac{4}{p}\sum_i (\alpha_j^{(0)} \alpha_j - \beta_j^{(0)} \beta_j) x_{ij} R_i$$

Since $(\alpha_j^{(0)} - \beta_j^{(0)})$ is a constant, we conclude that $(\alpha_j - \beta_j)$ is as well constant during training.

Using the equation $\hat{\alpha}_j = \sqrt{p}\, w_j$, we can rewrite as

$$\frac{\hat{\alpha}_j}{Jp} x_{ij} R_i$$

which is proportional to the expression for $\frac{d\hat{\beta}_j}{dt}$ in (4) at the stationary point. Similarly, we can show that the expression for $\hat{\alpha}_j$ at the stationary point

is proportional to the solution of the normal equations for the least square problem.

Therefore, at the stating point (train completes), the weights $\hat{\alpha}_j$ and $\hat{\beta}_j$ have converged to the values that minimize the least square loss function, and the neural network $\hat{f}(x)$ coincides with the least square estimator.

3, Take the ratio of (3) to (4), we have

$$\frac{d\alpha_j}{d\beta_j} = \sum_i \frac{x_{ij} R_i}{T_p \alpha_j}$$

Integrating both sides wrt β_j, we obtain

$$\int_{\alpha_j^{(0)}}^{\alpha_j} \frac{d\alpha_j'}{T_p \alpha_j'} = \int_{\beta_j^{(0)}}^{\beta_j} \frac{\sum x_{ij} R_i}{\beta_j'} d\beta_j'$$

We have the result above.

Let $r_j = \alpha_j - \beta_j$, then

$$\frac{d}{dt} r_j = \frac{d}{dt}(\alpha_j^2 - \beta_j^2)$$

$$= 2\alpha_j \left(\sum_{i=1}^n (Y_i - f(x_i)) \beta_j x_i\right) - 2\beta_j \left(\sum_{i=1}^n (Y_i - f(x_i)) \alpha_j x_i\right)$$

$$= -2 \sum_{i=1}^n (Y_i - f(x_i)) r_j x_i^2$$

$\Rightarrow$ for each j, $\frac{d}{dt} r_j$ is not depends on j

$\Rightarrow \alpha_j^2 - \beta_j^2$ remain constant for each j.

4. At initialization, $\alpha_j = \alpha^0$, $\beta_j = \beta^0$

Let $\mathcal{E}^0 = \mathcal{E}(\alpha^0, \beta^0)$.

At stationary point, we have

$$\frac{d\alpha_j}{dt} = \frac{d\beta_j}{dt} = 0 \quad \text{for all } j\text{'s.}$$

$\Rightarrow$

$$\sum_{i=1}^n (Y_i - f(x_i)) \beta_j x_i$$

$$+) \quad \sum_{i=1}^n (Y_i - f(x_i)) \alpha_j x_i \qquad \text{for all } j\text{'s}$$

$$\downarrow$$

$$\frac{d}{dt}(\alpha_j + \beta_j) = \frac{2}{\sqrt{p}} \sum_{i=1}^n (\alpha_j + \beta_j) x_i R_i$$

$$= \frac{2}{\sqrt{p}} (\alpha_j + \beta_j) S$$

By observation, we have the general solution $\alpha_j + \beta_j = C e^{2tS/\sqrt{p}}$, where C is constat that depends on the initial values of α_j and β_j.

7.

```python
import numpy as np

# Define the linear activation function
def sigma(x):
    return x

# Define the loss function
def loss(X, Y, alpha, beta):
    Z = sigma(np.outer(alpha, X))
    f = np.sum(beta * Z, axis=1) / np.sqrt(len(alpha))
    return np.sum((Y - f) ** 2)

# Define the gradient of the loss function with respect to alpha
def grad_alpha(X, Y, alpha, beta):
    Z = sigma(np.outer(alpha, X))
    f = np.sum(beta * Z, axis= 1) / np.sqrt(len(alpha))
    R = Y - f
    return -2 * np.sum(np.outer(R, X) * beta * np.sqrt(len(alpha)) * Z, axis = 0)

# Define the gradient of the loss function with respect to beta
def grad_beta(X, Y, alpha, beta):
    Z = sigma(np.outer(alpha, X))
    f = np.sum(beta * Z, axis = 1) / np.sqrt(len(alpha))
    R = Y - f
    return -2 * np.sum(np.outer(R, Z) / np.sqrt(len(alpha)), axis = 0)

# Initialize the weights
p = 1000
alpha = np.random.normal(loc = 0, scale = 1, size = p)
beta = np.random.normal(loc = 0, scale = 1, size = p)


# Generate some toy data
n = 1000
X = np.random.normal(loc = 0, scale = 1, size = n)
Y = 2 * X + np.random.normal(loc = 0, scale = 0.5, size = n)

# Set the learning rate and the number of iterations
eta = 1e-3
num_iters = 100

# Train the neural network using gradient descent
for i in range(num_iters):
    alpha -= eta * grad_alpha(X, Y, alpha, beta)
    beta -= eta * grad_beta(X, Y, alpha, beta)

# Compute the final estimate of the function
def f_hat(x):
    Z = sigma(alpha * x)
    return np.sum(beta * Z) / np .sqrt(p)

# Test the final estimate on some new data
X_test = np.random.normal (loc=0, scale=1, size = n)
Y_test = 2 * X_test + np.random.normal (loc=0, scale = 0.5, size = n)
predictions = np.array([f_hat(x) for x in X_test])
mse = np.mean((Y_test - predictions) ** 2)
print('MSE:', mse)
```

This code generates a random toy dataset of n = 1000 samples, where Y is a noisy version of 2X. It then initializes the weights of the neural network with random normal variables and trains the network using gradient descent with a learning rate of 1e-3 for 10000 iterations. Finally, it computes the final estimate of the function f(x) and tests it on a new set of data, computing the mean squared error between the true Y values and the predicted values.

8.1

No, it does not necessarily contradict the fact that f(x) changes noticeably during training. While each individual weight does not change much during training, the overall effect of changing all the weights together can still result in a significant change in the output function. In other words, even small changes in each weight can accumulate and result in a noticeable change in the overall function.

To be more specific, the fact that individual weights change very little during training does not necessarily imply that the overall function f(x) does not change significantly. In this case, the change in f(x) from x to 2x is due to the combined effect of all the weights and inputs.
As p->∞, the number of weights in the network becomes very large, and each weight contributes only a small amount to the overall function f(x). However, since there are many weights, the combined effect of all the weights can still be significant, leading to a large change in f(x).

Moreover, the fact that c->1 as p->∞ indicates that the weights are becoming increasingly balanced, with the magnitudes of the aj and Bj becoming closer together. This means that the contribution of each weight to the overall function f(x) becomes more equal, further emphasizing the combined effect of all the weights.

the defination of the function $f(x)$. Specifically, we have:

$$f(x) = \frac{1}{\sqrt{p}} \sum_{j=1}^{p} B_j z_j = \frac{1}{p} \sum_{j=1}^{p} B_j \alpha_j x = \frac{1}{p} \sum_{j=1}^{p} \frac{B_j}{\alpha_j} \alpha_j^2 x$$

$$= \frac{1}{p} \sum_{j=1}^{p} \frac{B_j}{\alpha_j} z_j'$$

where $z_j' := \alpha_j^2 x$ is the activation of jth neuron. after applying the linear transformation α_j.

9. Using the same argument as in part 7, we can show that during training, $\alpha_j^2 - \beta_j^2$ remains constant for all j, and that $\alpha_j + \beta_j$ converges to some constant $c > 0$.

To compute value of c, we can take the sum of the equations for $\frac{d\alpha_j}{dt}$ and $\frac{d\beta_j}{dt}$

$$\sum_{j=1}^{p}\left(\frac{d\alpha_j}{dt} + \frac{d\beta_j}{dt}\right) = -2\sum_{i=1}^{n} R_i x_i = -2\sum_{i=1}^{n}(y_i - f(x_i))x_i$$

$$= -2\sum_{i=1}^{n}\left(y_i - \frac{1}{p}\sum \frac{\beta_j}{\alpha_j} z_j'\right)x_i = -2\sum_{i=1}^{n}(y_i - f'(x_i))x_i$$

Where $f'(x) = \frac{1}{p}\sum_{j=1}^{p}\frac{\beta_j}{\alpha_j}\alpha_j x$ is the intermediate funtion we defind above.

Taking the derivative of the loss function with respect to α_j and β_j, we get.

$$\frac{dL}{d\alpha_j} = -2\sum_{i=1}^{n}(y_i - f(x_i))\frac{\beta_j}{\alpha_j}\alpha_j x_i$$

$$= -2\sum_{i=1}^{n}(y_i - f(x_i))\alpha_j^2 x_i$$

↓

Setting rhs of these equations to zero, we get:

$$\sum_{i=1}^{n} (Y_i - f(X_i)) B_j X_i = 0$$