# Deep Learning

homework 2

Amirparsa Bahrami 401101332
github repository

# Q1

## 1. Restate the Cost Function

The cost function for the binary classification problem using a single neuron with a sigmoid activation function is:

$$E(w,b) = -\sum_n [y_n \ln \hat{y}_n + (1-y_n)\ln(1-\hat{y}_n)]$$

where:

- $y_n$ is the true label for the $n$-th data point ($y_n \in \{0,1\}$),
- $\hat{y}_n$ is the predicted probability from the model,
- $\hat{y}_n = \sigma(z_n) = \frac{1}{1+e^{-z_n}}$,
- $z_n = wx_n + b$.

## 2. Compute the Gradient of the Cost Function

The gradient vector $\nabla E(w,b)$ consists of the partial derivatives of $E$ with respect to $w$ and $b$:

### *2.1 Partial Derivative with Respect to* $w$

$$\frac{\partial E}{\partial w} = \sum_n (\hat{y}_n - y_n)\, x_n$$

Derivation:

1. Compute $\frac{\partial E}{\partial \hat{y}_n} = -\left(\frac{y_n}{\hat{y}_n} - \frac{1-y_n}{1-\hat{y}_n}\right)$.
2. Compute $\frac{\partial \hat{y}_n}{\partial z_n} = \hat{y}_n(1-\hat{y}_n)$.
3. Using the chain rule:

$$\frac{\partial E}{\partial w} = \sum_n \frac{\partial E}{\partial \hat{y}_n} \cdot \frac{\partial \hat{y}_n}{\partial z_n} \cdot \frac{\partial z_n}{\partial w} = \sum_n ((\hat{y}_n - y_n)x_n)$$

### *2.2 Partial Derivative with Respect to* $b$

$$\frac{\partial E}{\partial b} = \sum_n (\hat{y}_n - y_n)$$

Derivation:

1. Compute $\frac{\partial z_n}{\partial b} = 1$.
2. Using the chain rule:

$$\frac{\partial E}{\partial b} = \sum_n \frac{\partial E}{\partial \hat{y}_n} \cdot \frac{\partial \hat{y}_n}{\partial z_n} \cdot \frac{\partial z_n}{\partial b} = \sum_n (\hat{y}_n - y_n)$$

# 3. Compute the Hessian Matrix

The Hessian matrix $H$ is a square matrix of second-order partial derivatives:

$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial w^2} & \frac{\partial^2 E}{\partial w \partial b} \\ \frac{\partial^2 E}{\partial b \partial w} & \frac{\partial^2 E}{\partial b^2} \end{bmatrix}$$

## *3.1 Compute* $\frac{\partial^2 E}{\partial w^2}$

$$\frac{\partial^2 E}{\partial w^2} = \sum_n \left(\hat{y}_n(1 - \hat{y}_n)x_n^2\right)$$

## *3.2 Compute* $\frac{\partial^2 E}{\partial w \partial b}$ *and* $\frac{\partial^2 E}{\partial b \partial w}$

Since the mixed partial derivatives are equal (by Schwarz's theorem):

$$\frac{\partial^2 E}{\partial w \partial b} = \frac{\partial^2 E}{\partial b \partial w} = \sum_n (\hat{y}_n(1 - \hat{y}_n)x_n)$$

## *3.3 Compute* $\frac{\partial^2 E}{\partial b^2}$

$$\frac{\partial^2 E}{\partial b^2} = \sum_n (\hat{y}_n(1 - \hat{y}_n))$$

# 4. Construct the Hessian Matrix

Combining the above results:

$$H = \sum_n \hat{y}_n(1 - \hat{y}_n) \begin{bmatrix} x_n^2 & x_n \\ x_n & 1 \end{bmatrix}$$

# 5. Show that the Hessian is Positive Semidefinite

To prove that $H$ is positive semidefinite, we need to show that for any non-zero vector $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, the quadratic form $v^T H v \geq 0$.

## *5.1 Consider the Quadratic Form*

$$v^T H v = \sum_n \hat{y}_n(1 - \hat{y}_n) \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} x_n^2 & x_n \\ x_n & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

## *5.2 Simplify the Quadratic Form*

$$v^T H v = \sum_n \hat{y}_n(1 - \hat{y}_n)(v_1 x_n + v_2)^2$$

Since $\hat{y}_n(1 - \hat{y}_n) \geq 0$ (because $\hat{y}_n \in (0, 1)$):

- Each term $\hat{y}_n(1-\hat{y}_n)(v_1x_n+v_2)^2 \geq 0$.
- The sum of non-negative terms is non-negative.

Therefore:

$$v^T H v \geq 0$$

This implies that $H$ is positive semidefinite.

# 6. Conclusion on Convexity

Since the Hessian $H$ is positive semidefinite, the cost function $E(w,b)$ is convex with respect to $w$ and $b$.

# Part 2: Deriving a Recursive Relation to Reach the Minimum Point

To find the minimum point, we'll derive the gradient descent update rules for $w$ and $b$.

## *Step 1: Compute the Gradients*

**1. Gradient with respect to $w$:**

$$\frac{\partial E}{\partial w} = -\sum_n \left( y_n \frac{\partial}{\partial w} \ln \hat{y}_n + (1-y_n) \frac{\partial}{\partial w} \ln(1-\hat{y}_n) \right)$$

Compute derivatives:

$$\frac{\partial}{\partial w} \ln \hat{y}_n = (1-\hat{y}_n)x_n$$

$$\frac{\partial}{\partial w} \ln(1-\hat{y}_n) = -\hat{y}_n x_n$$

Substitute back:

$$\frac{\partial E}{\partial w} = -\sum_n \left( y_n(1-\hat{y}_n)x_n - (1-y_n)\hat{y}_n x_n \right)$$

$$\frac{\partial E}{\partial w} = -\sum_n \left( y_n x_n - y_n \hat{y}_n x_n - \hat{y}_n x_n + y_n \hat{y}_n x_n \right)$$

$$\frac{\partial E}{\partial w} = -\sum_n \left( y_n x_n - \hat{y}_n x_n \right) = -\sum_n (y_n - \hat{y}_n)x_n$$

**2. Gradient with respect to $b$:**

Following a similar approach:

$$\frac{\partial E}{\partial b} = -\sum_n (y_n - \hat{y}_n)$$

## *Step 2: Develop the Recursive Update Rules*

Using gradient descent, the parameters are updated as:

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial E}{\partial w}$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial E}{\partial b}$$

where $\eta$ is the learning rate.
Substitute the gradients:
**Update rule for $w$:**

$$w^{(k+1)} = w^{(k)} - \eta \left( - \sum_n (y_n - \hat{y}_n) x_n \right)$$

$$w^{(k+1)} = w^{(k)} + \eta \sum_n (y_n - \hat{y}_n) x_n$$

**Update rule for $b$:**

$$b^{(k+1)} = b^{(k)} - \eta \left( - \sum_n (y_n - \hat{y}_n) \right)$$

$$b^{(k+1)} = b^{(k)} + \eta \sum_n (y_n - \hat{y}_n)$$

## ***Recursive Relations***

The recursive relations to update $w$ and $b$ are:

$$w^{(k+1)} = w^{(k)} + \eta \sum_n (y_n - \hat{y}_n) x_n$$

$$b^{(k+1)} = b^{(k)} + \eta \sum_n (y_n - \hat{y}_n)$$

# Q2

## Covariate Shift in Neural Networks and How Batch Normalization Resolves It

Covariate shift in neural networks refers to the change in the distribution of input data that each layer receives during training. As the parameters of earlier layers update, the distribution of outputs they produce—and hence the inputs to subsequent layers—can shift. This continuous change forces each layer to adapt to new input distributions, slowing down training and potentially leading to convergence issues.

Batch Normalization (BN) addresses covariate shift by normalizing the inputs to each layer. Specifically, BN standardizes the inputs by subtracting the batch mean and dividing by the batch standard deviation:

$$\hat{x}_i = \frac{x_i - \mu}{\sigma}$$

where $\mu$ and $\sigma$ are the mean and standard deviation computed over the mini-batch. BN stabilizes the input distributions across layers, allowing each layer to learn from inputs with a consistent distribution. This normalization mitigates the internal covariate shift, accelerates training, and enables the use of higher learning rates.

## How Batch Normalization Enhances Generalization

Batch Normalization improves the generalization of neural networks through several mechanisms:

- **Regularization Effect:** The use of mini-batches introduces noise in the estimation of $\mu$ and $\sigma$, which acts as a regularizer. This stochasticity can prevent overfitting by discouraging the network from becoming too tailored to the training data.
- **Smoother Loss Landscape:** By stabilizing the input distributions, BN creates a smoother and more predictable loss surface. This smoothness facilitates optimization algorithms in finding better minima that generalize well to unseen data.
- **Improved Gradient Flow:** BN alleviates issues like vanishing or exploding gradients by maintaining consistent input scales across layers. This consistency ensures that gradients propagate effectively during backpropagation.
- **Higher Learning Rates:** The normalization allows for the use of higher learning rates without risking divergence, enabling faster convergence and potentially better generalization.

## Calculating $\frac{\partial L}{\partial x_i}$ in Terms of $\frac{\partial L}{\partial y_j}$

Given:

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \text{and} \quad \hat{x}_i = x_i - \mu$$

$$y_i = \gamma \hat{x}_i + \beta$$

To find $\frac{\partial L}{\partial x_i}$, we consider how $x_i$ influences the loss $L$ through $y_j$:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \left( \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i} \right)$$

### *Computing* $\frac{\partial y_j}{\partial x_i}$*:*

1. Compute $\frac{\partial \hat{x}_j}{\partial x_i}$:

   Since $\hat{x}_j = x_j - \mu$ and $\mu = \frac{1}{n}\sum_{k=1}^{n} x_k$,

   $$\frac{\partial \mu}{\partial x_i} = \frac{1}{n}$$

   $$\frac{\partial \hat{x}_j}{\partial x_i} = \delta_{ij} - \frac{1}{n}$$

   where $\delta_{ij}$ is the Kronecker delta (1 if $i = j$, 0 otherwise).

2. Compute $\frac{\partial y_j}{\partial x_i}$:

   $$\frac{\partial y_j}{\partial x_i} = \gamma \cdot \frac{\partial \hat{x}_j}{\partial x_i} = \gamma \left(\delta_{ij} - \frac{1}{n}\right)$$

   Substitute back into $\frac{\partial L}{\partial x_i}$:

   $$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \left(\frac{\partial L}{\partial y_j} \cdot \gamma \left(\delta_{ij} - \frac{1}{n}\right)\right)$$

Simplify the expression:

$$\frac{\partial L}{\partial x_i} = \gamma \left(\frac{\partial L}{\partial y_i} - \frac{1}{n}\sum_{j=1}^{n} \frac{\partial L}{\partial y_j}\right)$$

## Evaluating $\frac{\partial L}{\partial x_i}$ for $n = 1$ and $n \to \infty$

### *Case 1: When* $n = 1$

With only one data point:

- The mean $\mu = x_1$, so $\hat{x}_1 = x_1 - x_1 = 0$.
- The output $y_1 = \gamma \cdot 0 + \beta = \beta$, which is constant with respect to $x_1$.

Computing the gradient:

$$\frac{\partial L}{\partial x_1} = \gamma \left(\frac{\partial L}{\partial y_1} - \frac{1}{1} \cdot \frac{\partial L}{\partial y_1}\right) = \gamma \left(\frac{\partial L}{\partial y_1} - \frac{\partial L}{\partial y_1}\right) = 0$$

Result: $\frac{\partial L}{\partial x_1} = 0$

### *Case 2: When* $n \to \infty$

As $n$ becomes very large:

- The term $\frac{1}{n}\sum_{j=1}^{n} \frac{\partial L}{\partial y_j}$ approaches the expected value $E\left[\frac{\partial L}{\partial y}\right]$.

Assuming that the average gradient $\frac{1}{n}\sum_{j=1}^{n} \frac{\partial L}{\partial y_j}$ remains bounded, it becomes negligible compared to individual $\frac{\partial L}{\partial y_i}$. Computing the gradient:

$$\frac{\partial L}{\partial x_i} = \gamma \left(\frac{\partial L}{\partial y_i} - \frac{1}{n}\sum_{j=1}^{n} \frac{\partial L}{\partial y_j} \approx 0 \text{ as } n \to \infty\right) \approx \gamma \cdot \frac{\partial L}{\partial y_i}$$

Result: $\frac{\partial L}{\partial x_i} \approx \gamma \cdot \frac{\partial L}{\partial y_i}$

## ***Summary of Results***

- For $n = 1$: $\frac{\partial L}{\partial x_i} = 0$
- For $n \to \infty$: $\frac{\partial L}{\partial x_i} \approx \gamma \cdot \frac{\partial L}{\partial y_i}$

These results highlight how the size of the mini-batch affects the gradient during backpropagation in Batch Normalization. With small batches, normalization can dominate, potentially hindering learning (zero gradient). With large batches, normalization has less impact, and gradients resemble those of the unnormalized inputs.

# Q3

**(a) Compute $\frac{\partial \hat{y}_k}{\partial z_i^{(2)}}$ and simplify your answer in terms of $\hat{y}$.**

**Solution:**

Recall the softmax function:

$$\hat{y}_k = \frac{e^{z_k^{(2)}}}{\sum_{j=1}^{K} e^{z_j^{(2)}}}$$

Let $S = \sum_{j=1}^{K} e^{z_j^{(2)}}$. Then,

$$\hat{y}_k = \frac{e^{z_k^{(2)}}}{S}$$

To compute $\frac{\partial \hat{y}_k}{\partial z_i^{(2)}}$, we differentiate $\hat{y}_k$ with respect to $z_i^{(2)}$:

$$\frac{\partial \hat{y}_k}{\partial z_i^{(2)}} = \frac{\partial}{\partial z_i^{(2)}} \left( \frac{e^{z_k^{(2)}}}{S} \right) = \frac{e^{z_k^{(2)}}}{S} \left( \delta_{ik} - \frac{e^{z_i^{(2)}}}{S} \right) = \hat{y}_k \left( \delta_{ik} - \hat{y}_i \right)$$

where $\delta_{ik}$ is the Kronecker delta, equal to 1 if $i = k$ and 0 otherwise.

**Answer:**

$$\frac{\partial \hat{y}_k}{\partial z_i^{(2)}} = \hat{y}_k \left( \delta_{ik} - \hat{y}_i \right)$$

**(b) Assume that the $k$-th element of the label vector is equal to 1 and the rest of its elements are zero ($y_k = 1$ and $y_i = 0$ for $i \neq k$). Compute $\frac{\partial L}{\partial z_i^{(2)}}$ and simplify your answer in terms of $\hat{y}$.**

**Solution:**

Given $y_k = 1$ and $y_i = 0$ for $i \neq k$, the loss function simplifies to:

$$L = -\log(\hat{y}_k)$$

Our goal is to compute $\frac{\partial L}{\partial z_i^{(2)}}$ for each $i \in \{1, 2, \ldots, K\}$.

**Step 1: Compute $\frac{\partial L}{\partial \hat{y}_j}$**

$$\frac{\partial L}{\partial \hat{y}_j} = \begin{cases} -\frac{1}{\hat{y}_k}, & \text{if } j = k \\ 0, & \text{if } j \neq k \end{cases}$$

**Step 2: Use the result from part (a) for $\frac{\partial \hat{y}_j}{\partial z_i^{(2)}}$**

$$\frac{\partial \hat{y}_j}{\partial z_i^{(2)}} = \hat{y}_j \left( \delta_{ij} - \hat{y}_i \right)$$

**Step 3: Compute $\frac{\partial L}{\partial z_i^{(2)}}$ using the chain rule**

$$\frac{\partial L}{\partial z_i^{(2)}} = \sum_{j=1}^{K} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(2)}} = \left( -\frac{1}{\hat{y}_k} \right) \hat{y}_k \left( \delta_{ik} - \hat{y}_i \right) = -\left( \delta_{ik} - \hat{y}_i \right) = -\delta_{ik} + \hat{y}_i$$

Since $y_i = \delta_{ik}$, we can write:

$$\frac{\partial L}{\partial z_i^{(2)}} = \hat{y}_i - y_i$$

**Answer:**

$$\frac{\partial L}{\partial z_i^{(2)}} = \hat{y}_i - y_i$$

**(c) Calculate $\frac{\partial L}{\partial W^{(1)}}$.**

**Solution:**

To compute $\frac{\partial L}{\partial W^{(1)}}$, we'll propagate the gradients backward through the network layers.

**Step 1: Compute $\frac{\partial L}{\partial z^{(2)}}$**

$$\frac{\partial L}{\partial z^{(2)}} = \hat{y} - y$$

This is a vector of size $K \times 1$.

**Step 2: Compute $\frac{\partial L}{\partial a^{(1)}}$**

$$\frac{\partial L}{\partial a^{(1)}} = (W^{(2)})^\top \frac{\partial L}{\partial z^{(2)}}$$

**Step 3: Account for Dropout**

The dropout layer randomly sets a fraction $p = 0.2$ of the activations to zero during training. Let $m$ be the dropout mask applied during the forward pass, where $m_i = 0$ if neuron $i$ was dropped and $m_i = 1$ otherwise. Then,

$$\frac{\partial L}{\partial \hat{a}^{(1)}} = m \odot \frac{\partial L}{\partial a^{(1)}}$$

where $\odot$ denotes element-wise multiplication.

**Step 4: Compute $\frac{\partial L}{\partial z^{(1)}}$ through LeakyReLU**

The LeakyReLU activation function is defined as:

$$\hat{a}_i^{(1)} = \begin{cases} z_i^{(1)}, & \text{if } z_i^{(1)} > 0 \\ \alpha z_i^{(1)}, & \text{if } z_i^{(1)} \leq 0 \end{cases}$$

where $\alpha = 0.01$.

The derivative of LeakyReLU is:

$$\text{LeakyReLU}'(z_i^{(1)}) = \begin{cases} 1, & \text{if } z_i^{(1)} > 0 \\ \alpha, & \text{if } z_i^{(1)} \leq 0 \end{cases}$$

Therefore:

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial \hat{a}^{(1)}} \odot \text{LeakyReLU}'(z^{(1)})$$

**Step 5: Compute $\frac{\partial L}{\partial W^{(1)}}$**

Using the chain rule:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z^{(1)}} x^\top$$

This computes the outer product of $\frac{\partial L}{\partial z^{(1)}}$ (a column vector of size $d_a \times 1$) and $x^\top$ (a row vector of size $1 \times d_x$), resulting in a matrix of size $d_a \times d_x$.

**Final Expression:**

$$\frac{\partial L}{\partial W^{(1)}} = \left[(W^{(2)})^\top (\hat{y} - y) \odot m \odot \text{LeakyReLU}'(z^{(1)})\right] x^\top$$

**Summary of All Parts:**

(a)

$$\frac{\partial \hat{y}_k}{\partial z_i^{(2)}} = \hat{y}_k \left(\delta_{ik} - \hat{y}_i\right)$$

(b)

$$\frac{\partial L}{\partial z_i^{(2)}} = \hat{y}_i - y_i$$

(c)

$$\frac{\partial L}{\partial W^{(1)}} = \left[(W^{(2)})^\top (\hat{y} - y) \odot m \odot \text{LeakyReLU}'(z^{(1)})\right.$$

# Q4

To show that the Hessian matrix of a scalar function $y(u, v, z)$ can be expressed as the Jacobian matrix of its gradient, we'll start by defining both the Hessian matrix and the Jacobian matrix in this context. We'll then demonstrate that these two matrices are indeed equivalent under the given transformation.

# Definitions

1. **Gradient of $y(u, v, z)$:** The gradient vector $\nabla y$ of a scalar function $y$ with respect to variables $u, v, z$ is given by:

$$\nabla y = \begin{bmatrix} \frac{\partial y}{\partial u} \\ \frac{\partial y}{\partial v} \\ \frac{\partial y}{\partial z} \end{bmatrix}$$

2. **Jacobian Matrix of the Gradient $\nabla y$:** The Jacobian matrix $J(\nabla y)$ of the gradient vector $\nabla y$ is a matrix of first-order partial derivatives of the components of $\nabla y$ with respect to $u, v, z$:

$$J(\nabla y) = \begin{bmatrix} \frac{\partial}{\partial u}\left(\frac{\partial y}{\partial u}\right) & \frac{\partial}{\partial v}\left(\frac{\partial y}{\partial u}\right) & \frac{\partial}{\partial z}\left(\frac{\partial y}{\partial u}\right) \\ \frac{\partial}{\partial u}\left(\frac{\partial y}{\partial v}\right) & \frac{\partial}{\partial v}\left(\frac{\partial y}{\partial v}\right) & \frac{\partial}{\partial z}\left(\frac{\partial y}{\partial v}\right) \\ \frac{\partial}{\partial u}\left(\frac{\partial y}{\partial z}\right) & \frac{\partial}{\partial v}\left(\frac{\partial y}{\partial z}\right) & \frac{\partial}{\partial z}\left(\frac{\partial y}{\partial z}\right) \end{bmatrix}$$

3. **Hessian Matrix of $y(u, v, z)$:** The Hessian matrix $H(y)$ of a scalar function $y$ is a square matrix of second-order partial derivatives:

$$H(y) = \begin{bmatrix} \frac{\partial^2 y}{\partial y^2} & \frac{\partial^2 y}{\partial u \partial v} & \frac{\partial^2 y}{\partial u \partial z} \\ \frac{\partial^2 y}{\partial v \partial u} & \frac{\partial^2 y}{\partial y^2} & \frac{\partial^2 y}{\partial v \partial z} \\ \frac{\partial^2 y}{\partial z \partial u} & \frac{\partial^2 y}{\partial z \partial v} & \frac{\partial^2 y}{\partial z^2} \end{bmatrix}$$

# Proof

Our goal is to show that $H(y) = J(\nabla y)$.

## *Step 1: Compute the Jacobian Matrix of the Gradient*

Let's compute the $(i, j)$-th entry of $J(\nabla y)$:

$$[J(\nabla y)]_{ij} = \frac{\partial}{\partial x_j}\left(\frac{\partial y}{\partial x_i}\right) = \frac{\partial^2 y}{\partial x_i \partial x_j}$$

where $x_1 = u$, $x_2 = v$, $x_3 = z$, and $i, j = 1, 2, 3$.

## *Step 2: Observe the Symmetry of Second Partial Derivatives*

Under the assumption that $y(u, v, z)$ is twice continuously differentiable (i.e., $C^2$ class function), the mixed partial derivatives are equal due to Clairaut's theorem:

$$\frac{\partial^2 y}{\partial x_i \partial x_j} = \frac{\partial^2 y}{\partial x_j \partial x_i}$$

## *Step 3: Compare the Hessian Matrix and the Jacobian Matrix*

From the definitions, the $(i, j)$-th entry of the Hessian matrix $H(y)$ is:

$$[H(y)]_{ij} = \frac{\partial^2 y}{\partial x_i \partial x_j}$$

Similarly, the $(i, j)$-th entry of the Jacobian matrix $J(\nabla y)$ is:

$$[J(\nabla y)]_{ij} = \frac{\partial^2 y}{\partial x_i \partial x_j}$$

### *Step 4: Conclude the Equality*

Since both $H(y)$ and $J(\nabla y)$ have identical entries, it follows that:

$$H(y) = J(\nabla y)$$

## Conclusion

We have shown that the Hessian matrix of the scalar function $y(u, v, z)$ is equal to the Jacobian matrix of its gradient $\nabla y$:

$$H(y) = J(\nabla y)$$

This demonstrates that the Hessian matrix can indeed be written in the form of the Jacobian matrix of the gradient of the transformation $\psi(u, v, z) = y(u, v, z)$.

*Note:* This result relies on the continuity and differentiability of the function $y(u, v, z)$. The equality of mixed partial derivatives is a key aspect of this proof, which holds true under standard regularity conditions (i.e., $y$ is at least twice continuously differentiable).

# Q5

**Answer to Part (a):**
First, compute the gradient of the objective function $J_1$ with respect to $W_i$:
Given the error function:

$$J_1 = 0.5 \left( y_d - \sum_{k=1}^{n} \delta_k W_k x_k \right)^2$$

Compute the partial derivative:

$$\frac{\partial J_1}{\partial W_i} = - \left( y_d - \sum_{k=1}^{n} \delta_k W_k x_k \right) \delta_i x_i$$

Let $E = y_d - \sum_{k=1}^{n} \delta_k W_k x_k$. Then,

$$\frac{\partial J_1}{\partial W_i} = -E \delta_i x_i$$

Next, compute the expected value over the random variables $\delta_k$:

$$E \left[ \frac{\partial J_1}{\partial W_i} \right] = -x_i E[E\delta_i]$$

Expanding $E[E\delta_i]$:

$$E[E\delta_i] = E \left[ \left( y_d - \sum_{k=1}^{n} \delta_k W_k x_k \right) \delta_i \right]$$

Simplify the expected value using properties of Gaussian variables:

$$E[E\delta_i] = y_d E[\delta_i] - \sum_{k=1}^{n} W_k x_k E[\delta_k \delta_i]$$

Since $\delta_k \sim \text{Normal}(1, \sigma^2)$, we have:

- $E[\delta_i] = 1$
- $E[\delta_i^2] = 1 + \sigma^2$
- For $k \neq i$, $E[\delta_k \delta_i] = 1$

Compute the expected value:

$$E[E\delta_i] = y_d - W_i x_i (1 + \sigma^2) - \sum_{k \neq i} W_k x_k$$

Simplify:

$$E[E\delta_i] = y_d - \sum_{k=1}^{n} W_k x_k - W_i x_i \sigma^2$$

Therefore, the expected gradient is:

$$E \left[ \frac{\partial J_1}{\partial W_i} \right] = -x_i \left( y_d - \sum_{k=1}^{n} W_k x_k - W_i x_i \sigma^2 \right)$$

Simplify further:

$$E\left[\frac{\partial J_1}{\partial W_i}\right] = \left(\sum_{k=1}^{n} W_k x_k - y_d\right) x_i + \sigma^2 x_i^2 W_i$$

This shows that the expected gradient consists of the usual gradient term plus an additional term proportional to $W_i$:

$$E\left[\frac{\partial J_1}{\partial W_i}\right] = \frac{\partial J_0}{\partial W_i} + \sigma^2 x_i^2 W_i$$

where

$$J_0 = 0.5\left(y_d - \sum_{k=1}^{n} W_k x_k\right)^2$$

is the standard error function without dropout.

**Answer to Part (b):**

Based on the additional term $\sigma^2 x_i^2 W_i$ in the expected gradient, we can propose a modification to the regularization term in the objective function. Specifically, we introduce a regularization term that penalizes the weights scaled by the square of the inputs:

$$\text{Regularization term} = \frac{\sigma^2}{2}\sum_{k=1}^{n} x_k^2 W_k^2$$

The modified (regularized) objective function becomes:

$$J = 0.5\left(y_d - \sum_{k=1}^{n} W_k x_k\right)^2 + \frac{\sigma^2}{2}\sum_{k=1}^{n} x_k^2 W_k^2$$

This regularization term effectively incorporates the impact of Gaussian multiplicative dropout into the learning process by penalizing larger weights more heavily, especially when associated with larger input features $x_k$.

**Final Answer:**

(a) The expected gradient is:

$$E\left[\frac{\partial J_1}{\partial W_i}\right] = \left(\sum_{k=1}^{n} W_k x_k - y_d\right) x_i + \sigma^2 x_i^2 W_i$$

(b) We can modify the regularization term to include the effect of dropout by adding $\frac{\sigma^2}{2}\sum_{k=1}^{n} x_k^2 W_k^2$ to the objective function. The new regularized objective function is:

$$J = 0.5\left(y_d - \sum_{k=1}^{n} W_k x_k\right)^2 + \frac{\sigma^2}{2}\sum_{k=1}^{n} x_k^2 W_k^2$$

# Q6

## *First Part: Convergence of Newton's Method for* $f(x) = g'(x)$

To prove the convergence of Newton's method for the function $f(x) = g'(x)$, we start by considering that $f$ is twice continuously differentiable, $f(x^*) = 0$, and $f'(x^*) \neq 0$ at the point $x^*$.

## *Newton's Method Iteration*

Newton's method for finding a root of $f(x) = 0$ is given by:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

## *Proof of Convergence*

1. **Taylor Expansion Near $x^*$:** Since $f$ has continuous second derivatives, we can expand $f$ around $x^*$:

$$f(x_n) = f(x^*) + f'(x^*)(x_n - x^*) + \frac{1}{2}f''(\xi_n)(x_n - x^*)^2$$

where $\xi_n$ lies between $x_n$ and $x^*$. Given that $f(x^*) = 0$:

$$f(x_n) = f'(x^*)(x_n - x^*) + \frac{1}{2}f''(\xi_n)(x_n - x^*)^2$$

2. **Error Term Definition:** Let $e_n = x_n - x^*$ be the error at iteration $n$.

3. **Substituting into Newton's Method:** Using the Taylor expansion, the iteration becomes:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \approx x_n - \frac{f'(x^*)e_n + \frac{1}{2}f''(\xi_n)e_n^2}{f'(x^*)}$$

Simplifying further, we get:

$$x_{n+1} = x_n - e_n - \frac{f''(\xi_n)}{2f'(x^*)}e_n^2$$

and thus,

$$x_{n+1} = x^* - \frac{f''(\xi_n)}{2f'(x^*)}e_n^2$$

4. **Error Update:** The new error is:

$$e_{n+1} = x_{n+1} - x^* = -\frac{f''(\xi_n)}{2f'(x^*)}e_n^2$$

5. **Quadratic Convergence:** Since $f''(\xi_n)$ and $f'(x^*)$ are constants near $x^*$, there exists a constant $C$ such that:

$$|e_{n+1}| \leq C|e_n|^2$$

This indicates quadratic convergence of the method.

## *Conclusion*

Under the given assumptions, Newton's method applied to $f(x) = g'(x)$ converges quadratically to the root $x^*$ when the initial guess is sufficiently close to $x^*$.

# Second Part: Convergence of Newton's Method for Optimization

We are to show that the second-order Newton's method for finding the optimal point $x^*$ of the function $g$ is convergent.

## *Newton's Method for Optimization*

Newton's method for finding a stationary point of $g$ is:

$$x_{n+1} = x_n - \frac{g'(x_n)}{g''(x_n)}$$

## *Proof of Convergence*

1. **Assumptions:**

- $g$ has continuous third derivatives.
- $g'(x^*) = 0$ (since $x^*$ is an optimal point).
- $g''(x^*) \neq 0$ (second derivative is non-zero at $x^*$).

2. **Taylor Expansion of $g'$:** Expand $g'$ around $x^*$:

$$g'(x_n) = g'(x^*) + g''(x^*)(x_n - x^*) + \frac{1}{2}g'''(\xi_n)(x_n - x^*)^2$$

where $e_n = x_n - x^*$ and $\xi_n$ is between $x_n$ and $x^*$.

3. **Approximating $g''(x_n)$:** For $x_n$ close to $x^*$, $g''(x_n) \approx g''(x^*)$.

4. **Substituting into Newton's Method:**

$$x_{n+1} = x_n - \frac{g'(x_n)}{g''(x_n)} \approx x_n - \frac{g''(x^*)e_n + \frac{1}{2}g'''(\xi_n)e_n^2}{g''(x^*)}$$

Simplifying, we get:

$$x_{n+1} = x^* - \frac{g'''(\xi_n)}{2g''(x^*)}e_n^2$$

5. **Error Update:** The error at the next iteration is:

$$e_{n+1} = x_{n+1} - x^* = -\frac{g'''(\xi_n)}{2g''(x^*)}e_n^2$$

6. **Quadratic Convergence:** Since $g'''(\xi_n)$ and $g''(x^*)$ are constants near $x^*$, there exists a constant $C$ such that:

$$|e_{n+1}| \leq C|e_n|^2$$

This indicates quadratic convergence of the method.

# Q7

## (a) Prove that the gradient of the loss function with respect to $z$ is given by:

$$\nabla_z L = \hat{y} - y.$$

### *Solution:*

To find the gradient $\nabla_z L$, we need to compute the partial derivatives of $L$ with respect to each component of $z$.

#### *Step 1: Compute the Partial Derivative of $L$ with respect to $z_i$*

The loss function is:

$$L = -\sum_{k=1}^{K} y_k \log \hat{y}_k.$$

The softmax output $\hat{y}_k$ depends on $z$. Therefore, we use the chain rule:

$$\frac{\partial L}{\partial z_i} = -\sum_{k=1}^{K} y_k \frac{\partial}{\partial z_i} \left( \log \hat{y}_k \right).$$

#### *Step 2: Compute $\frac{\partial}{\partial z_i} \left( \log \hat{y}_k \right)$*

Recall that:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

The derivative of $\hat{y}_k$ with respect to $z_i$ is:

$$\frac{\partial \hat{y}_k}{\partial z_i} = \hat{y}_k (\delta_{ki} - \hat{y}_i),$$

where $\delta_{ki}$ is the Kronecker delta, which is 1 if $k = i$ and 0 otherwise.

Next, compute:

$$\frac{\partial}{\partial z_i} \left( \log \hat{y}_k \right) = \frac{1}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_i} = \hat{y}_k (\delta_{ki} - \hat{y}_i) = \delta_{ki} - \hat{y}_i.$$

#### *Step 3: Substitute Back into the Gradient Expression*

Substitute $\frac{\partial}{\partial z_i} \left( \log \hat{y}_k \right)$ into the gradient:

$$\frac{\partial L}{\partial z_i} = -\sum_{k=1}^{K} y_k (\delta_{ki} - \hat{y}_i).$$

#### *Breaking Down the Sum*

Let's expand the sum and simplify term by term.

**First Term:** $\sum_{k=1}^{K} y_k \delta_{ki}$ Since $\delta_{ki}$ is 1 only when $k = i$, and $y_k$ is non-zero only for the correct class (due to the one-hot encoding), we have:

$$\sum_{k=1}^{K} y_k \delta_{ki} = y_i.$$

**Second Term:** $\sum_{k=1}^{K} y_k \hat{y}_i$ Here, $\hat{y}_i$ does not depend on $k$, so it can be pulled out of the sum:

$$\sum_{k=1}^{K} y_k \hat{y}_i = \hat{y}_i \sum_{k=1}^{K} y_k = \hat{y}_i.$$

**Putting It All Together** Now, rewrite the gradient expression with these simplified terms:

$$\frac{\partial L}{\partial z_i} = -(y_i - \hat{y}_i).$$

Simplify the negative sign:

$$\frac{\partial L}{\partial z_i} = -y_i + \hat{y}_i.$$

This can be rearranged as:

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i.$$

**Expressing the Gradient as a Vector** Since this holds for each $i = 1, 2, \ldots, K$, we can write the gradient vector as:

$$\nabla_z L = \hat{y} - y.$$

# (b-1) Calculating the Hessian Matrix H

**Objective:** Compute the second-order partial derivatives of the loss function $L(z, y)$ with respect to each pair of components $z_i$ and $z_j$, forming the Hessian matrix $\mathbf{H}$.

## *Step 1: Recall the Gradient of the Loss Function*

From part (a), we have determined that the gradient of the loss function with respect to $z$ is:

$$\nabla_z L = \hat{y} - y,$$

where:

- $\hat{y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_K]^\top$ is the vector of predicted probabilities from the softmax function.
- $y = [y_1, y_2, \ldots, y_K]^\top$ is the one-hot vector of true labels.

This means that for each $i$:

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i.$$

## *Step 2: Compute the Second-Order Partial Derivatives*

The Hessian matrix $\mathbf{H}$ consists of elements $H_{ij}$, where:

$$H_{ij} = \frac{\partial^2 L}{\partial z_i \partial z_j}.$$

Since $y_i$ is constant with respect to $z$, we only need to consider the derivatives of $\hat{y}_i$:

$$H_{ij} = \frac{\partial}{\partial z_j}\left(\frac{\partial L}{\partial z_i}\right) = \frac{\partial}{\partial z_j}(\hat{y}_i - y_i) = \frac{\partial \hat{y}_i}{\partial z_j}.$$

Our task is now to compute $\frac{\partial \hat{y}_i}{\partial z_j}$ for all $i$ and $j$.

### *Step 3: Compute* $\frac{\partial \hat{y}_i}{\partial z_j}$

Recall the softmax function:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}.$$

Let $S = \sum_{k=1}^{K} e^{z_k}$.

**Compute the derivative:** Using the quotient rule:

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_i}}{S}\right) = \frac{e^{z_i}\delta_{ij}S - e^{z_i}\frac{\partial S}{\partial z_j}}{S^2}.$$

But

$$\frac{\partial S}{\partial z_j} = e^{z_j},$$

since $S = \sum_{k=1}^{K} e^{z_k}$.

**Simplify:**

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{e^{z_i}\delta_{ij}S - e^{z_i}e^{z_j}}{S^2}.$$

Dividing the numerator and denominator by $S$, we get:

$$\frac{\partial \hat{y}_i}{\partial z_j} = \hat{y}_i(\delta_{ij} - \hat{y}_j).$$

Here, $\delta_{ij}$ is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}.$$

**Interpretation:**

- When $i = j$: $\frac{\partial \hat{y}_i}{\partial z_i} = \hat{y}_i(1 - \hat{y}_i)$.
- When $i \neq j$: $\frac{\partial \hat{y}_i}{\partial z_j} = -\hat{y}_i\hat{y}_j$.

### *Step 4: Assemble the Hessian Matrix* H

The Hessian matrix **H** has elements:

$$H_{ij} = \hat{y}_i(\delta_{ij} - \hat{y}_j).$$

This is a $K \times K$ matrix where each element $H_{ij}$ depends on the predicted probabilities $\hat{y}_i$ and $\hat{y}_j$.

## (b-2) Prove that H is Positive Semi-Definite

**Objective:** Show that for any vector $v \in \mathbb{R}^K$:

$$v^\top \mathbf{H} v \geq 0.$$

This would confirm that **H** is positive semi-definite (PSD).

## *Step 1: Write $v^\top \mathbf{H} v$ in Terms of $\hat{y}_i$ and $v_i$*

Compute the quadratic form:

$$v^\top \mathbf{H} v = \sum_{i=1}^{K} \sum_{j=1}^{K} v_i H_{ij} v_j = \sum_{i=1}^{K} \sum_{j=1}^{K} v_i \hat{y}_i (\delta_{ij} - \hat{y}_j) v_j.$$

**Split the sum into two parts:** 1. Diagonal terms ($i = j$):

$$\sum_{i=1}^{K} v_i \hat{y}_i (1 - \hat{y}_i) v_i = \sum_{i=1}^{K} \hat{y}_i (1 - \hat{y}_i) v_i^2.$$

2. Off-diagonal terms ($i \neq j$):

$$\sum_{i=1}^{K} \sum_{j \neq i} v_i \hat{y}_i (-\hat{y}_j) v_j = -\sum_{i=1}^{K} \sum_{j \neq i} \hat{y}_i \hat{y}_j v_i v_j.$$

So, the quadratic form becomes:

$$v^\top \mathbf{H} v = \sum_{i=1}^{K} \hat{y}_i (1 - \hat{y}_i) v_i^2 - \sum_{i=1}^{K} \sum_{j \neq i} \hat{y}_i \hat{y}_j v_i v_j.$$

## *Step 2: Simplify the Off-Diagonal Sum*

Notice that the double sum over $i$ and $j$ with $i \neq j$ can be combined. Since the terms $\hat{y}_i \hat{y}_j v_i v_j$ are symmetric in $i$ and $j$, we can write:

$$\sum_{i=1}^{K} \sum_{j \neq i} \hat{y}_i \hat{y}_j v_i v_j = \sum_{i=1}^{K} \sum_{j=1}^{K} \hat{y}_i \hat{y}_j v_i v_j - \sum_{i=1}^{K} \hat{y}_i^2 v_i^2.$$

**Explanation:**

- The full double sum over $i$ and $j$ includes all pairs $(i, j)$.
- Subtracting the terms where $i = j$ (i.e., $\sum_{i=1}^{K} \hat{y}_i^2 v_i^2$) leaves only the off-diagonal terms.

## *Step 3: Rewrite the Quadratic Form*

Now, the quadratic form becomes:

$$v^\top \mathbf{H} v = \sum_{i=1}^{K} \hat{y}_i (1 - \hat{y}_i) v_i^2 - \left( \sum_{i=1}^{K} \sum_{j=1}^{K} \hat{y}_i \hat{y}_j v_i v_j - \sum_{i=1}^{K} \hat{y}_i^2 v_i^2 \right).$$

Simplify:

$$v^\top \mathbf{H} v = \left( \sum_{i=1}^{K} \hat{y}_i v_i^2 - \sum_{i=1}^{K} \hat{y}_i^2 v_i^2 \right) - \left( \sum_{i=1}^{K} \sum_{j=1}^{K} \hat{y}_i \hat{y}_j v_i v_j - \sum_{i=1}^{K} \hat{y}_i^2 v_i^2 \right).$$

Simplify the expression:

$$v^\top \mathbf{H} v = \sum_{i=1}^{K} \hat{y}_i v_i^2 - \left( \sum_{i=1}^{K} \sum_{j=1}^{K} \hat{y}_i \hat{y}_j v_i v_j \right).$$

### *Step 4: Recognize the Second Term as a Square*

Note that:

$$\left(\sum_{i=1}^{K} \hat{y}_i v_i\right)^2 = \sum_{i=1}^{K}\sum_{j=1}^{K} \hat{y}_i \hat{y}_j v_i v_j.$$

Therefore, the quadratic form simplifies to:

$$v^\top \mathbf{H} v = \sum_{i=1}^{K} \hat{y}_i v_i^2 - \left(\sum_{i=1}^{K} \hat{y}_i v_i\right)^2.$$

### *Step 5: Interpret the Expression as a Variance*

Consider $v$ as a random variable that takes value $v_i$ with probability $\hat{y}_i$. Then:

- The expected value (mean) of $v$ is:

$$E[v] = \sum_{i=1}^{K} \hat{y}_i v_i.$$

- The expected value of $v^2$ is:

$$E[v^2] = \sum_{i=1}^{K} \hat{y}_i v_i^2.$$

- The variance of $v$ is:

$$\mathrm{Var}(v) = E[v^2] - (E[v])^2.$$

Therefore, the quadratic form represents the variance of $v$:

$$v^\top \mathbf{H} v = \mathrm{Var}(v).$$

Since variance is always non-negative, we have:

$$v^\top \mathbf{H} v \geq 0.$$

### *Step 6: Conclusion*

Because $v^\top \mathbf{H} v \geq 0$ for all $v \in \mathbb{R}^K$, the Hessian matrix $\mathbf{H}$ is positive semi-definite.

# End of homework 2