Task 1

```
import numpy as np


#Defining ReLu, its derivative, squared loss function and initialising parameters.
def ReLU(x):
  return np.maximum(0,x)

def ReLU_der(x):
  return np.where(x)

def squared_loss(y_true,y_pred):
  return 0.5*np.mean((y_true - y_pred)**2)

def initialize_parameters(input_size,hidden_size,output_size):
  np.random.seed(1)
  w1 =np.random.randn(input_size,hidden_size)*0.01
  b1=np.zeros((1,hidden_size))
  w2=np.random.randn(hidden_size,output_size)*0.01
  b2=np.zeros((1,output_size))
  return w1,b1,w2,b2

def forward_pass(x,w1,w2,b1,b2):
  z1 = np.dot(x,w1) + b1
  a1=ReLU(z1)
  z2=np.dot(a1,w2) + b2
  return z2,a1



# Sample input data , true output (for testing) and initialising parameters
X_sample =np.array([[0.5,0.3,0.2]])
y_true=np.array([1])

input_size = 3
hidden_size = 3
output_size =1

w1,b1,w2,b2 = initialize_parameters(input_size,hidden_size,output_size)

#Forward pass
y_pred,hidden_activation = forward_pass(X_sample,w1,w2,b1,b2)

# Computing loss function
loss = squared_loss(y_true,y_pred)

# Output results
print(f"Predicted output : {y_pred}")
print(f"Squared loss : {loss}")
```

```
Predicted output : [[-2.09282703e-05]]
Squared loss : 0.5000209284893086
```

Task 2

```
# Softmax activation function
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))  # Stability improvement
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)

# Cross entropy loss function
def cross_entropy_loss(y_true, y_pred):
    # Ensure y_true is in one-hot encoded format
    y_true = np.array(y_true)
```

```
    return -np.mean(np.sum(y_true * np.log(y_pred + 1e-9), axis=1))  # Add epsilon for numerical stability

# Initialize weights and biases randomly
def initialize_parameters(input_size, hidden_size1, hidden_size2, output_size):
    np.random.seed(1)  # For reproducibility
    W1 = np.random.randn(input_size, hidden_size1) * 0.01
    b1 = np.zeros((1, hidden_size1))
    W2 = np.random.randn(hidden_size1, hidden_size2) * 0.01
    b2 = np.zeros((1, hidden_size2))
    W3 = np.random.randn(hidden_size2, output_size) * 0.01
    b3 = np.zeros((1, output_size))
    return W1, b1, W2, b2, W3, b3

# Forward pass function
def forward_pass(X, W1, b1, W2, b2, W3, b3):
    Z1 = np.dot(X, W1) + b1
    A1 = ReLU(Z1)

    Z2 = np.dot(A1, W2) + b2
    A2 = ReLU(Z2)

    Z3 = np.dot(A2, W3) + b3
    A3 = softmax(Z3)

    return A3

# Sample input data and true output (one-hot encoded)
X_sample = np.array([[0.5, 0.2]])
y_true = np.array([[0, 1]])  # Assuming 2 classes, and the true label is the second class

# Initialize parameters
input_size = 2
hidden_size1 = 3
hidden_size2 = 3
output_size = 2

W1, b1, W2, b2, W3, b3 = initialize_parameters(input_size, hidden_size1, hidden_size2, output_size)

# Perform forward pass
y_pred = forward_pass(X_sample, W1, b1, W2, b2, W3, b3)

# Compute loss
loss = cross_entropy_loss(y_true, y_pred)

# Output results
print(f"Predicted Probabilities: {y_pred}")
print(f"Cross Entropy Loss: {loss}")
```

```
Predicted Probabilities: [[0.49999984 0.50000016]]
Cross Entropy Loss: 0.6931468555186525
```