Name: Subrato Tapaswi	**Class/Roll No.: D16AD/60**	**Grade:**

Title of Experiment: Design and implement a fully connected deep neural network with at least 2 hidden layers for a classification application. Use appropriate Learning Algorithm, output function and loss function.

Objective of Experiment: Design a deep neural network with two or more hidden layers for classification and involves selecting appropriate activation functions, a suitable learning algorithm, and loss function to showcase effective model creation.

Description / Theory:
The main aim of optimization algorithms is to reduce the loss function, which measures how wrong the network's predictions are compared to the actual targets. The choice of which algorithm to use depends on your specific problem, the network structure, and the data you have. Experimentation and tuning are often required to find the best optimization strategy for your deep learning model. Here are some common learning algorithms used in deep learning:

1. Gradient Descent: Gradient Descent is the foundation of most optimization algorithms in deep learning. It involves calculating the gradients of the loss function with respect to the model's parameters (weights and biases) and adjusting the parameters in the opposite direction of the gradients to minimize the loss.

   a. **Stochastic Gradient Descent**: Updates the parameters using a small randomly selected subset (mini-batch) of the training data at each iteration. This helps speed up convergence and is less computationally intensive than using the entire dataset.

b. **Batch Gradient Descent:** Updates the parameters using the entire training dataset at each iteration. Can be slow for large datasets but provides a more accurate estimate of the gradients.

c. **Mini-Batch Gradient Descent:** A compromise between SGD and Batch GD, where the parameters are updated using a moderate-sized mini-batch of the training data.

2. Momentum: Momentum is a technique that accelerates SGD by accumulating a velocity term that keeps the model moving in the right direction, even when the gradients change direction frequently.

3. Adaptive Learning Rate Methods:

   a. **Adagrad**: Adjusts the learning rate for each parameter based on the historical gradient information, allowing the learning rate to adapt to different parameters.

   b. **Adam:** Combines ideas from Momentum and RMSProp, adapting both the learning rate and momentum for each parameter.

4. Nesterov Accelerated Gradient (NAG)**:** An enhancement to Momentum that computes the gradient not at the current parameter values but slightly ahead using the momentum.

Activation Function -

The activation functions used in the output layer of a neural network are essential in determining the kind of predictions the network makes. Your choice of the output function depends on the specific task you're tackling, whether it's classification, regression, or something else. Here are some frequently used output functions in deep learning:

1. **Sigmoid Activation (Logistic):**

➔ Range: [0, 1]

➔ Commonly used in binary classification problems where you want to predict probabilities.

➔ Example: Predicting whether an email is spam (1) or not (0).

2. **Softmax Activation:**

➔ Used for multi-class classification problems.

➔ Converts the raw scores (logits) into a probability distribution over multiple class.

➔ Ensures that the sum of probabilities across all classes is 1.

➔ Example: Image classification into different categories (e.g., classifying handwritten digits).

3. **Linear Activation:**

➔ Used in regression problems where you want to predict continuous values.

➔ The output is the weighted sum of inputs, and no nonlinearity is applied.

➔ Example: Predicting the price of a house based on its features.

4. **Hyperbolic Tangent (Tanh) Activation:**

➔ Range: [-1, 1]

➔ Similar to the sigmoid function but with outputs symmetrically centered around 0.

➔ Used in cases where inputs can be negative as well as positive.

➔ Example: Sentiment analysis where the sentiment can be positive, neutral, or negative.

5. **Rectified Linear Unit (ReLU) Activation:**

➔ Range: [0, $+\infty$]

➔ Widely used in hidden layers to introduce nonlinearity and prevent the vanishing gradient problem.

➔ Not suitable for the output layer of classification tasks due to the unbounded positive range.

➔ Example: Used as an activation function in intermediate layers of deep neural networks.

6. **Leaky ReLU and Parametric ReLU (PReLU):**

➔ Variants of ReLU that allow a small slope for negative inputs, helping to mitigate the "dying ReLU" problem.

➔ PReLU introduces a learnable parameter for the negative slope.

Loss Function -

Loss function (also known as a cost function or objective function) is a crucial component that quantifies how well the predictions made by a neural network align with the actual target values in the training data. Here are some commonly used loss functions in deep learning:

1. *Mean Squared Error (MSE):*

➔ Used for regression tasks.

➔ Measures the average squared difference between predicted and actual continuous values.

➔ Formula: `MSE = (1 / n) * Σ(y_true - y_pred)^2`, where `n` is the number of samples.

2. *Binary Cross-Entropy Loss (BCE Loss):*

➔ Used for binary classification tasks.

➔ Measures the difference between predicted probabilities and actual binary labels.

➔ Formula: `BCE Loss = -[y * log(p) + (1 - y) * log(1 - p)]`, where `y` is the true label and `p` is the predicted probability of the positive class.

3. *Categorical Cross-Entropy Loss:*

➔ Used for multi-class classification tasks.

➔ Measures the difference between predicted class probabilities and actual one-hot encoded labels.

➔ Formula: `Categorical CE Loss = -∑(y * log(p))`, where `y` is the true one-hot encoded vector of class labels and `p` is the predicted vector of class probabilities.

Program & Output:

1. Programs on Basic programming constructs like branching and looping.

```
[1] import numpy as np
    import tensorflow as tf
    from sklearn.datasets import load_breast_cancer
    from sklearn.model_selection import train_test_split
    from tensorflow import keras
    from tensorflow.keras import layers

[2] # Load the Breast Cancer dataset
    data = load_breast_cancer()
    X = data.data
    y = data.target

    # Convert labels to one-hot encoding
    y_one_hot = tf.keras.utils.to_categorical(y, num_classes=2)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2,
    random_state=42)

    # Build the neural network model
    model = keras.Sequential([
     layers.Input(shape=(30,)),
     layers.Dense(128, activation='relu'),
     layers.Dense(64, activation='relu'),
     layers.Dense(2, activation='softmax')
    ])
```

Early Stopping:

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Implement early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10,restore_best_weights=True)

# Train the model with early stopping
batch_size = 32
epochs = 100
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.2, callbacks=[early_stopping])
```

Output Screenshots:

```
[5] # Evaluate the model on the test data
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")

    4/4 [==============================] - 0s 4ms/step - loss: 0.5013 - accuracy: 0.9123
    Test Loss: 0.5013, Test Accuracy: 0.9123
```

After Optimization:

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")

4/4 [==============================] - 0s 4ms/step - loss: 0.8682 - accuracy: 0.9386
Test Loss: 0.8682, Test Accuracy: 0.9386
```

Results and Discussions: The experiment achieved success by using a deep neural network with two hidden layers for classification. It applied the Adam optimization algorithm, used softmax activation for the output layer, ReLU activation for the hidden layers, and utilized the cross-entropy loss function. These selections contributed to efficient model training and resulted in precise classification outcomes.