Neural Network (NN)

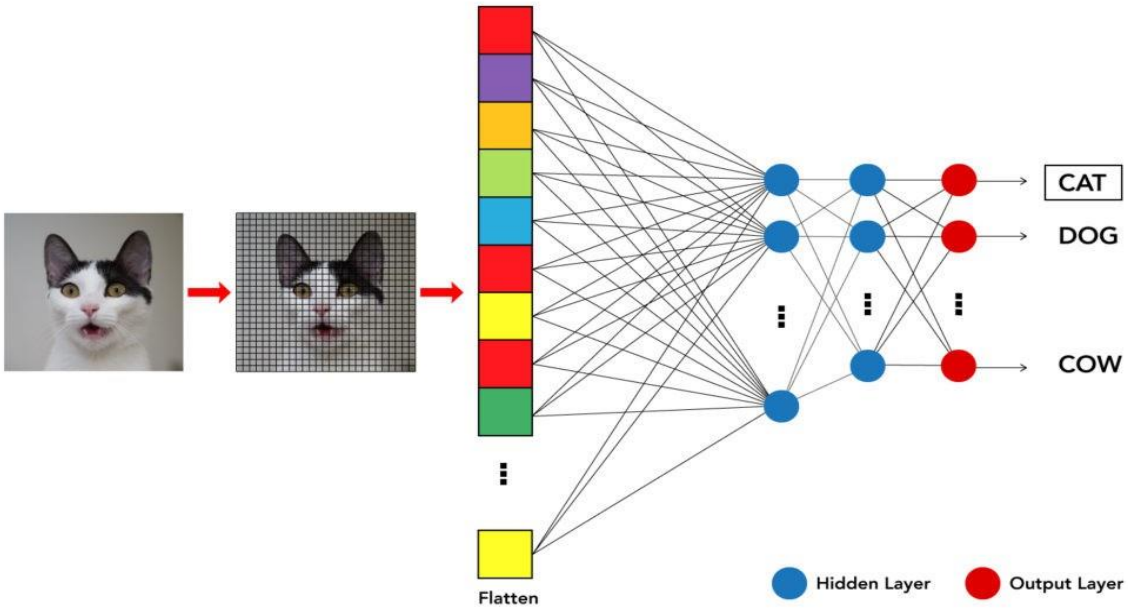
Lab plan 08-09

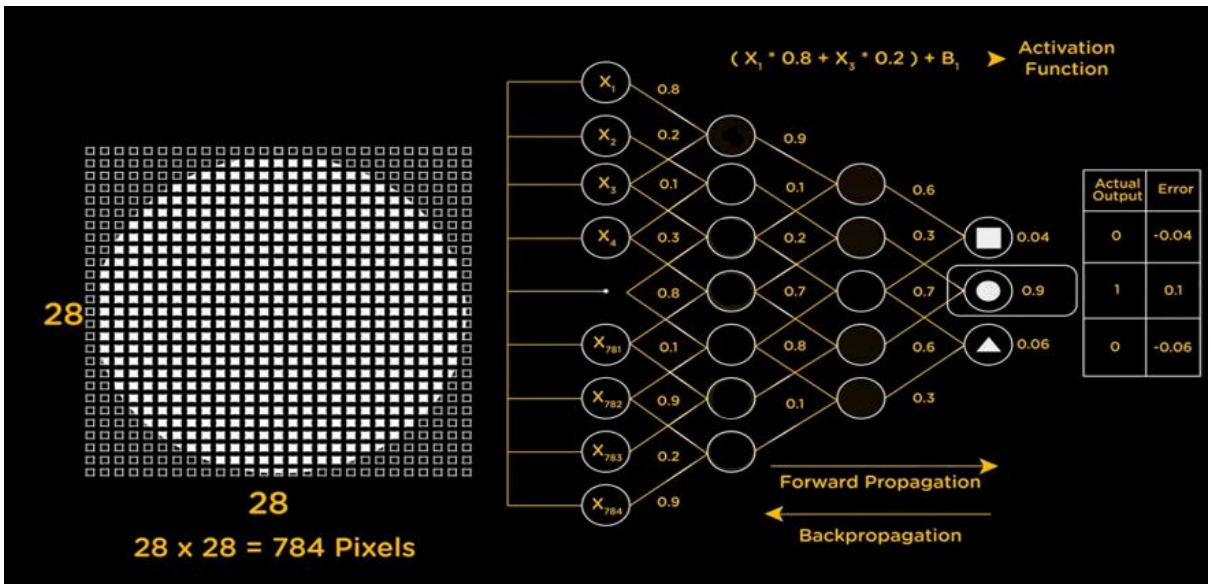
Input Image A picture is given to the model.

Pixel Representation The image is converted into a grid of pixels

Hidden Layers **N**etwork processes these values through multiple layers

Output Layer The model predicts the class based on learned features.





Types of Activation Functions

1. **Linear Activation Function**:
   - The output is directly proportional to the input (Identity function).
   - Not commonly used as it does not introduce non-linearity.
   - Example: Output = Input (e.g., for house price prediction).
2. **Non-Linear Activation Functions**
   - Can learn any type of data
   - Examples: Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax

Activation Function	Formula	Output Range	Use / Application
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$	Converts values into probabilities → widely used in **binary classification** (final layer).
ReLU (Rectified Linear Unit)	$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	$[0, \infty)$	Most popular in **hidden layers** of deep networks because it's simple and reduces vanishing gradient.
Tanh (Hyperbolic Tangent)	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1, 1)$	Better than sigmoid in some cases since it's **zero-centered** → used in RNNs and some hidden layers.
Softmax	For $\mathbf{x} = [x_1, \ldots, x_n]$: $\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$	$(0, 1)$, with $\sum_i \text{Softmax}(x_i) = 1$	Used in **multi-class classification** (final layer). Converts raw scores into class probabilities.
Linear	$f(x) = x$	$(-\infty, +\infty)$	Commonly used in the **output layer for regression tasks** where the target is continuous.
Leaky ReLU	$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \; 0 < \alpha \ll 1$	$(-\infty, \infty)$	Fixes "dying ReLU" problem → allows small gradient for negative values. Used in hidden layers.

```
import numpy as np  # Import NumPy for numerical operations
import tensorflow as tf  # Import TensorFlow for building neural networks
from tensorflow import keras  # Import Keras from TensorFlow for a high-level neural network API
from tensorflow.keras import layers  # Import layers module from Keras
from sklearn.datasets import load_iris  # Import function to load the Iris dataset
from sklearn.model_selection import train_test_split  # Import function to split datasets
from sklearn.preprocessing import StandardScaler  # Import scaler for data standardization

# Function to run multi-class classification with Softmax activation in the output layer
def run_multi_class_classification(hidden_activation):
    # Load the Iris dataset
    iris = load_iris()
    X, y = iris.data, iris.target  # Separate features (X) and labels (y)

    # Convert labels to categorical format (one-hot encoding)
    y = keras.utils.to_categorical(y, num_classes=3)
```

```
    # Standardize the feature data
    scaler = StandardScaler()
    X = scaler.fit_transform(X)  # Fit the scaler and transform the data

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

    # Build the neural network model
    model = keras.Sequential([
        layers.Input(shape=(X_train.shape[1],)),  # Input layer with
shape equal to number of features
        layers.Dense(64, activation=hidden_activation),  # Hidden layer
with specified activation function
        layers.Dense(3, activation='softmax')  # Output layer with
Softmax activation for multi-class classification
    ])

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

    # Train the model on the training data
    model.fit(X_train, y_train, epochs=50, verbose=0)  # Suppress verbose
output during training

    # Evaluate the model on the test data and get accuracy
    accuracy = model.evaluate(X_test, y_test, verbose=0)[1]  # Extract
accuracy from the evaluation result
    return accuracy  # Return the accuracy

# List of hidden layer activation functions to apply
activation_functions = {
    'Tanh': 'tanh',  # Hyperbolic tangent activation function
    'Sigmoid': 'sigmoid',  # Sigmoid activation function
    'ReLU': 'relu',  # Rectified Linear Unit activation function
    'Leaky ReLU': layers.LeakyReLU(alpha=0.2),  # Leaky ReLU with a small
slope for negative values
    'Linear': 'linear'  # Linear activation function
}

# Run the classification for each hidden layer activation function and
print the accuracy
print("=== Multi-Class Classification with Different Hidden Layer
Activations on Iris Dataset ===")
for name, activation in activation_functions.items():
    accuracy = run_multi_class_classification(activation)  # Run
classification with the current activation
    print(f'Accuracy with {name} activation in hidden layer (Softmax in
output): {accuracy:.4f}')  # Print the accuracy
```

```
Accuracy with Tanh activation in hidden layer (Softmax in output): 0.9667
Accuracy with Sigmoid activation in hidden layer (Softmax in output): 0.9333
Accuracy with ReLU activation in hidden layer (Softmax in output): 0.9667
Accuracy with Leaky ReLU activation in hidden layer (Softmax in output): 0.9667
Accuracy with Linear activation in hidden layer (Softmax in output): 0.9667
```

Parameters Tunning

Parameters

Parameters are internal variables learned from the training data, such as weights and biases in neural networks. They are updated during training through optimization.

Hyperparameters

Hyperparameters, on the other hand, are external configurations set before training, like learning rate, number of epochs, and batch size. Hyperparameters are adjusted using techniques like grid search or random search

1. Grid Search

Grid search is a brute-force method that involves defining a grid of hyperparameter values and evaluating all possible combinations.

2. Random Search

Description: Instead of searching every combination, random search randomly samples a fixed number of hyperparameter combinations from the defined space.

Example: Using the same hyperparameters as in grid search, random search might randomly choose:

- (0.1, 100)
- (0.01, 50)
- (1.0, 10)

3. Hyperband

Its like a **fast filter system**. You test **many hyperparameter settings quickly** (only a few training steps/epochs). You **eliminate poor ones** early and keep only the promising ones.The survivors are trained longer to see which performs best. Efficient because it **saves time and resources**.

Test a lot quickly, drop the losers, focus on the winners.

Small, simple problem → Grid Search

Large space, limited budget → Random Search

Expensive training, want speed → Hyperband

Hyperparameter Optimization Techniques for Neural Networks(Classification Problem): A Comparative Study Using Grid Search, Random Search, Hyperband.

1. Classification problem

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense
import random
import itertools

!pip install optuna keras-tuner pyswarms

import optuna
import keras_tuner as kt
import pyswarms as ps

# ------------------------------
# Load and preprocess Wine dataset
# ------------------------------
wine = load_wine()
X = wine.data
y = wine.target

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# ------------------------------
# Base model builder
# ------------------------------
def build_model(optimizer='adam', activation='relu', neurons=32):
    model = Sequential()
    model.add(Dense(neurons, input_dim=X.shape[1],
activation=activation))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
optimizer=optimizer, metrics=['accuracy'])
    return model

optimizers = ['adam', 'sgd', 'rmsprop']
activations = ['relu', 'sigmoid']
neurons_list = [16, 32, 64]
batch_sizes = [16, 32, 64]
epochs_list = [10, 20, 30]

# ------------------------------
# Random Search
# ------------------------------
def random_search(n_iter=5):
    best_acc, best_params = 0, None
    for _ in range(n_iter):
        params = {
            'optimizer': random.choice(optimizers),
            'activation': random.choice(activations),
            'neurons': random.choice(neurons_list),
            'batch_size': random.choice(batch_sizes),
            'epochs': random.choice(epochs_list)
        }
        model = build_model(params['optimizer'],
params['activation'], params['neurons'])
        model.fit(X_train, y_train, batch_size=params['batch_size'],
epochs=params['epochs'], verbose=0)
        y_pred = np.argmax(model.predict(X_test), axis=1)
        acc = accuracy_score(y_test, y_pred)
        if acc > best_acc:
            best_acc, best_params = acc, params
    return best_params

# ------------------------------
# Grid Search
# ------------------------------
def grid_search():
    best_acc, best_params = 0, None
    for combo in itertools.product(optimizers, activations,
neurons_list, batch_sizes, epochs_list):
        params = {
            'optimizer': combo[0],
            'activation': combo[1],
            'neurons': combo[2],
            'batch_size': combo[3],
            'epochs': combo[4]
```

```
        }
        model = build_model(params['optimizer'],
params['activation'], params['neurons'])
        model.fit(X_train, y_train, batch_size=params['batch_size'],
epochs=params['epochs'], verbose=0)
        y_pred = np.argmax(model.predict(X_test), axis=1)
        acc = accuracy_score(y_test, y_pred)
        if acc > best_acc:
            best_acc, best_params = acc, params
    return best_params

# ------------------------------
# Hyperband (Keras Tuner)
# ------------------------------
def hyperband_model(hp):
    model = Sequential()
    model.add(Dense(hp.Choice('neurons', neurons_list),
                    activation=hp.Choice('activation', activations),
                    input_shape=(X.shape[1],)))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer=hp.Choice('optimizer', optimizers),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = kt.Hyperband(hyperband_model,
                     objective='val_accuracy',
                     max_epochs=30,
                     directory='hyperband_dir',
                     project_name='wine')
tuner.search(X_train, y_train, validation_split=0.2, verbose=0)
best_hyperband = tuner.get_best_hyperparameters(1)[0].values

# ------------------------------
# Run and Print Results
# ------------------------------
best_random = random_search()
best_grid = grid_search()
best_hyperband= hyperband_model()
best_bayes = study.best_params

print("Best Random Search Params:", best_random)
print("Best Grid Search Params:", best_grid)
print("Best Hyperband Params:", best_hyperband)
```

Grid Search

Best Parameters:

```
{'batch_size': 5, 'epochs': 20, 'model__activation': 'tanh', 'model__neurons': 30,
'model__optimizer': 'sgd'}
```

Random Search

Best Parameters:

```
{'model__optimizer': 'sgd', 'model__neurons': 10, 'model__activation': 'tanh', 'epochs': 20,
'batch_size': 5}
```

Hyperband

Best Parameters:

```
{'neurons': 16, 'activation': 'relu', 'optimizer': 'rmsprop', 'tuner/epochs': 2,
```