B2: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

In [8]:
```
# import libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

In [9]:
```
# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

In [10]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [12]:
```
# Create and train the MLP model
# The 'adam' solver uses stochastic gradient descent + backpropagation to update weights.
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, activation='relu', solver='adam', random_state
=42)
```

Parameter	Meaning
`hidden_layer_sizes=(10,)`	One hidden layer with **10 neurons**. You can have multiple layers like `(10, 5)` for two layers: 10 neurons in first, 5 in second.
`max_iter=1000`	Train the model for a **maximum of 1000 iterations** (epochs) if convergence isn't reached earlier.
`activation='relu'`	Use the **ReLU (Rectified Linear Unit)** activation function: $f(x) = \max(0, x)$. ReLU helps avoid vanishing gradients.
`solver='adam'`	Optimization algorithm used: **Adam** optimizer (Adaptive Moment Estimation). It combines RMSProp and SGD with momentum.
`random_state=42`	Ensures reproducibility by setting a **fixed seed** for random number generation.

In [13]:
```
mlp.fit(X_train, y_train)
```

```
c:\Users\Venkat\AppData\Local\Continuum\anaconda3\envs\myenv_upgrade\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:780: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[13]:

▾ MLPClassifier (https://scikit-learn.org/1.7/modules/generated/

Parameters

In [14]:
```
#Make predictions
y_pred = mlp.predict(X_test)
```

In [15]:
```
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 97.78%
```