Deep Learning is a subset of Machine Learning that uses Artificial Neural Networks (ANNs) with multiple layers (hence "deep") to model complex patterns in data.

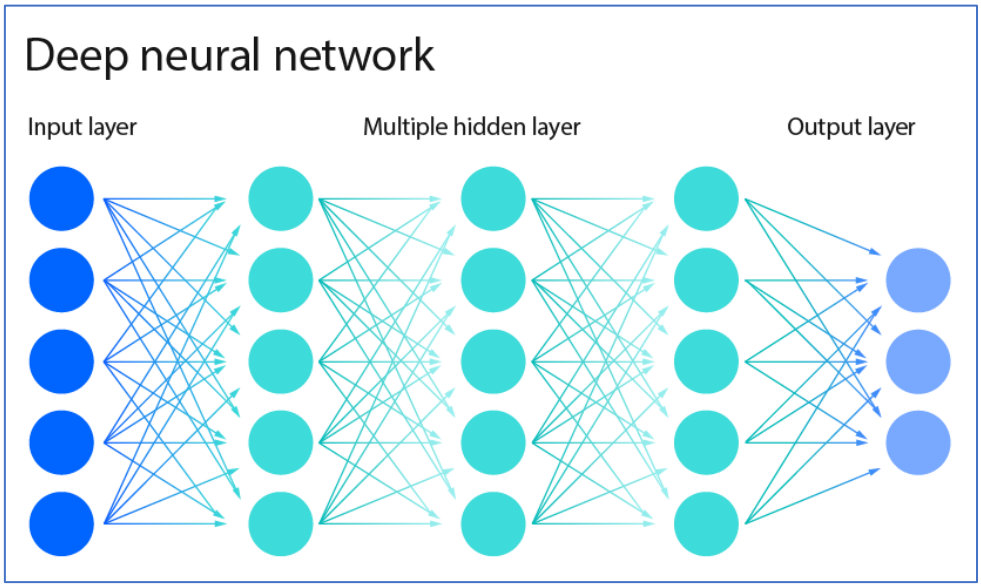
#Why Deep Learning?

- Handles large-scale, high-dimensional data (images, text, audio).
- Automatically learns hierarchical feature representations from raw data → Reduces need for manual feature engineering.
- Outperforms traditional ML in tasks like image classification, speech recognition, and language modelling.

Characteristic	Explanation
Multiple Layers	Input → Hidden Layers (Multiple) → Output Layer. Each layer learns increasingly abstract features.
Non-Linearity	Uses activation functions (ReLU, Sigmoid) to model non-linear relationships.
Backpropagation	Core algorithm to adjust model weights using gradient descent based on error signal.
Large Data Requirement	Needs large labeled datasets for good performance (e.g., ImageNet, huge text corpora).
High Compute Requirement	Typically requires GPUs for efficient training.

#Typical Workflow in Deep Learning:

1. Data Collection: Images, Text, Time-Series
2. Data Preprocessing: Normalize, Tokenize, Embed
3. Model Design: Choose architecture (CNN, RNN, Transformer, etc.)
4. Training:
    - Forward pass → Compute prediction.
    - Loss calculation → Compare prediction with true labels.
    - Backward pass → Compute gradients, update weights (Gradient Descent).
5. Evaluation: Accuracy, Precision, Loss curves, Confusion Matrix.
6. Deployment: Serve model in production environment (APIs, Web apps).



1) Artificial Neural Networks (ANN):

- Structure: Input Layer → Hidden Layers → Output Layer.
- Neurons → Apply weighted sum of inputs + activation function → Output.

#Activation Functions:

Activation	Output Range	Use Case
Sigmoid	(0, 1)	Binary classification
ReLU (Rectified Linear Unit)	[0, ∞)	Hidden layers to introduce non-linearity
Softmax	[0,1] (Sum = 1)	Multi-class classification (output layer)

#Simple Example (Binary Classification):

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define Model
model = Sequential()
model.add(Dense(8, activation='relu', input_dim=10))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer

# Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Sample Data (Dummy)
import numpy as np
X = np.random.rand(100, 10)  # 100 samples, 10 features
y = np.random.randint(2, size=100)  # Binary labels

# Train Model
model.fit(X, y, epochs=5, batch_size=10)
```

2) Convolutional Neural Network (CNN):

- Used for spatial data → Images, sometimes 1D sequences (text).
- Key Components:
  - Convolution Layer → Applies filter (kernel) to detect features.
  - Pooling Layer → Reduces spatial dimension (e.g., MaxPooling).
  - Fully Connected Layer → Final prediction.

#Example (Simple CNN for Image Classification):

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))  # Convolution
model.add(MaxPooling2D(pool_size=(2,2)))  # Pooling
model.add(Flatten())  # Flatten for Dense layers
model.add(Dense(128, activation='relu'))
```

```
model.add(Dense(10, activation='softmax'))  # 10 classes (MNIST)

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

3) **Recurrent Neural Networks (RNN) & LSTM**

- Processes sequential data step-by-step.
- Maintains hidden state → Captures sequence dependency.

#Limitation of Vanilla RNN:

- Vanishing Gradient Problem → Hard to learn long-term dependencies.

#LSTM (Long Short-Term Memory):

- Gates (Input, Forget, Output) control flow of information.

Example (Simple LSTM for Sequence Classification):

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding

model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))  # Word embeddings
model.add(LSTM(64))  # Sequence processing
model.add(Dense(1, activation='sigmoid'))  # Binary classification

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

4) Autoencoders (AE)

- Unsupervised → Learn compressed representation of input.
- Structure:
    - Encoder → Latent space
    - Decoder → Reconstruct input.

#Example (Simple Autoencoder):

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

input_layer = Input(shape=(784,))
encoded = Dense(64, activation='relu')(input_layer)
decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(inputs=input_layer, outputs=decoded)
autoencoder.compile(optimizer='adam', loss='mse')

# Dummy data
X = np.random.rand(1000, 784)

autoencoder.fit(X, X, epochs=5, batch_size=32)
```

5) Optimizers & Regularization

#Common Optimizers:

Optimizer	Description
SGD	Basic gradient descent
Adam	Adaptive optimizer (recommended for most cases)
RMSProp	Good for RNNs, handles non-stationary objectives

#Regularization Techniques:

1. **Dropout**: Randomly ignores neurons during training → Reduces overfitting.

Example:
```
from tensorflow.keras.layers import Dropout
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # 50% dropout
```

2. **L1 & L2 Regularization**
   - L1 → Sparse weights
   - L2 → Penalizes large weights (weight decay)

Example:
```
from tensorflow.keras.regularizers import l2
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
```

6) Loss Functions

Loss Function	Use Case
Binary Cross-Entropy	Binary Classification
Categorical Cross-Entropy	Multi-Class Classification
Mean Squared Error (MSE)	Regression Tasks