Neural Networks

Definition

Neural networks are a class of machine learning models designed to simulate the way the human brain processes information. They consist of layers of artificial neurons that process input data, learn patterns, and make predictions. These networks are widely used in applications such as image recognition, natural language processing, and autonomous systems.

Why Neural Networks Are Better Than Other Techniques

1. **Feature Learning**: Unlike traditional machine learning models that require manual feature engineering, neural networks automatically extract important features from raw data.
2. **Non-linearity**: Activation functions enable neural networks to learn complex, non-linear relationships in data, which linear models cannot handle.
3. **Scalability**: Neural networks can process large amounts of data efficiently, making them suitable for large-scale tasks.
4. **Generalization**: When trained properly with regularization techniques, neural networks can make accurate predictions on unseen data by capturing essential patterns.

Activation Functions

What Are Activation Functions?

Activation functions are mathematical functions applied to the output of each neuron in a neural network to introduce non-linearity. This allows the network to learn complex patterns and relationships in data. Without activation functions, a neural network would behave like a simple linear model, limiting its ability to solve complex problems.

Types of Activation Functions

For Classification

- **Sigmoid Function**: Outputs values between 0 and 1, making it suitable for binary classification problems. **Python Implementation:**

```python
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

- **Softmax Function**: Used for multi-class classification, producing probabilities for each class. **Python Implementation:**

```python
def softmax(x):
    exp_x = np.exp(x - np.max(x))
    return exp_x / exp_x.sum(axis=0)
```

For Regression

- **ReLU (Rectified Linear Unit)**: Outputs max(0, x), which helps prevent vanishing gradients and improves learning efficiency. **Python Implementation:**

```python
def relu(x):
    return np.maximum(0, x)
```

- **Leaky ReLU**: Addresses the issue of zero gradients in standard ReLU by allowing small gradients for negative values. **Python Implementation:**

```python
def leaky_relu(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)
```

Network Structures

Depiction of a Neural Network

(Insert an image of a simple neural network with 3 inputs, a hidden layer with 4 neurons, and an output layer with 1 neuron.)

Components of a Neural Network

- **Input Layer**: Receives raw input data.
- **Hidden Layers**: Process inputs through weighted connections and activation functions.
- **Output Layer**: Produces final predictions based on learned representations.

- **Weights & Biases**: Parameters that adjust during training to minimize error.

Python Code Example

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(4, activation='relu', input_shape=(3,)),
    layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
```

Multilayer Neural Networks

Depiction of a Multilayer Network

(Insert an image of a neural network with 3 inputs, one hidden layer with 4 neurons, and 1 output neuron.)

Python Code for Multilayer Perceptron

```
model = keras.Sequential([
    layers.Dense(4, activation='relu', input_shape=(3,)),
    layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
model.summary()
```

Overfitting and Dropout

What is Overfitting?

Overfitting occurs when a neural network learns patterns specific to the training data, including noise, rather than generalizing well to new, unseen data. This results in high accuracy on training data but poor performance on validation data.

Solution: Dropout

Dropout is a regularization technique that randomly deactivates a percentage of neurons during training, preventing the model from relying too much on specific features.

Python Code with Dropout:

```
model = keras.Sequential([
    layers.Dense(4, activation='relu', input_shape=(3,)),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

TensorFlow Code for OCR

Optical Character Recognition (OCR) using TensorFlow involves using convolutional neural networks (CNNs) to process and recognize text in images.

Python Code for OCR with TensorFlow:

```
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(26, activation='softmax') # Assuming 26 character classes
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
```

Computer Vision

Convolution

Convolution is the process of applying a filter (kernel) to an image to extract important features like edges, shapes, and textures.

Max Pooling

Max pooling is a downsampling operation that reduces the size of feature maps while preserving key information. It helps in reducing computational complexity and preventing overfitting.

Python Code for Max Pooling:

```
layers.MaxPooling2D(pool_size=(2,2))
```

Convolutional Neural Network (CNN) Code

```
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```