Architectures in Neural Networks

Neural network architectures are frameworks that define how layers, neurons, and operations are organized to process input data and generate outputs. The two primary architectures are **Sequential** and **Functional**.

---

1. Sequential Architecture

Overview

The **Sequential** API in frameworks like TensorFlow/Keras is a linear stack of layers. This is suitable for simple feedforward models, where layers are added one after the other in a straightforward manner.

Example: Binary Classification (Sigmoid Activation for Output)

Let's build a neural network for binary classification:

- Input: 4 features (x_1, x_2, x_3, x_4)
- Hidden Layer 1: 8 neurons, ReLU activation
- Hidden Layer 2: 4 neurons, ReLU activation
- Output Layer: 1 neuron, Sigmoid activation (for binary classification)

Code: Sequential Model

python

Copy code

```
from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense # Initialize Sequential model model = Sequential() # Add layers model.add(Dense(8, input_dim=4, activation='relu')) # Hidden Layer 1
```

```
model.add(Dense(4, activation='relu')) # Hidden Layer 2 model.add(Dense(1, activation='sigmoid')) # Output Layer (binary classification) # Compile the model model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']) # Display the model architecture model.summary()
```

- **Explanation**:
  - `Dense(8, input_dim=4, activation='relu')` : First hidden layer with 8 neurons, accepting 4 input features.
  - `Dense(1, activation='sigmoid')` : Output layer with 1 neuron, using sigmoid for binary classification.

---

2. Functional Architecture

Overview

The **Functional** API allows for building more flexible and complex architectures. It is useful for models with:

- Multiple inputs/outputs
- Non-linear connections
- Shared layers

Example: **Multiclass Classification (Softmax Activation for Output)**

Let's build a neural network for multiclass classification with 3 classes:

- Input: 4 features
- Hidden Layer 1: 8 neurons, ReLU activation
- Hidden Layer 2: 6 neurons, ReLU activation

- Output Layer: 3 neurons, Softmax activation (for 3 classes)

Code: Functional Model

```python
from tensorflow.keras.models import Model from tensorflow.keras.layers import Input, Dense # Input layer inputs = Input(shape=(4,)) # 4 input features # Hidden layers x = Dense(8, activation='relu')(inputs) # Hidden Layer 1 x = Dense(6, activation='relu')(x) # Hidden Layer 2 # Output layer outputs = Dense(3, activation='softmax')(x) # Output layer for 3 classes # Define the model model = Model(inputs=inputs, outputs=outputs) # Compile the model model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']) # Display the model architecture model.summary()
```

- **Explanation**:
  - `Input(shape=(4,))` : Specifies the input with 4 features.
  - `Dense(3, activation='softmax')` : Output layer with 3 neurons (one for each class), using softmax activation for multiclass classification.

Activation Functions: Sigmoid vs Softmax

1. Sigmoid Activation: For Binary Classification

- **Formula**:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Range**: Outputs a value between 0 and 1, representing the probability of a single class.
- **Use Case**: Binary classification problems where the target variable is 0 or 1.

Example:

Let's classify whether an email is spam (1) or not spam (0).

```python
# Predicted probability using Sigmoid predicted = model.predict([[0.8, 1.2, 0.4, -0.6]]) # Output: [[0.83]] -> 83% probability of being spam
```

Copy code

---

2. Softmax Activation: For Multiclass Classification

- **Formula**:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

  Here, N is the number of classes.
- **Range**: Outputs probabilities for each class, all summing to 1.
- **Use Case**: Multiclass classification problems where the target has more than 2 classes.

Example:

Let's classify an image into 3 categories: Cat, Dog, or Rabbit.

python

Copy code

```
# Predicted probabilities using Softmax predicted = model.predict([[0.8, 1.2, 0.4, -0.6]]) # Output: [[0.1, 0.6, 0.3]] -> Probabilities for Cat, Dog, Rabbit
```

The highest probability is for Dog (60%).

Summary Table

Feature	Sequential Model	Functional Model
Structure	Linear stack of layers	Flexible, non-linear connections
Flexibility	Limited (one input, one output)	High (multiple inputs/outputs)
Complexity	Simple architectures	Complex architectures
Use Case	Feedforward networks	Shared layers, residual connections

Activation Function	Output Type	Use Case
Sigmoid	Single probability	Binary classification
Softmax	Probabilities per class	Multiclass classification