```
import numpy as np
```

Exercise 1: Define Scalar and Vectors.

- Define a scalar bias value
- Define an input vector and a weight vector
- Display their values

```
# Define a scalar bias value
bias = 0.5

# Define an input vector
input_vector = np.array([1.0, 2.0, 3.0])

# Define a weight vector
weight_vector = np.array([0.1, 0.2, 0.3])

# Display their values
print(f"Scalar Bias: {bias}")
print(f"Input Vector: {input_vector}")
print(f"Weight Vector: {weight_vector}")
```

```
Scalar Bias: 0.5
Input Vector: [1. 2. 3.]
Weight Vector: [0.1 0.2 0.3]
```

Exercise 2: Dot Product

Objective: Objective: Compute the dot product between input and weight vectors.

- Use NumPy to compute the dot product
- Print the output

```
# Compute the dot product
dot_product = np.dot(input_vector, weight_vector)

# Display the result
print(f"Dot Product: {dot_product}")
```

```
Dot Product: 1.4
```

Exercise 3: Linear Combination

Objective: Compute the linear output of a neuron.

- Add bias to the dot product
- Observe how bias shifts the decision boundary

```
result = bias + dot_product
print(f"Output after adding bias: {result}")
```

```
Output after adding bias: 1.9
```

Exercise 4: Step Activation function

Objective: Convert linear output into binary function

- Implement simple activation function
- Apply it to linear output

```
def step_activation(x):
    return 1 if x >= 0 else 0

# Apply the step activation function to the linear output
activated_output = step_activation(result)
```

```
print(f"Linear output: {result}")
print(f"Output after step activation: {activated_output}")
```

```
Linear output: 1.9
Output after step activation: 1
```

Exercise 5: Matrix as Batch of Samples

Objective: Understand batch processing.

- Define a matrix where each row is a data sample
- Print the matrix

```
matrix = np.arange(9).reshape(3, 3)
print(matrix)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Exercise 6: Bacth Dot product

Objective: Apply dot product to multiple samples simultaneously.

- Compute dot product between matrix and weight vector.
- Add bias

```
dot_product = np.dot(weight_vector, matrix)
perceptron = bias + dot_product
print(f"Dot Product: {dot_product}")
print(f"Output after adding bias: {perceptron}")
```

```
Dot Product: [2.4 3.  3.6]
Output after adding bias: [2.9 3.5 4.1]
```

Post-Lab Reflection Questions

1. What role do weights play in the dot product?

- Weights play a crucial role in determining the importance or contribution of each input feature to the overall output. They essentially scale each element of the input vector. A larger weight for a particular input feature means that feature will have a greater impact on the dot product's result, while a smaller weight will result in a lesser impact. This scaling allows the network to learn which inputs are more relevant for making predictions or classifications.

2. How does bias affect the decision boundary?

- Bias acts as an offset or a threshold. It allows the decision boundary to be shifted away from the origin, enabling the model to represent a wider range of decision boundaries. Without bias, the decision boundary would always have to pass through the origin. By adjusting the bias, the model can make classifications even when the input features are all zero or can fine-tune the classification threshold without changing the relative influence of the input features (which are governed by weights). Essentially, bias provides flexibility in the model's output by allowing the activation function to be triggered at different points.

3. Why is dot product fundamental to neural networks?

- It is the core operation for calculating the weighted sum of inputs within a neuron. Each connection in a neural network has an associated weight, and a neuron combines its inputs by multiplying each input value by its corresponding weight and then summing these products. This entire process is efficiently represented and computed using a dot product.
- Specifically, if x is the input vector and w is the weight vector, their dot product w · x (or x · w) yields a single scalar value. This scalar value represents the aggregate influence of all inputs, scaled by their respective weights, on that particular neuron. This weighted sum is then typically passed through an activation function to produce the neuron's output. Therefore, the dot product is essential for modeling how information flows and is processed within each layer of a neural network.

4. How does batch computation help in training deep learning models?

- Batch computation is fundamental to training deep learning models for several key reasons:

1. **Computational Efficiency**: Modern deep learning frameworks and hardware (like GPUs) are highly optimized for parallel processing. Processing data in batches allows for vectorized operations, which are significantly faster than processing one sample at a time. This parallelization dramatically speeds up the training process.
2. **Regularization Effect**: Using batches introduces a certain level of noise into the gradient estimates compared to using the entire dataset (Batch Gradient Descent). This noise can sometimes act as a form of regularization, preventing the model from overfitting too quickly to the training data.
3. **Memory Management** : For very large datasets, it's often impossible to load all the data into memory at once. Batching allows you to load and process data in manageable chunks, making training feasible even with limited memory resources.

```
Start coding or generate with AI.
```