Tensor Creation and Basic Operations

In [ ]:
```
import tensorflow as tf
a = tf.constant(5)
b = tf.constant(3)

# Vector
vec1 = tf.constant([1,2, 3])
vec2 = tf.constant([4, 5, 6])
# Matrix
mat1 = tf.constant([[1, 2], [2, 5]])
mat2 = tf.constant([[3, 4], [4, 6]])
print(f"Scalar Addition:{a+b}")
print(f"Vector Addition:{vec1+vec2}")
print(f"Matrix Multiplication:{tf.matmul(mat1,mat2)}")
```

```
Scalar Addition:8
Vector Addition:[5 7 9]
Matrix Multiplication:[[11 16]
 [26 38]]
```

Linear Regression Using TensorFlow

In [ ]:
```
import numpy as np

# Data
x = np.array([1, 2, 3, 4, 5], dtype=float)
y = np.array([2, 4, 6, 8, 10], dtype=float)

# Model
model = tf.keras.Sequential([
    tf.keras.Input(shape=[1]), # Using Input layer as recommended
    tf.keras.layers.Dense(units=1)
        ])
model.compile(optimizer='sgd', loss = 'mean_squared_error')
model.fit(x, y, epochs =100, verbose = 0)

# Prediction
print(f"Prediction for x=6:{model.predict(np.array([6], dtype=float))}"
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
Prediction for x=6:[[11.748373]]
```

Simple Neural Network for Classification

In [ ]:
```
from sklearn.datasets import make_classification
x, y = make_classification(n_samples = 500, n_features = 2, n_informati

# Model
model = tf.keras.Sequential([
    tf.keras.Input(shape = [2]),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])

model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics=[
model.fit(x, y, epochs=20, verbose = 0)
loss, acc = model.evaluate(x, y, verbose = 0)
print(f"Accuracy: {acc}")
```

```
Accuracy: 0.9919999837875366
```

Exercise

1. Modify Program 1 to perform division and transpose operations

In [ ]:
```
import tensorflow as tf
a = tf.constant(5)
b = tf.constant(3)

# Vector
vec1 = tf.constant([1,2, 3])
vec2 = tf.constant([4, 5, 6])
# Matrix
mat1 = tf.constant([[1, 2], [2, 5]])
mat2 = tf.constant([[3, 4], [4, 6]])
print(f"Scalar Addition:{a+b}")
print(f"Vector Addition:{vec1+vec2}")
print(f"Matrix Division:{tf.math.divide(mat1,mat2)}")
print(f"Transpose of Matrix 1: {tf.transpose(mat1)}")
```

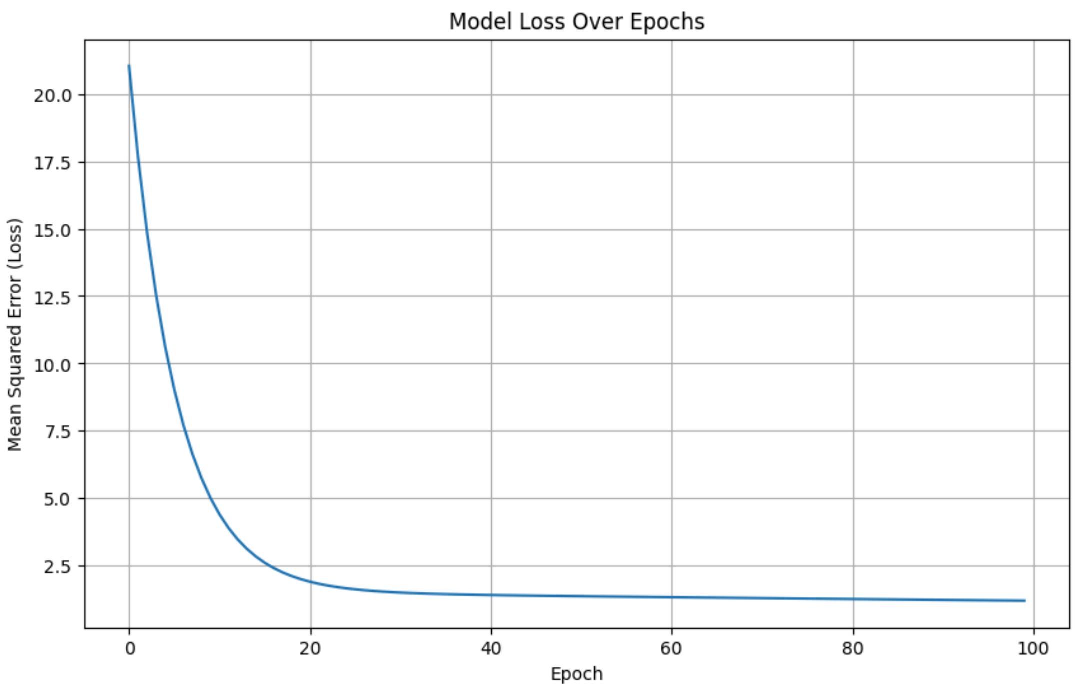
```
Scalar Addition:8
Vector Addition:[5 7 9]
Matrix Division:[[0.33333333 0.5       ]
 [0.5        0.83333333]]
Transpose of Matrix 1: [[1 2]
 [2 5]]
```

2.Change the dataset in Program 2 and observe loss variation

In [ ]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Data
x = np.array([1, 1.5, 1.8, 2.2, 2.6], dtype=float)
y = np.array([2, 4, 6, 8, 10], dtype=float)

# Model
model = tf.keras.Sequential([
    tf.keras.Input(shape=[1]), # Using Input layer as recommended
    tf.keras.layers.Dense(units=1)
        ])
model.compile(optimizer='sgd', loss = 'mean_squared_error')
history = model.fit(x, y, epochs =100, verbose = 0)

# Prediction
print(f"Prediction for x=6:{model.predict(np.array([6], dtype=float))}"

# Plotting the loss variation
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.title('Model Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error (Loss)')
plt.grid(True)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
Prediction for x=6:[[19.654558]]
```



3. Add extra hidden layer in program 3 and compare accuracy

In [ ]:

```
from sklearn.datasets import make_classification
import tensorflow as tf

# Generate a multiclass dataset
# n_informative=2, n_classes=3, n_clusters_per_class=1 ensures 3 <= 2**2
# n_informative (2) < n_features (3)
x, y = make_classification(n_samples=500, n_features=3, n_informative=2

# Model for multiclass classification with Softmax
model = tf.keras.Sequential([
    tf.keras.Input(shape=[3]), # Input layer adjusted for n_features=3
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax') # Output layer: 3 un
])

# Compile the model with sparse_categorical_crossentropy for integer lab
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',

# Train the model
model.fit(x, y, epochs=20, verbose=0)

# Evaluate the model
loss, acc = model.evaluate(x, y, verbose=0)
print(f"Accuracy: {acc}")
print(f"Loss: {loss}")
```

```
Accuracy: 0.7860000133514404
Loss: 0.6119690537452698
```

In [ ]:

```
from sklearn.datasets import make_classification
x, y = make_classification(n_samples = 500, n_features = 2, n_informativ

# Model
model = tf.keras.Sequential([
    tf.keras.Input(shape = [2]),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(4, activation='sigmoid'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])

model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics=[
model.fit(x, y, epochs=20, verbose = 0)
loss, acc = model.evaluate(x, y, verbose = 0)
print(f"Accuracy: {acc}")
print(f"Loss: {loss}")
```

```
Accuracy: 0.9739999771118164
Loss: 0.39105358719825745
```

4. Replace optimizer adam with sgd and analyze performance

In [ ]:
```
from sklearn.datasets import make_classification
x, y = make_classification(n_samples = 500, n_features = 2, n_informati

# Model
model = tf.keras.Sequential([
    tf.keras.Input(shape = [2]),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])

model.compile(optimizer='sgd', loss = 'binary_crossentropy', metrics=['
model.fit(x, y, epochs=20, verbose = 0)
loss, acc = model.evaluate(x, y, verbose = 1)
print(f"Accuracy: {acc}")
print(f"Loss: {loss}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9116 - loss: 0.39
57
Accuracy: 0.9020000100135803
Loss: 0.39485734701156616
```

5. Implement multi-class classification using Softmax activation

In [ ]:
```
from sklearn.datasets import make_classification
import tensorflow as tf
x, y = make_classification(n_samples = 500, n_features = 3, n_informati

# Model
model = tf.keras.Sequential([
    tf.keras.Input(shape = [3]),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
    ])

model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy
model.fit(x, y, epochs=20, verbose = 0)

loss, acc = model.evaluate(x, y, verbose = 0)
print(f"Accuracy: {acc}")
print(f"Loss: {loss}")
```

```
Accuracy: 0.871999979019165
Loss: 0.5519391894340515
```

Exercise Question

1. What is a tensor?

Ans:

- Tensors are multidimensional array that stores data in structured format.They are similar to mstrix with arbitary type and can store any dimension of data.

2. Difference between Numpy and TensorFlow tensors

Ans:

- NumPy arrays are handled on the CPU, while TensorFlow tensors can be processed on either CPUs, GPUs, or TPUs, which is beneficial for deep learning computations.
- TensorFlow tensors are immutable, meaning their values cannot be changed after creation. NumPy arrays, on the other hand, are mutable.
- TensorFlow provides automatic differentiation, which is crucial for training neural networks. NumPy does not have this built-in capability.
- TensorFlow tensors are part of a computational graph, enabling optimization and distributed computing. NumPy arrays are more focused on direct numerical operations.

3. What is optimizer

Ans:

- An optimizer is an algorithm or a method used to change the attributes of your neural network, such as weights and learning rate, in order to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function. In simple words, it tweaks the internal parameters of the model to achieve the best possible performance.

4. Explain Loss functions

Ans:

- In machine learning, a **loss function** (or cost function) is a method that quantifies how well a predictive model is performing. It measures the discrepancy between the predicted output of the model and the true target values.

Purpose:

- **Guidance for Optimization:** During the training process, the goal of a machine learning model is to minimize the value of the loss function. This minimization is achieved by adjusting the model's internal parameters (weights and biases) through optimization algorithms like Gradient Descent.
- **Performance Evaluation:** A lower loss value generally indicates a better-performing model, meaning its predictions are closer to the actual values.

Examples of Common Loss Functions:

1. **Mean Squared Error (MSE):**
   - **Formula:** $MSE = (1/n) * \Sigma(y_i - \hat{y}_i)^2$, where y_i is the true value, $\hat{y}_i$ is the predicted value, and n is the number of data points.
   - **Use Case:** Primarily used for **regression** tasks. It penalizes larger errors more significantly due to the squaring term, making it sensitive to outliers.
2. **Binary Cross-Entropy:**
   - **Formula:** $BCE = - (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$, where y_i is the true binary label (0 or 1), and $\hat{y}_i$ is the predicted probability (between 0 and 1).
   - **Use Case:** Used for **binary classification** problems where the output is a probability of belonging to one of two classes.
3. **Categorical Cross-Entropy (or Sparse Categorical Cross-Entropy):**
   - **Formula:** $CCE = - \Sigma(y_i * \log(\hat{y}_i))$, where y_i is a one-hot encoded vector of the true class, and $\hat{y}_i$ is the predicted probability distribution over classes.
   - **Use Case:** Used for **multi-class classification** problems. Categorical Cross-Entropy is used when target labels are one-hot encoded, while Sparse Categorical Cross-Entropy is used when target labels are integers.

5. What is backpropagation

- Ans: Backpropagation is an algorithm used in training artificial neural networks to calculate the gradient of the loss function with respect to the weights of the network. This gradient is then used by optimization algorithms (like gradient descent) to adjust the weights in a way that minimizes the loss function. In simpler terms, it's how a neural network learns from its errors by propagating the error backwards through the network layers to update the parameters.