# COSINE

## Our method / Tensor Decomposition

In [ ]:
```
np.random.seed(42)
n_samples = 10000
lower_bound = -2 * np.pi
upper_bound = 2 * np.pi
# lower_bound = -10
# upper_bound = 10

X = np.random.uniform(lower_bound, upper_bound, size=(n_samples, 1))
#X = np.arange(lower_bound, upper_bound, 0.001)
y = np.cos(X)

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, ra
def build_model(input_shape, filters):
    rank = 3
    input_layer = Input(shape=input_shape)
    x = input_layer

    h1 = H1Layer()
    h2 = H2Layer(h1)
    h3 = H3Layer(h1,h2)
    h4 = H4Layer(h2,h3)
    x = Dense(filters)(x)
    x = h2(x)
    x = Dense(filters)(x)
    x = TensorDecompositionLayer(rank)(x)
    x = h3(x)
    x = Dense(filters)(x)
    x = TensorDecompositionLayer(rank)(x)
    x = h4(x)
    x = Dense(filters)(x)
    x = TensorDecompositionLayer(rank)(x)

    output_layer = Dense(1)(x)
    model = Model(inputs=input_layer, outputs=output_layer)

    return model


input_shape = (1,)
filters = 64
modelN4 = build_model(input_shape, filters)
modelN4.summary()
optimizer = Adam(learning_rate=0.001) # Reduce learning rate
modelN4.compile(optimizer=optimizer, loss='mse')

batch_size = 64
epochs = 150

history = modelN4.fit(X_train, y_train,
```

```
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1,
                     validation_data=(X_val, y_val),
                     callbacks=[callback])

train_loss = history.history['loss']
val_loss = history.history['val_loss']

val_loss = modelN4.evaluate(X_val, y_val, verbose=0)
print(f"Validation loss: {val_loss}")

import matplotlib.pyplot as plt

num_test_samples = 1000
X_test = np.linspace(lower_bound, upper_bound, num=num_test_samples).resh
y_true = np.cos(X_test)
y_pred = modelN4.predict(X_test)

plt.figure(figsize=(10, 6))
plt.plot(X_test, y_true, label='True Cosine Values', color='b', linewidth
plt.plot(X_test, y_pred, label='Model Predictions', color='r', linestyle=
plt.xlabel('Input Value')
plt.ylabel('Cosine Value')
plt.title('Cosine Function and Model Predictions')
plt.legend()
plt.grid()
plt.show()

# Number of epochs actually trained
actual_epochs = len(history.history['loss'])

# Plot the loss graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, actual_epochs + 1), train_loss[:actual_epochs], label='
plt.plot(range(1, actual_epochs + 1), history.history['val_loss'], label=
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Time')
plt.legend()
plt.grid()
plt.show()
```

# f(x1,x2)=3cos(2pi( x1^2-x2^2)) Prediction

## Our method / CP decomposition

In [ ]:

```
# Define the function
def f(x1, x2):
    return 3 * np.cos(2 * np.pi * (x1**2 - x2**2))

# Set the parameters
lower_bound = -1
upper_bound = 1
n_samples = 1000
```

```
# Generate the data
x1_values = np.linspace(lower_bound, upper_bound, n_samples).reshape(n_sa
x2_values = np.linspace(lower_bound, upper_bound, n_samples).reshape(n_sa

# Get a meshgrid for x1 and x2 values
X1, X2 = np.meshgrid(x1_values, x2_values)

# Calculate y values using the function
y_values = f(X1, X2)

# Reshape the data for training
X = np.column_stack((X1.ravel(), X2.ravel()))
y = y_values.ravel()

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, ra

# Assuming H1Layer, H2Layer, H3Layer, H4Layer, H5Layer, and TensorDecompo
def build_model(input_shape, filters):
    rank = 4
    input_layer = Input(shape=input_shape)
    x = input_layer

    h1 = H1Layer()
    h2 = H2Layer(h1)
    h3 = H3Layer(h1,h2)
    h4 = H4Layer(h2,h3)
    h5 = H5Layer(h3,h4)

    # Using 'he_normal' initialization for the Dense layers
    x = Dense(filters, kernel_initializer='he_normal')(x)
    x = h2(x)
    x = Dense(filters, kernel_initializer='he_normal')(x)
    x = TensorDecompositionLayer(rank)(x)
    x = h3(x)
    x = Dense(filters, kernel_initializer='he_normal')(x)
    x = TensorDecompositionLayer(rank)(x)
    x = h4(x)
    x = Dense(filters, kernel_initializer='he_normal')(x)
    x = TensorDecompositionLayer(rank)(x)
    x = h5(x)
    x = Dense(filters, kernel_initializer='he_normal')(x)
    x = TensorDecompositionLayer(rank)(x)

    output_layer = Dense(1, )(x)
    model = Model(inputs=input_layer, outputs=output_layer)

    return model


input_shape = (2,)
filters = 128
modelN4 = build_model(input_shape, filters)
optimizer = Adam(learning_rate=0.0001) # Reduce learning rate
modelN4.compile(optimizer=optimizer, loss='mse')

batch_size = 128
epochs = 20

history = modelN4.fit(X_train, y_train,
```

```
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_val, y_val),
                    callbacks=[callback])

val_loss = modelN4.evaluate(X_val, y_val, verbose=0)
print(f"Validation loss: {val_loss}")

import matplotlib.pyplot as plt
modelN4.summary()

# 1. Extract loss values
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# 2. Determine the number of epochs
actual_epochs = len(train_loss)

# 3. Create a plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, actual_epochs + 1), train_loss, label='Training Loss',
plt.plot(range(1, actual_epochs + 1), val_loss, label='Validation Loss',
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Time')
plt.legend()
plt.grid()
plt.show()
```

# Function f(x1,x2) = 3*sin*(pi*x1)* cos*(pi*x2)*cos*(pi^2*x1*x2)

# THIS IS THE NEW 2 BRANCH ARCHITECTURE

In [ ]:

```
# Define the function
def f(x1, x2):
    return 3 * np.sin(np.pi*x1)*np.cos(np.pi*x2)*np.cos(np.pi**2 *x1*x2)

# Set the parameters
lower_bound = -1
upper_bound = 1
n_samples = 1000

# Generate the data
x1_values = np.linspace(lower_bound, upper_bound, n_samples).reshape(n_sa
x2_values = np.linspace(lower_bound, upper_bound, n_samples).reshape(n_sa

# Get a meshgrid for x1 and x2 values
X1, X2 = np.meshgrid(x1_values, x2_values)

# Calculate y values using the function
y_values = f(X1, X2)
```

```
# # Reshape the data for training
# X = np.column_stack((X1.ravel(), X2.ravel()))
# y = y_values.ravel()

# X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,

# Flatten the data for training
X1_flat = X1.ravel().reshape(-1, 1)  # Reshaped as a 2D array for model i
X2_flat = X2.ravel().reshape(-1, 1)  # Reshaped as a 2D array for model i
y_flat = y_values.ravel()

# Split the data into training and validation sets
X1_train, X1_val, X2_train, X2_val, y_train, y_val = train_test_split(X1_


# Assuming H1Layer, H2Layer, H3Layer, H4Layer, H5Layer, and TensorDecompo
def build_model(input_shape, filters):

    rank = 4
    input_x1 = Input(shape=(1,))
    input_x2 = Input(shape=(1,))

    h1 = H1Layer()
    h2 = H2Layer(h1)
    h3 = H3Layer(h1,h2)
    h4 = H4Layer(h2,h3)
    h5 = H5Layer(h3,h4)
    #Branch 1
    branch_x1 = Dense(filters)(input_x1)
    branch_x1 = Dense(filters)(branch_x1)

    #Branch 2
    branch_x2 = Dense(filters)(input_x2)
    branch_x2 = Dense(filters)(branch_x2)

    #Merge
    merged = concatenate([branch_x1, branch_x2])
    merged = Dense(filters)(merged)
    merged = h2(merged)
    merged = Dense(filters)(merged)
    merged = TensorDecompositionLayer(rank)(merged)
    merged = h3(merged)
    merged = Dense(filters)(merged)
    merged = TensorDecompositionLayer(rank)(merged)
    merged = h4(merged)
    merged = Dense(filters)(merged)
    merged = TensorDecompositionLayer(rank)(merged)

    output = Dense(1)(merged)  # Single output for your function
    model = Model(inputs=[input_x1, input_x2], outputs=output)

    return model


input_shape = (2,)
filters = 128
modelN4 = build_model(input_shape, filters)
optimizer = Adam(learning_rate=0.0001) # Reduce learning rate
modelN4.compile(optimizer=optimizer, loss='mse')
```

```
batch_size = 128
epochs = 20

# history = modelN4.fit([X1_train], y_train,
#                       batch_size=batch_size,
#                       epochs=epochs,
#                       verbose=1,
#                       validation_data=(X_val, y_val),
#                       callbacks=[callback])

history = modelN4.fit([X1_train, X2_train], y_train,
                      batch_size=batch_size,
                      epochs=epochs,
                      verbose=1,
                      validation_data=([X1_val, X2_val], y_val),
                      callbacks=[callback])


val_loss = modelN4.evaluate([X1_val, X2_val], y_val, verbose=0)
print(f"Validation loss: {val_loss}")

import matplotlib.pyplot as plt
modelN4.summary()

# 1. Extract loss values
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# 2. Determine the number of epochs
actual_epochs = len(train_loss)

# 3. Create a plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, actual_epochs + 1), train_loss, label='Training Loss',
plt.plot(range(1, actual_epochs + 1), val_loss, label='Validation Loss',
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Time')
plt.legend()
plt.grid()
plt.show()
```