TF_tests\TF_test_gradient_evaluation.py

```python
# Import necessary libraries:
import tensorflow as tf
import tensorflow.keras.backend as K
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Generate random example data
x_train = np.random.rand(100, 10)
y_train = np.random.rand(100, 1)  # Modify output dimension to 2

x_eval = np.random.rand(50, 10)
y_eval = np.random.rand(50, 1)

# Create TensorFlow variables for input data
input_train = tf.Variable(x_train)
input_eval = tf.Variable(x_eval)

# Define a custom loss function
def custom_loss(y_true, y_pred):
    mse_loss = tf.reduce_mean(tf.square(y_true[:, 0] - y_pred[:, 0]))  # Mean squared error

    if tf.keras.backend.learning_phase() == 0:  # Training phase
        X = input_train
    else:    # Evaluation phase
        X = input_eval

    out = model(X)
    grad1 = tf.gradients(out[:, 0], X)[0][:, 1]
    grad1 = tf.cast(grad1,tf.float32)
    mae_loss = tf.reduce_mean(tf.abs(grad1 - out[:, 1]))  # Mean absolute error with derivative

    return mse_loss + mae_loss  # Combine the two losses

# Define your network architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(10,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(2)  # Modify output dimension to 2
])

# Compile the model with custom loss function
model.compile(optimizer='adam', loss=custom_loss)

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32,
          validation_data=(x_eval, y_eval))

# Plot the model's fitting history
losses = pd.DataFrame(model.history.history)
losses.plot()
plt.show()
```