simpleNN

April 13, 2025

[1]:

```python
import numpy as np
import os
import sys
import gym
import zipfile
import autograd
import matplotlib.gridspec as gridspec
# Use tf.random.set_seed for TensorFlow 2.0 and above
#from scipy.signal.waveforms import square
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from sklearn.model_selection import train_test_split
import random
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, model_from_json
from keras.layers import Dense
from keras.layers import Input
from tensorflow.keras import layers
```

```
2025-04-12 20:45:02.500076: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-04-12 20:45:02.503575: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-04-12 20:45:02.514590: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2025-04-12 20:45:02.556905: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2025-04-12 20:45:02.567014: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2025-04-12 20:45:02.599135: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
```

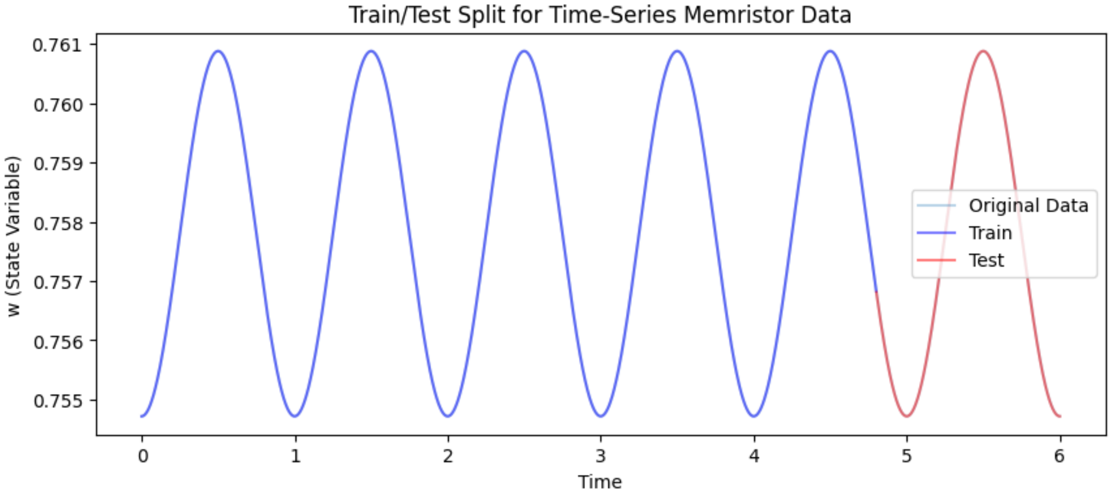
```
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2025-04-12 20:45:04.724983: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[2]:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# -------------- Physical Parameters ----------------
frequency = 1
A_train = 1.5
W_train = 2 * np.pi * frequency

mu_v = 10**4
D = 60
r_on = 0.1
r_off = 16
r0 = 4
w0 = (r0 - r_off) / (r_on - r_off)

points_per_period = 600

total_points = 10 * points_per_period

# -------------- Solving ODE ----------------
def f(t, w, A, W, mu_v, D, r_on, r_off):
    k = mu_v * (r_on / D**2)
    f_w = w * (1 - w)
    r = r_on * w + r_off * (1 - w)
    I = A * np.sin(W * t) / r
    return I * f_w * k

t_all = np.linspace(0, 6, total_points)
sol_all=solve_ivp(f, (0, 6), [w0], t_eval=t_all, args=(A_train, W_train, mu_v,␣
 ↪D, r_on, r_off),
          method='RK45', max_step=0.001)


w_all = sol_all.y[0]
v_all = A_train * np.sin(W_train * t_all)
r_all = r_on * w_all + r_off * (1 - w_all)
I_all = v_all / r_all
```

```
X_all = np.column_stack([t_all[:-1], w_all[:-1], I_all[:-1]])
y_all = w_all[1:]

# -------------- Split Data ----------------
test_ratio = 0.2
test_size = int(test_ratio * len(X_all))


test_index = np.arange(len(X_all) - test_size, len(X_all))
train_index = np.arange(0, len(X_all) - test_size)


X_train, X_test = X_all[train_index], X_all[test_index]
y_train, y_test = y_all[train_index], y_all[test_index]

plt.figure(figsize=(10, 4))
plt.plot(t_all[:-1], w_all[:-1], label="Original Data", alpha=0.3)
plt.plot(X_train[:, 0], y_train, color='blue', alpha=0.5, label='Train')
plt.plot(X_test[:, 0], y_test, color='red', alpha=0.5, label='Test')
plt.xlabel("Time")
plt.ylabel("w (State Variable)")
plt.title("Train/Test Split for Time-Series Memristor Data")

plt.legend()
plt.savefig("rungkutta_train_test.pdf")
plt.show()

from sklearn.preprocessing import StandardScaler, MinMaxScaler
import numpy as np

# ------------------ normalization------------------
scaler_time = StandardScaler()
X_train[:, 0] = scaler_time.fit_transform(X_train[:, 0].reshape(-1, 1)).
 ↪flatten()
X_test[:, 0] = scaler_time.transform(X_test[:, 0].reshape(-1, 1)).flatten()

scaler_features = MinMaxScaler(feature_range=(-1, 1))
X_train[:, 1:] = scaler_features.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler_features.transform(X_test[:, 1:])

scaler_y = MinMaxScaler(feature_range=(-1, 1))
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))

X_train_scaled = X_train
X_test_scaled = X_test
```

```
print("Mean of X_train_scaled:", np.mean(X_train, axis=0))
print("Std of X_train_scaled:", np.std(X_train, axis=0))
print("Mean of X_test_scaled:", np.mean(X_test, axis=0))
print("Std of X_test_scaled:", np.std(X_test, axis=0))
print("Mean of y_train_scaled:", np.mean(y_train_scaled))
print("Std of y_train_scaled:", np.std(y_train_scaled))
print("Mean of y_test_scaled:", np.mean(y_test_scaled))
print("Std of y_test_scaled:", np.std(y_test_scaled))
```



```
Mean of X_train_scaled: [-6.65671222e-17  3.04609012e-02  2.26622042e-02]
Std of X_train_scaled: [1.         0.70298873 0.71031816]
Mean of X_test_scaled: [ 2.16470271 -0.12662702 -0.09072466]
Std of X_test_scaled: [0.24979159 0.70958015 0.68660316]
Mean of y_train_scaled: 0.030603244374404435
Std of y_train_scaled: 0.7028492870931501
Mean of y_test_scaled: -0.12719823824974452
Std of y_test_scaled: 0.7100073018388313
```

[3]:

```
N0 = 1
Nf = X_train.shape[0]
Nd = y_train.shape[0]

#col_weights = tf.Variable(1.0)   # for ode loss
#u_weights = tf.Variable(1.0)     # for ic loss
#data_weights = tf.Variable(1.5)  # for data loss
#_______________________________________
col_weights = tf.Variable(tf.ones(Nf), dtype=tf.float32)   # weight ODE
data_weights = tf.Variable(tf.ones(Nd), dtype=tf.float32)  # data weight
u_weights = tf.Variable(tf.ones(N0), dtype=tf.float32)
```

```
optimizer_col_weights = tf.keras.optimizers.Adam(learning_rate=1e-2)
optimizer_data_weights = tf.keras.optimizers.Adam(learning_rate=1e-4)

print("done")
```

```
done

WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1744478121.689597    9994 cuda_executor.cc:1015] successful NUMA
node read from SysFS had negative value (-1), but there must be at least one
NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2025-04-12 20:45:21.690649: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:2343] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…
```

[4]:

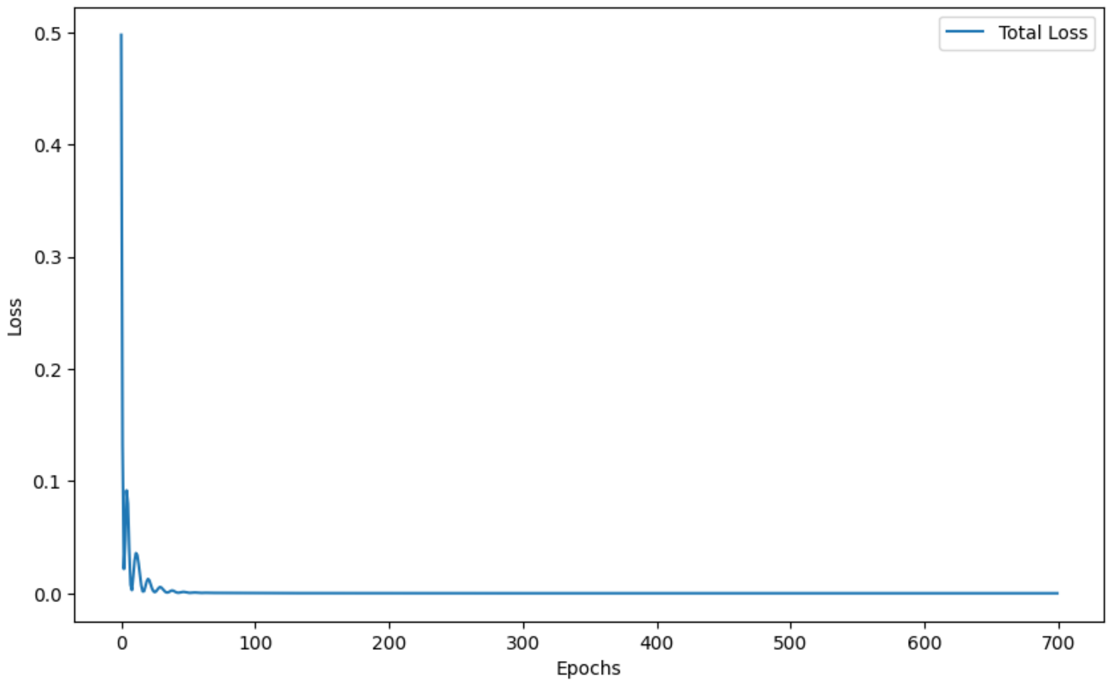
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.optimizers import Adam


simple_nn = Sequential([
    Input(shape=(3,)),
    Dense(128, activation='tanh'),
    Dense(128, activation='tanh'),
    Dense(128, activation='tanh'),
    Dense(128, activation='tanh'),
    Dense(1)
])


#simple_nn.compile(optimizer=Adam(learning_rate=0.001), loss='mse',␣
 ↪metrics=['mae'])

#history_simple_nn = simple_nn.fit(X_train_scaled, y_train_scaled,
 #                    epochs=700, batch_size=32, validation_data=(X_test_scaled,␣
 ↪y_test_scaled))
```

```
Nd = y_train.shape[0]

data_weights = tf.Variable(tf.ones(Nd), dtype=tf.float32)  #

optimizer_data_weights = tf.keras.optimizers.Adam(learning_rate=1e-4)

print("done")


def compute_loss(X, y_true, model, data_weights):
    X = tf.convert_to_tensor(X, dtype=tf.float32)
    y_true = tf.convert_to_tensor(y_true, dtype=tf.float32)

    with tf.GradientTape() as tape:
        w_pred = model(X)
        data_loss = tf.reduce_mean(tf.square(data_weights * (w_pred - y_true)))

    total_loss = data_loss
    return total_loss


#################################################### trainig loop

from tensorflow.keras.optimizers.schedules import ExponentialDecay
#earning_rate = tf.keras.optimizers.schedules.ExponentialDecay(
   #initial_learning_rate=1e-3
    #ecay_steps=10000 ,
    #ecay_rate=0.75
#
learning_rate=1e-3



optimizer=tf.keras.optimizers.Adam(learning_rate, clipnorm=1.0)


epochs = 700
train_loss_record, data_loss_record = [], []

for epoch in range(epochs):
    with tf.GradientTape(persistent=True) as tape:
        total_loss = compute_loss(X_train_scaled, y_train_scaled, simple_nn,␣
 ↪data_weights)

    grads = tape.gradient(total_loss, simple_nn.trainable_variables)
    optimizer.apply_gradients(zip(grads, simple_nn.trainable_variables))
    train_loss_record.append(total_loss.numpy())

    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch+1}/{epochs} | Total Loss: {train_loss_record[-1]:.6f}")

# ---------------- Plot Training Loss ----------------
plt.figure(figsize=(10, 6))
plt.plot(train_loss_record, label='Total Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
#plt.grid(True)
plt.show()
```

```
done
```

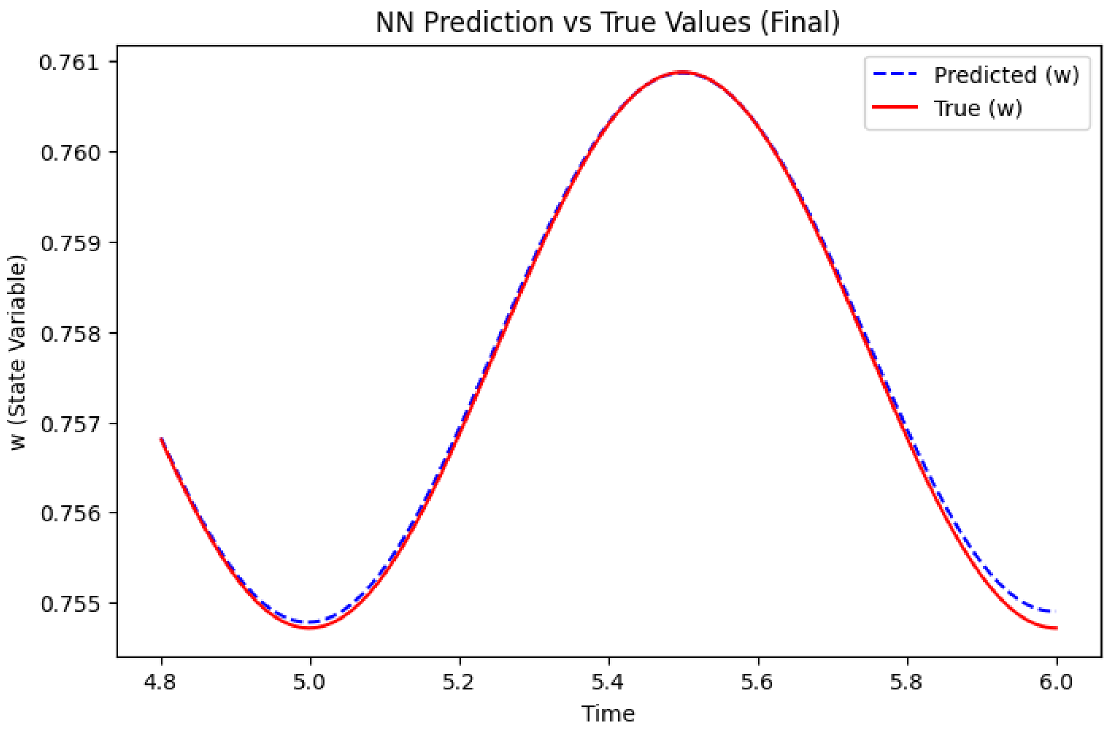
```
2025-04-12 20:45:26.824231: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 20:45:26.836700: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 20:45:26.869980: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 20:45:26.884223: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 20:45:26.916282: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
```

```
Epoch 100/700 | Total Loss: 0.000163
Epoch 200/700 | Total Loss: 0.000043
Epoch 300/700 | Total Loss: 0.000014
Epoch 400/700 | Total Loss: 0.000009
Epoch 500/700 | Total Loss: 0.000007
Epoch 600/700 | Total Loss: 0.000005
Epoch 700/700 | Total Loss: 0.000004
```



[5]:
```
w_pred_NN_scaled = simple_nn.predict(X_test_scaled)
w_pred_NN    = scaler_y.inverse_transform(w_pred_NN_scaled)

plt.figure(figsize=(8, 5))
plt.plot(t_all[test_index], w_pred_NN, label="Predicted (w)",␣
  ↪linestyle="dashed", color="blue")
plt.plot(t_all[test_index], y_test, label="True (w)", linestyle="solid",␣
  ↪color="red")
plt.xlabel("Time")
plt.ylabel("w (State Variable)")
plt.legend()
#plt.grid()
plt.title("NN Prediction vs True Values (Final)")
plt.show()
```

```
38/38                 0s 2ms/step
```


NN Prediction vs True Values (Final)
w (State Variable)
0.761
0.760
0.759
0.758
0.757
0.756
0.755
Time
4.8
5.0
5.2
5.4
5.6
5.8
6.0
Predicted (w)
True (w)

[ ]: