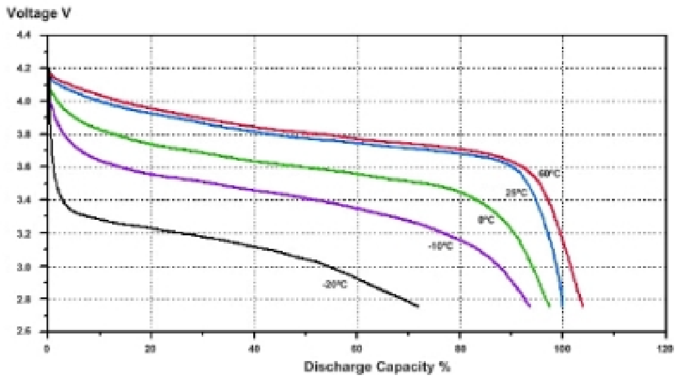
Tutorial 2

Battery Voltage estimator

When a fully charged Li ion battery (4.2 V) discharge at 1 A current , it will be get drianed at 2 hours 30 mins (150mins) with the voltage as 3.3 V.



In This project , The ML model will predict the battery voltage based on input (minutes).

Author - Manivannan

Steps to be followed

Steps to be followed in Creating a Machine Learning from scratch

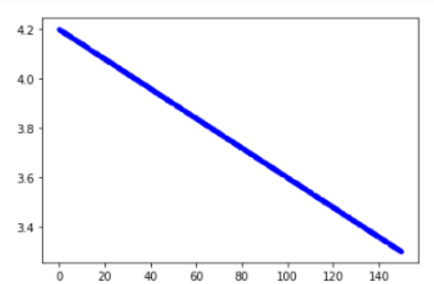
1. **Import dependencies**
2. **Training Data Generation**
3. **Noise generation**
4. **Data Splitting**
5. **Model Designing- Neural Network**
6. **Model training**
7. **Compare the loss**
8. **Test with New data**

Import dependencies

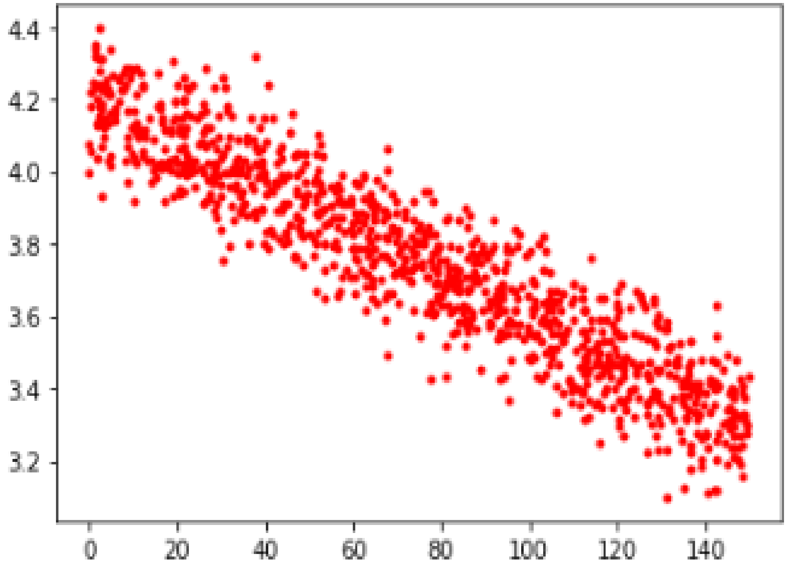
```
[1] # TensorFlow is an open source machine learning library
    import tensorflow as tf

    # Keras is TensorFlow's high-level API for deep learning
    from tensorflow import keras
    # Numpy is a math library
    import numpy as np
    # Pandas is a data manipulation library
    import pandas as pd
    # Matplotlib is a graphing library
    import matplotlib.pyplot as plt
    # Math is Python's math library
    import math

    # Set seed for experiment reproducibility
    seed = 1
    np.random.seed(seed)
    tf.random.set_seed(seed)
```

Data generation

```
# Number of sample datapoints
SAMPLES = 1000

# Generate a uniformly distributed set of random numbers in the range from
# 0 to 150 mins , which covers a complete battery discharge range time
time = np.random.uniform(
    low=0, high=150, size=SAMPLES).astype(np.float32)


# Shuffle the values to guarantee they're not in order
np.random.shuffle(time)

#The data generation will be similar to the above graph

# Calculate the corresponding voltage value using linear interpolation formula
Batteryvoltage = 4.2 + ((((time-0)*(3.3-4.2))/(150-0)))

# Plot our data. The 'b.' argument tells the library to print blue dots.
plt.plot(time, Batteryvoltage, 'b.')


plt.show()
```



```python
# Add a small random number to each y value
Batteryvoltage += 0.1 * np.random.randn(*Batteryvoltage.shape)

# Plot our data
plt.plot(time, Batteryvoltage, 'r.')
plt.show()
```



```python
# We'll use 60% of our data for training and 20% for testing. The remaining 20%
# will be used for validation.

#Calculate the indices of each section.
# Training = 600 , Testing 800
TRAIN_SPLIT =  int(0.6 * SAMPLES)
TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)

# Use np.split to chop our data into three parts.
# The second argument to np.split is an array of indices where the data will be
# split. We provide two indices, so the data will be divided into three chunks.
x_train, x_test, x_validate = np.split(time, [TRAIN_SPLIT, TEST_SPLIT])
y_train, y_test, y_validate = np.split(Batteryvoltage, [TRAIN_SPLIT, TEST_SPLIT])
# Double check that our splits add up correctly
assert (x_train.size + x_validate.size + x_test.size) ==  SAMPLES

# Plot the data in each partition in different colors:
plt.plot(x_train, y_train, 'b.', label="Train")
plt.plot(x_test, y_test, 'r.', label="Test")


plt.plot(x_validate, y_validate, 'y.', label="Validate")
plt.legend()
plt.show()
```

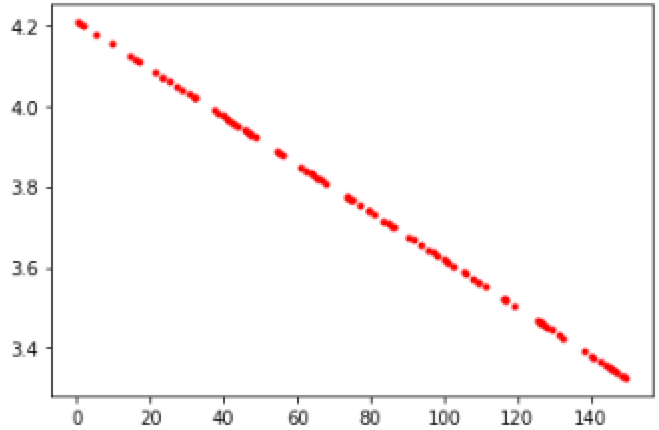
```python
# We'll use Keras to create a simple model architecture

model_1 = tf.keras.Sequential()

# First layer takes a scalar input and feeds it through 8 "neurons". The
# neurons decide whether to activate based on the 'relu' activation function.
model_1.add(keras.layers.Dense(8, input_shape=(1,)))

# The new second and third layer will help the network learn more complex representations
#model_1.add(keras.layers.Dense(8))

# Final layer is a single neuron, since we want to output a single value
model_1.add(keras.layers.Dense(1))

#model_1 = tf.keras.Sequential([keras.layers.Dense(units=1,input_shape=[1])])


# Compile the model using the standard 'adam' optimizer and the mean squared error or 'mse' loss function for regression.
model_1.compile(optimizer='adam', loss='mean_squared_error',metrics=['mae'])
```

```
# Train the model on our training data while validating on our validation set
history_1=model_1.fit(x_train,y_train,epochs=500,batch_size=32,validation_data=(x_validate,y_validate) )
```

```
[ ] print(model_1.predict([1]))
```

Important Note ** Make sure the val_loss is lesser than loss . Or else the Model is over fitting**

```
testing = np.random.uniform(
    low=0, high=150, size=100).astype(np.float32)

#print(model_1.predict([testing]))

# Plot our data
plt.plot(testing, model_1.predict([testing]), 'r.')
#plt.plot(time, Batteryvoltage, 'y.')
plt.show()
```



Link

https://github.com/Manivannan-maker/TinyML/tree/main/Tutorial-2