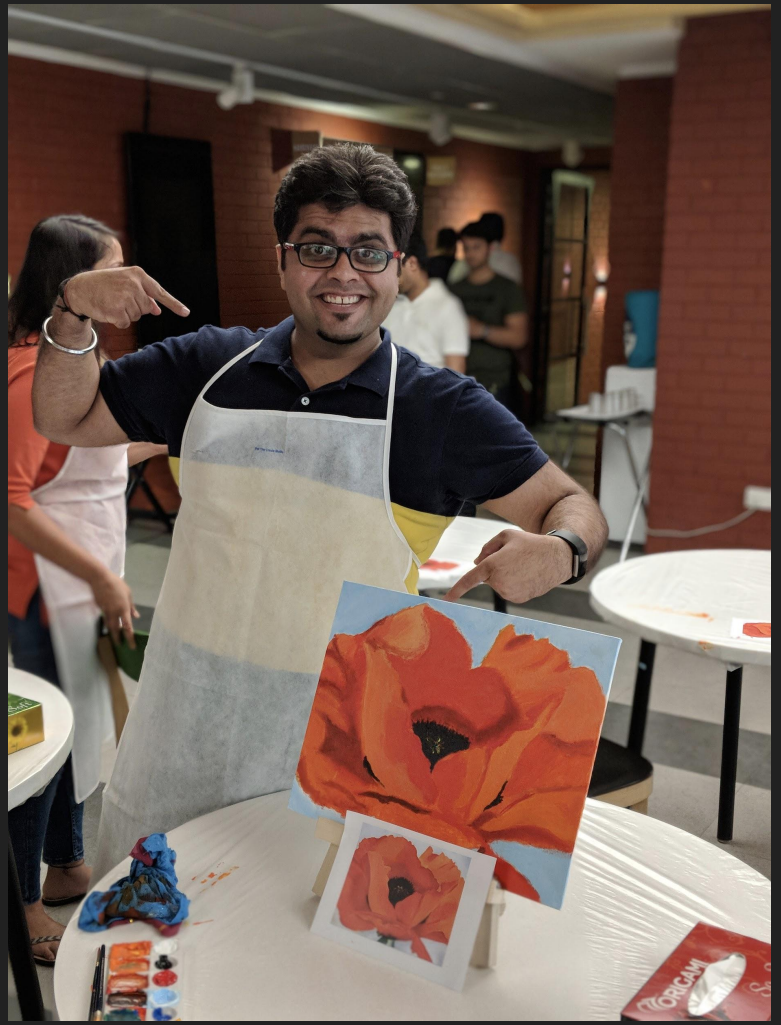
# Saurabh Rajpal

## Technical Solutions Consultant
## Google India

@imsaurabhrajpal



# how to Train your (Fly)dragon [Just Simply]

Making Predictions from 2D Data using TFJS

# What you will Build

- Webpage → using TensorFlow.js → Train model in the browser!
- “HorsePower” : “Miles Per Gallon” (Kitna Deti Hai?!)

To do this, you will:

- Load Data → Prepare it for training
- Define Model Architecture
- Train Model → Monitor Performance
- Evaluate Model → Make some predictions

# What you'll learn

- Best Practices for Data Preparation
  - Shuffling and normalization
- TensorFlow.js syntax
  - Tf.layers API
- Monitor in-browser training
  - Tfjs-vis library

# What you'll need?

- Chrome
- Text Editor - (E.g. VS Code)
- Web Server For Chrome (Chrome Extension)
- Basic knowledge of HTML, CSS, Javascript and Chrome Dev Tools
- High level understanding of ML concepts
  - Tensors
  - Transformations, etc...

# GET SET UP

# STEP 0

- Create a folder
  - Name it anything like “tf-project”

- Create 2 files
  - index.html
  - script.js

# STEP1 (index.js)


```
<!DOCTYPE html>
<html>
        <head>
                <title>TensorFlow.js Tutorial</title>
                <!-- Import TensorFlow.js -->
                <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.0.0/dist/tf.min.js"></script>
                <!-- Import tfjs-vis -->
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-vis@1.0.2/dist/tfjs-vis.umd.min.js"></script>
                <!-- Import the main script file -->
                <script src="script.js"></script>
        </head>
        <body>
        </body>
</html>
```

# STEP 2 (script.js)

```
console.log('Hello TensorFlow');
```

# LOAD, FORMAT & VISUALIZE INPUT DATA

# STEP 3

```
/**
 * Get the car data reduced to just the variables we are interested
 * and cleaned of missing data.
 */
async function getData() {
  const carsDataReq = await fetch('https://storage.googleapis.com/tfjs-tutorials/carsData.json');
  const carsData = await carsDataReq.json();
  const cleaned = carsData.map(car => ({
    mpg: car.Miles_per_Gallon,
    horsepower: car.Horsepower,
  }))
  .filter(car => (car.mpg != null && car.horsepower != null));

  return cleaned;
   }
```

# CARS DATA


```
[
  {
    "Name": "chevrolet chevelle malibu",
    "Miles_per_Gallon": 18,
    "Cylinders": 8,
    "Displacement": 307,
    "Horsepower": 130,
    "Weight_in_lbs": 3504,
    "Acceleration": 12,
    "Year": "1970-01-01",
    "Origin": "USA"
  },
…
]
```

# STEP 4

```
async function run() {
  // Load and plot the original input data that we are going to train on.
  const data = await getData();
  const values = data.map(d => ({
    x: d.horsepower,
    y: d.mpg,
  }));
  tfvis.render.scatterplot({name: 'Horsepower v MPG'},{values},
    {
      xLabel: 'Horsepower',
      yLabel: 'MPG',
      height: 300
    }
  );
  // More code will be added below
}

  document.addEventListener('DOMContentLoaded', run);
```

# VISUALIZE THE DATA

```
...
{
  "mpg":15,
  "horsepower":165,
},
{
  "mpg":18,
  "horsepower":150,
},
{
  "mpg":16,
  "horsepower":150,
},
   ...
```

# DEFINE THE MODEL ARCHITECTURE

# STEP 5

```
function createModel() {
  // Create a sequential model
  const model = tf.sequential();

  // Add a single hidden layer
  model.add(tf.layers.dense({inputShape: [1], units: 1, useBias: true}));

  // Add an output layer
  model.add(tf.layers.dense({units: 1, useBias: true}));

  return model;
  }
```

# STEP 6

```
//ADD THE FOLLOWING TO THE “RUN” FUNCTION WE DEFINED EARLIER

// Create the model

const model = createModel();

    tfvis.show.modelSummary({name: 'Model Summary'}, model);
```

# PREPARE THE DATA FOR TRAINING

# STEP 7

```
/**
 * Convert the input data to tensors that we can use for machine
 * learning. We will also do the important best practices of _shuffling_
 * the data and _normalizing_ the data
 * MPG on the y-axis.
 */
function convertToTensor(data) {
  // Wrapping these calculations in a tidy will dispose any
  // intermediate tensors.

  return tf.tidy(() => {
    // Step 1. Shuffle the data
    tf.util.shuffle(data);

    // Step 2. Convert data to Tensor
    const inputs = data.map(d => d.horsepower)
    const labels = data.map(d => d.mpg);

    const inputTensor = tf.tensor2d(inputs, [inputs.length, 1]);
    const labelTensor = tf.tensor2d(labels, [labels.length, 1]);
```

```
    //Step 3. Normalize the data to the range 0 - 1 using min-max scaling
    const inputMax = inputTensor.max();
    const inputMin = inputTensor.min();
    const labelMax = labelTensor.max();
    const labelMin = labelTensor.min();

    const normalizedInputs = inputTensor.sub(inputMin).div(inputMax.sub(inputMin));
    const normalizedLabels = labelTensor.sub(labelMin).div(labelMax.sub(labelMin));

    return {
      inputs: normalizedInputs,
      labels: normalizedLabels,
      // Return the min/max bounds so we can use them later.
      inputMax,
      inputMin,
      labelMax,
      labelMin,
    }
  });
    }
```

# TRAIN THE MODEL

# STEP 8

```
async function trainModel(model, inputs, labels) {
  // Prepare the model for training.
  model.compile({
    optimizer: tf.train.adam(),
    loss: tf.losses.meanSquaredError,
    metrics: ['mse'],
  });

  const batchSize = 32;
  const epochs = 50;

  return await model.fit(inputs, labels, {
    batchSize,
    epochs,
    shuffle: true,
    callbacks: tfvis.show.fitCallbacks(
      { name: 'Training Performance' },
      ['loss', 'mse'],
      { height: 200, callbacks: ['onEpochEnd'] }
    )
  });
    }
```

# STEP 9

```
//ADD THIS TO THE RUN FUNCTION

// Convert the data to a form we can use for training.

const tensorData = convertToTensor(data);

const {inputs, labels} = tensorData;



// Train the model

await trainModel(model, inputs, labels);

    console.log('Done Training');
```

# MAKE PREDICTIONS

# STEP 10

```
function testModel(model, inputData, normalizationData) {
  const {inputMax, inputMin, labelMin, labelMax} = normalizationData;

  // Generate predictions for a uniform range of numbers between 0 and 1;
  // We un-normalize the data by doing the inverse of the min-max scaling
  // that we did earlier.
  const [xs, preds] = tf.tidy(() => {
    const xs = tf.linspace(0, 1, 100);
    const preds = model.predict(xs.reshape([100, 1]));

    const unNormXs = xs
      .mul(inputMax.sub(inputMin))
      .add(inputMin);

    const unNormPreds = preds
      .mul(labelMax.sub(labelMin))
      .add(labelMin);

    // Un-normalize the data
    return [unNormXs.dataSync(), unNormPreds.dataSync()];
  });
```

```
const predictedPoints = Array.from(xs).map((val, i) => {
  return {x: val, y: preds[i]}
});

const originalPoints = inputData.map(d => ({
  x: d.horsepower, y: d.mpg,
}));


tfvis.render.scatterplot(
  {name: 'Model Predictions vs Original Data'},
  {values: [originalPoints, predictedPoints], series: ['original', 'predicted']},
  {
    xLabel: 'Horsepower',
    yLabel: 'MPG',
    height: 300
  }
);

  }
```

```
//Add it to the run function

// Make some predictions using the model and compare them to the

// original data

    testModel(model, data, tensorData);
```