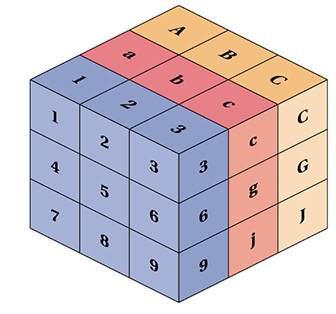
Tensors

**TensorFlow.js** is a **JavaScript** library to define and operate on **Tensors**.

The main data type in TensorFlow.js is the **Tensor**.

A **Tensor** is much the same as a multidimensional array.

A **Tensor** contains values in one or more dimensions:



A **Tensor** has the following main properties:

|  |  |
| --- | --- |
| **Property** | **Description** |
| dtype | The data type |
| rank | The number of dimensions |
| shape | The size of each dimension |

Sometimes in machine learning, the term "**dimension**" is used interchangeably with "**rank**.

[10, 5] is a 2-dimensional tensor or a 2-rank tensor.

In addition the term "dimensionality" can refer to the size of a one dimension.

Example: In the 2-dimensional tensor [10, 5], the dimensionality of the first dimension is 10.

## Creating a Tensor

The main data type in TensorFlow is the **Tensor**.

A Tensor is created from any N-dimensional array with the tf.tensor() method:

const myArr = [[1, 2, 3, 4]];  
const tensorA = tf.tensor(myArr);

### [Try itExample 2](https://www.w3schools.com/ai/tryit.asp?filename=tryai_tensor_create" \t "_blank)

[const myArr = [[1, 2], [3, 4]];  
const tensorA = tf.tensor(myArr);](https://www.w3schools.com/ai/tryit.asp?filename=tryai_tensor_create" \t "_blank)

### [YourExample 3](https://www.w3schools.com/ai/tryit.asp?filename=tryai_tensor_create" \t "_blank)

[const myArr = [[1, 2], [3, 4], [5, 6]];  
const tensorA = tf.tensor(myArr);](https://www.w3schools.com/ai/tryit.asp?filename=tryai_tensor_create" \t "_blank)

[self »](https://www.w3schools.com/ai/tryit.asp?filename=tryai_tensor_create" \t "_blank)

## Tensor Shape

A Tensor can also be created from an **array** and a **shape** parameter:

### Example1

const myArr = [1, 2, 3, 4]:  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);

onst myArr = [[1, 2], [3, 4]];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);

const myArr = [[1, 2], [3, 4]];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);  
tensorA.data().then(data => display(data));  
  
function display(data) {  
  document.getElementById("demo").innerHTML = data;  
}

const myArr = [[1, 2], [3, 4]];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);  
tensorA.array().then(array => display(array[0]));  
  
function display(data) {  
  document.getElementById("demo").innerHTML = data;  
}

const myArr = [[1, 2], [3, 4]];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);  
tensorA.array().then(array => display(array[1]));  
  
function display(data) {  
  document.getElementById("demo").innerHTML = data;  
}

const myArr = [1, 2, 3, 4];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);  
  
document.getElementById("demo").innerHTML = tensorA.rank;

[Try it Yourself »](https://www.w3schools.com/ai/tryit.asp?filename=tryai_tensor_get_rank)

---------------------------------------------------------------------------------------  
const myArr = [1, 2, 3, 4];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);  
  
document.getElementById("demo").innerHTML = tensorA.shape;

You can get the **datatype** of a tensor using tensor.dtype:

### Example

const myArr = [1, 2, 3, 4];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape);  
  
document.getElementById("demo").innerHTML = tensorA.dtype;

Tensor Data Types

A Tensor can have the following data types:

* bool
* int32
* float32 (default)
* complex64
* string

When you create a tensor, you can specify the data type as the third parameter:

const myArr = [1, 2, 3, 4];  
const shape = [2, 2];  
const tensorA = tf.tensor(myArr, shape, "int32");

# TensorFlow Operations

* Add
* Subtract
* Multiply
* Divide
* Square
* Reshape

const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);  
const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);  
  
// Tensor Addition  
const tensorNew = tensorA.add(tensorB);

const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);  
const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);  
  
// Tensor Subtraction  
const tensorNew = tensorA.sub(tensorB);  
  
// Result: [ [0, 3], [1, 6], [2, 9] ]

const tensorA = tf.tensor([1, 2, 3, 4]);  
const tensorB = tf.tensor([4, 4, 2, 2]);  
  
// Tensor Multiplication  
const tensorNew = tensorA.mul(tensorB);

const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);  
const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);  
  
// Tensor Division  
const tensorNew = tensorA.div(tensorB);

## Tensor Square

You can square a tensor using tensor.square():

### Example

const tensorA = tf.tensor([1, 2, 3, 4]);  
  
// Tensor Square  
const tensorNew = tensorA.square();  
  
// Result [ 1, 4, 9, 16 ]

## Tensor Reshape

The number of elements in a tensor is the product of the sizes in the shape.

Since there can be different shapes with the same size, it is often useful to reshape a tensor to other shapes with the same size.

You can reshape a tensor using tensor.reshape():

### Example

const tensorA = tf.tensor([[1, 2], [3, 4]]);  
const tensorB = tensorA.reshape([4, 1]);

A Tensorflow project has this typical workflow:

* Collecting Data
* Creating a Model
* Adding Layers to the Model
* Compiling the Model
* Training the Model
* Using the Model

Example

Suppose you knew a function that defined a strait line:

*Y = 1.2X + 5*

Then you could calculate any y value with the JavaScript formula:

y = 1.2 \* x + 5;

To demonstrate Tensorflow.js, we could train a Tensorflow.js model to predict Y values based on X inputs.

The TensorFlow model does not know the function.

// Create Training Data  
const xs = tf.tensor([0, 1, 2, 3, 4]);  
const ys = xs.mul(1.2).add(5);  
  
// Define a Linear Regression Model  
const model = tf.sequential();  
model.add(tf.layers.dense({units:1, inputShape:[1]}));  
  
// Specify Loss and Optimizer  
model.compile({loss:'meanSquaredError', optimizer:'sgd'});  
  
// Train the Model  
model.fit(xs, ys, {epochs:500}).then(() => {myFunction()});  
  
// Use the Model  
function myFunction() {  
  const xArr = [];  
  const yArr = [];  
  for (let x = 0; x <= 10; x++) {  
    xArr.push(x);  
    let result = model.predict(tf.tensor([Number(x)]));  
    result.data().then(y => {  
      yArr.push(Number(y));  
      if (x == 10) {plot(xArr, yArr)};  
    });  
  }  
}

# TensorFlow.js Visor

* **TensorFlow Visor** is a graphic tools for visualizing **Machine Learning**
* It contains functions for visualizing **TensorFlow Models**
* Visualizations can be organized in **Visors** (modal browser windows)
* Can be used with **Custom Tools** likes d3, Chart.js, and Plotly.js
* Often called **tfjs-vis**

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-vis"></script>

const surface = document.getElementById('demo');  
const series = ['First', 'Second'];  
  
const serie1 = [];  
const serie2 = [];  
for (let i = 0; i < 100; i++) {  
  serie1[i] = {x:i, y:Math.random() \* 100};  
  serie2[i] = {x:i, y:Math.random() \* 100};  
}  
  
const data = {values: [serie1, serie2], series}  
  
tfvis.render.scatterplot(surface, data);

Visualizations can be organized in a Visor (a modal browser window):

const series = ['First', 'Second'];  
  
const serie1 = [];  
const serie2 = [];  
for (let i = 0; i < 100; i++) {  
  serie1[i] = {x:i, y:Math.random() \* 100};  
  serie2[i] = {x:i, y:Math.random() \* 100};  
}  
  
const data = {values: [serie1, serie2], series}  
  
tfvis.render.scatterplot({name: "my Plots"}, data);

const surface = document.getElementById('demo');  
const data = [  
  {index: 0, value: 100},  
  {index: 1, value: 200},  
  {index: 2, value: 150},  
  {index: 2, value: 250},  
];  
  
tfvis.render.barchart(surface, data);

Visualizations can be organized in a Visor (a modal browser window):

const data = [  
  {index: 0, value: 100},  
  {index: 1, value: 200},  
  {index: 2, value: 150},  
  {index: 2, value: 250},  
];  
  
tfvis.render.barchart({name:"my Graphs"}, data);

## Line Graphs

### Example

const surface = document.getElementById('demo');  
  
let values = [  
  {x: 1, y: 20},  
  {x: 2, y: 30},  
  {x: 3, y: 5},  
  {x: 4, y: 12}  
];  
  
tfvis.render.linechart(surface, {values});

### Example with a Visor

let values = [  
  {x: 1, y: 20},  
  {x: 2, y: 30},  
  {x: 3, y: 5},  
  {x: 4, y: 12}  
];  
  
tfvis.render.linechart({name: 'my Lines'}, {values});

### Example with a Visor

let values = [  
  {x: 1, y: 20},  
  {x: 2, y: 30},  
  {x: 3, y: 5},  
  {x: 4, y: 12}  
];  
  
tfvis.render.linechart({name: 'my Lines'}, {values});