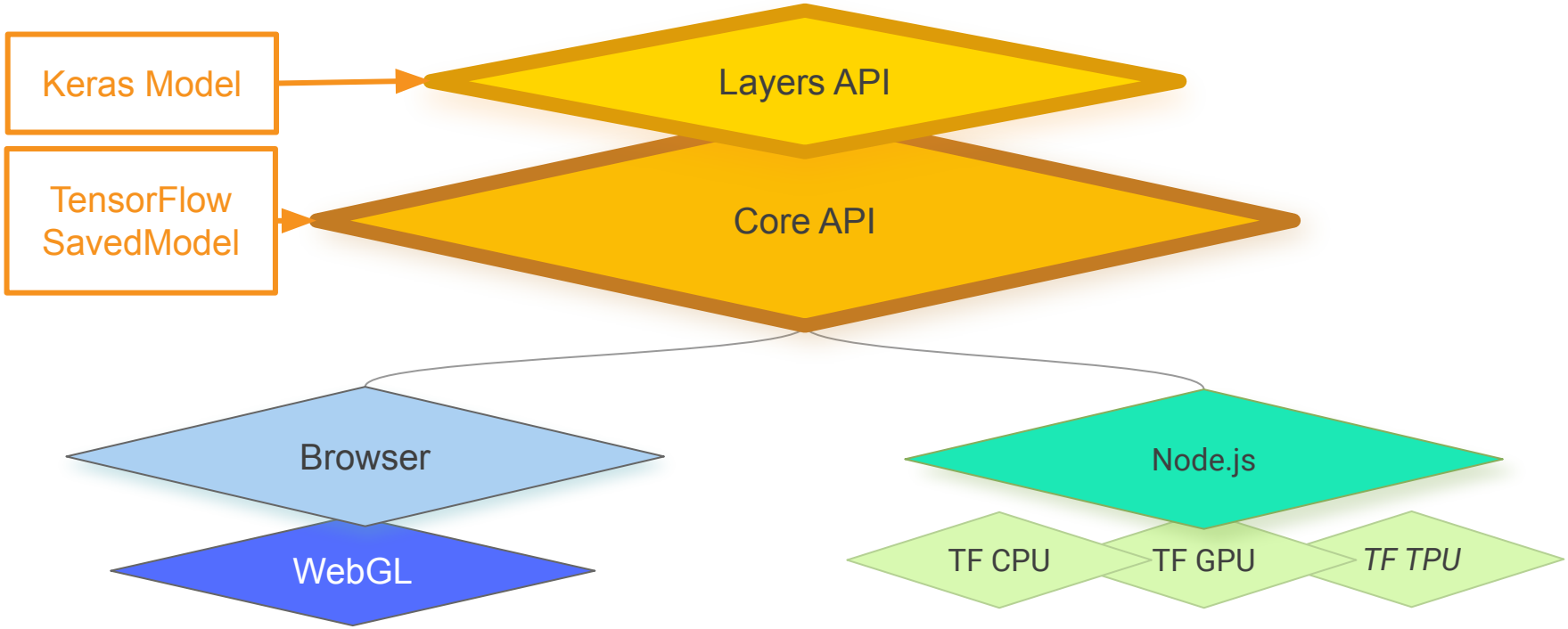
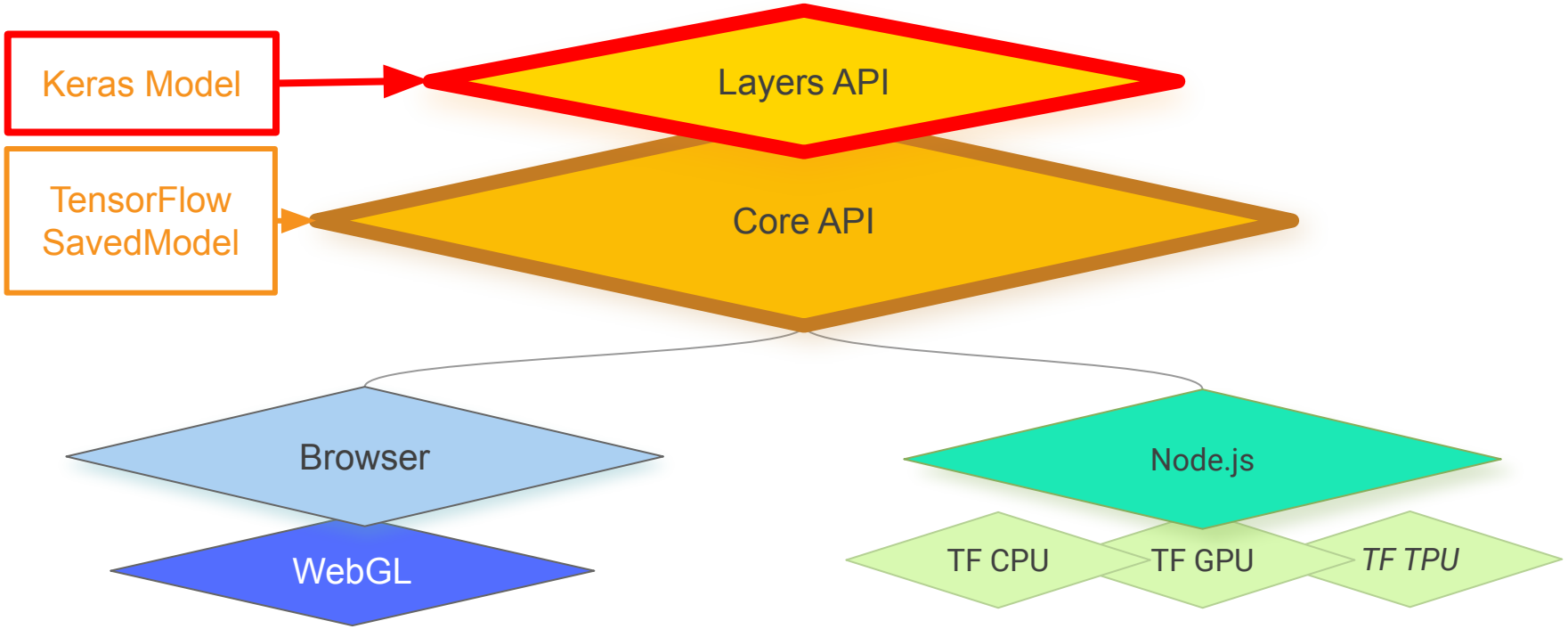
Copyright Notice

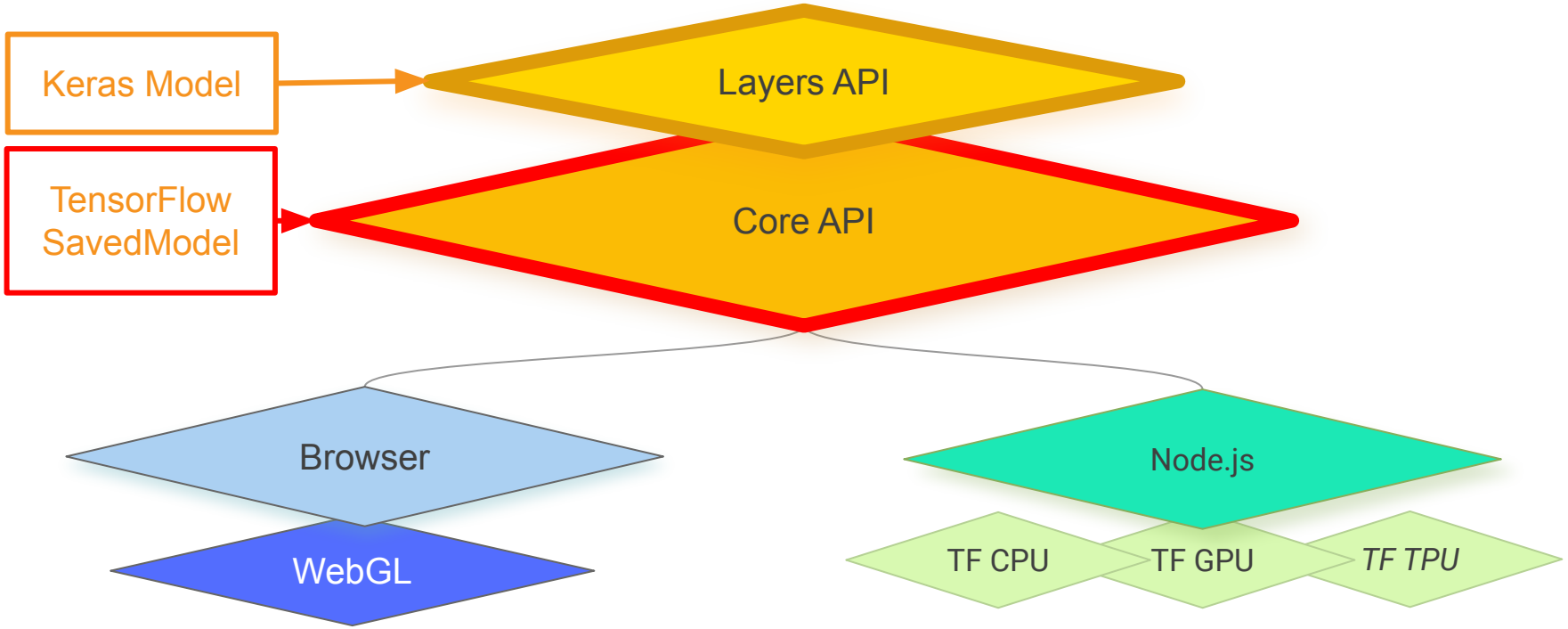
API


Keras Model
TensorFlow SavedModel
Layers API
Core API
Browser
WebGL
Node.js
TF CPU
TF GPU
TF TPU

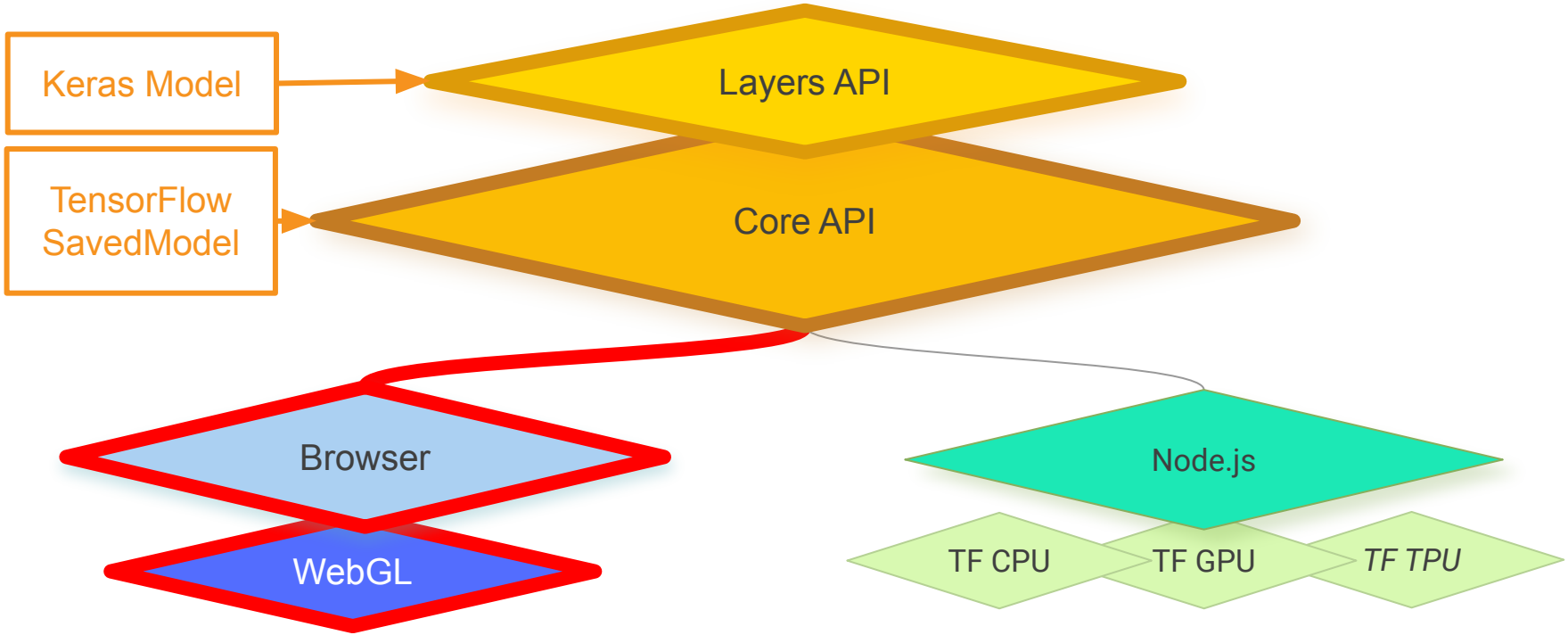
API


Keras Model
TensorFlow SavedModel
Layers API
Core API
Browser
WebGL
Node.js
TF CPU
TF GPU
TF TPU

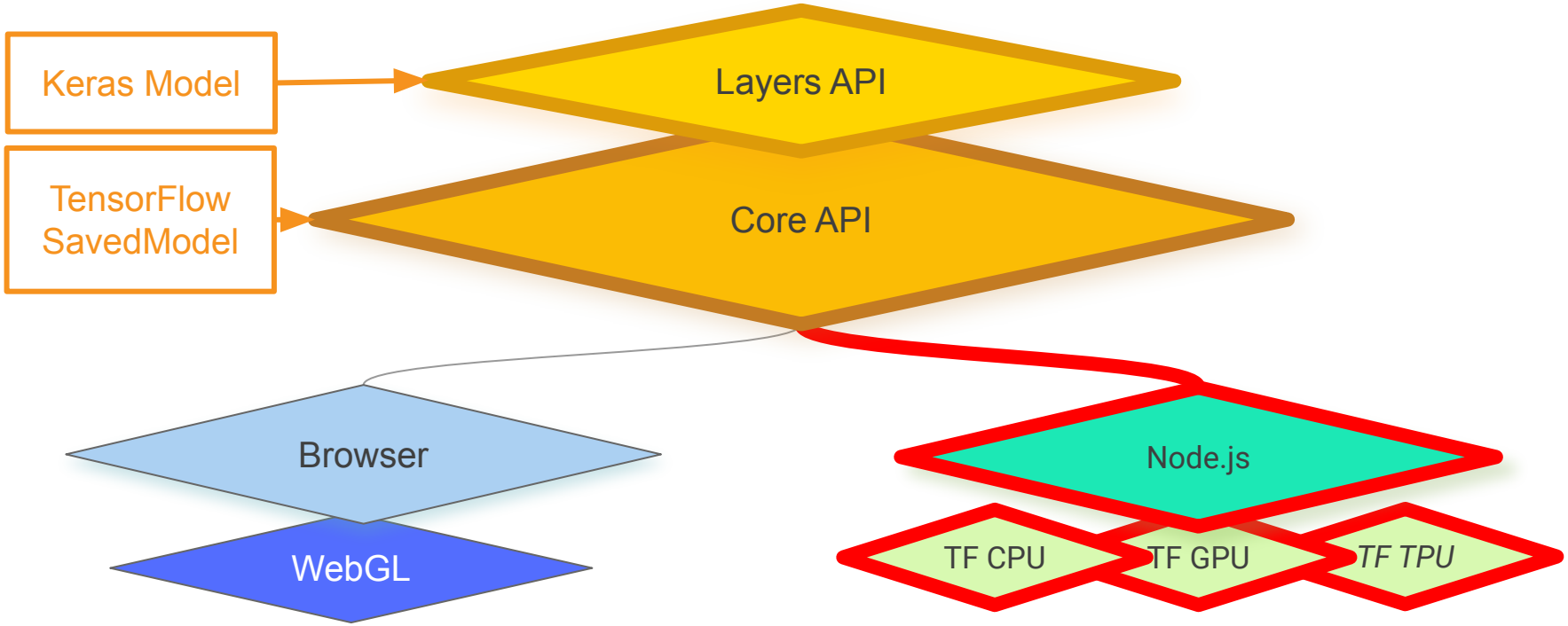
API


Keras Model
TensorFlow SavedModel
Layers API
Core API
Browser
WebGL
Node.js
TF CPU
TF GPU
TF TPU

API


Keras Model
TensorFlow SavedModel
Layers API
Core API
Browser
WebGL
Node.js
TF CPU
TF GPU
TF TPU

API


Keras Model
TensorFlow SavedModel
Layers API
Core API
Browser
WebGL
Node.js
TF CPU
TF GPU
TF TPU

```html
<html>
<head></head>
<body>
    <h1>First HTML Page</h1>
</body>
</html>
```

```html
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
```

```
<script lang="js">
    const model = tf.sequential();
    model.add(tf.layers.dense({units: 1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError',
                   optimizer:'sgd'});
    model.summary();
</script>
```

```
<script lang="js">
    const model = tf.sequential();
    model.add(tf.layers.dense({units: 1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError',
                   optimizer:'sgd'});
    model.summary();
</script>
```

```
<script lang="js">
    const model = tf.sequential();
    model.add(tf.layers.dense({units: 1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError',
                   optimizer:'sgd'});
    model.summary();
</script>
```

```
<script lang="js">
    const model = tf.sequential();
    model.add(tf.layers.dense({units: 1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError',
                   optimizer:'sgd'});
    model.summary();
</script>
```

```
<script lang="js">
    const model = tf.sequential();
    model.add(tf.layers.dense({units: 1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError',
                   optimizer:'sgd'});
    model.summary();
</script>
```

```
_________________________________________________________________
Layer (type)                 Output shape              Param #
=================================================================
dense_Dense1 (Dense)         [null,1]                  2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

```
const xs = tf.tensor2d([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], [6, 1]);
const ys = tf.tensor2d([-3.0, -1.0, 2.0, 3.0, 5.0, 7.0], [6, 1]);
```

```
const xs = tf.tensor2d([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], [6, 1]);
const ys = tf.tensor2d([-3.0, -1.0, 2.0, 3.0, 5.0, 7.0], [6, 1]);
```

```
const xs = tf.tensor2d([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], [6, 1]);
const ys = tf.tensor2d([-3.0, -1.0, 2.0, 3.0, 5.0, 7.0], [6, 1]);
```

```
doTraining(model).then(() => {
    alert(model.predict(tf.tensor2d([10], [1,1])));
});
```

```
doTraining(model).then(() => {
    alert(model.predict(tf.tensor2d([10], [1,1])));
});
```

```
doTraining(model).then(() => {
    alert(model.predict(tf.tensor2d([10], [1,1])));
});
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```javascript
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Iris Data Set

Download: Data Folder, Data Set Description

Abstract: Famous database; from Fisher, 1936

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	2512455

Source:

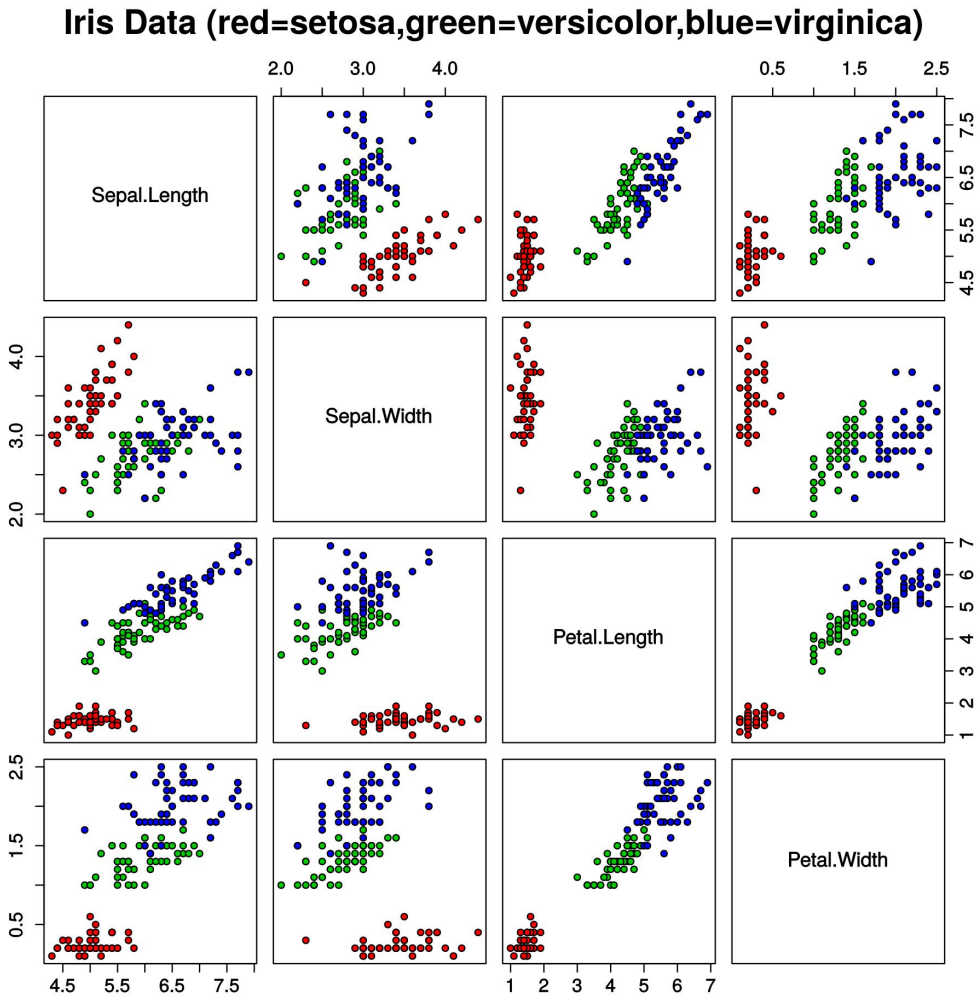
Creator:

R.A. Fisher

Donor:

Michael Marshall (MARSHALL%PLU '@' io.arc.nasa.gov)

https://archive.ics.uci.edu/ml/datasets/iris



```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
5,3.6,1.4,0.2,setosa
5.4,3.9,1.7,0.4,setosa
4.6,3.4,1.4,0.3,setosa
5,3.4,1.5,0.2,setosa
```

```
async function run() {
}
```

```
const csvUrl = 'iris.csv';
const trainingData = tf.data.csv(csvUrl, {
    columnConfigs: {
        species: {
            isLabel: true
        }
    }
});
```

```
const csvUrl = 'iris.csv';
const trainingData = tf.data.csv(csvUrl, {
    columnConfigs: {
        species: {
            isLabel: true
        }
    }
});
```

```
const csvUrl = 'iris.csv';
const trainingData = tf.data.csv(csvUrl, {
    columnConfigs: {
        species: {
            isLabel: true
        }
    }
});
```

```
const csvUrl = 'iris.csv';
const trainingData = tf.data.csv(csvUrl, {
    columnConfigs: {
        species: {
            isLabel: true
        }
    }
});
```

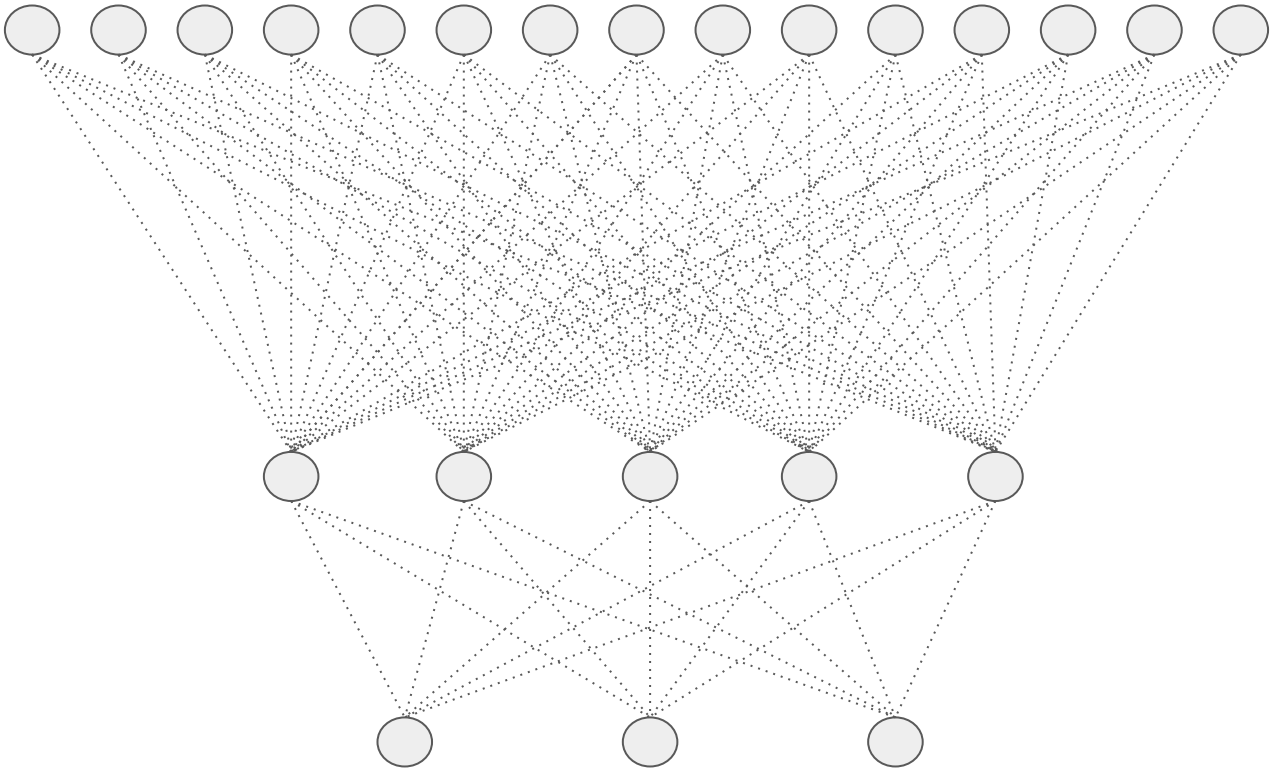
```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

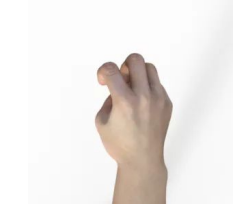
```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

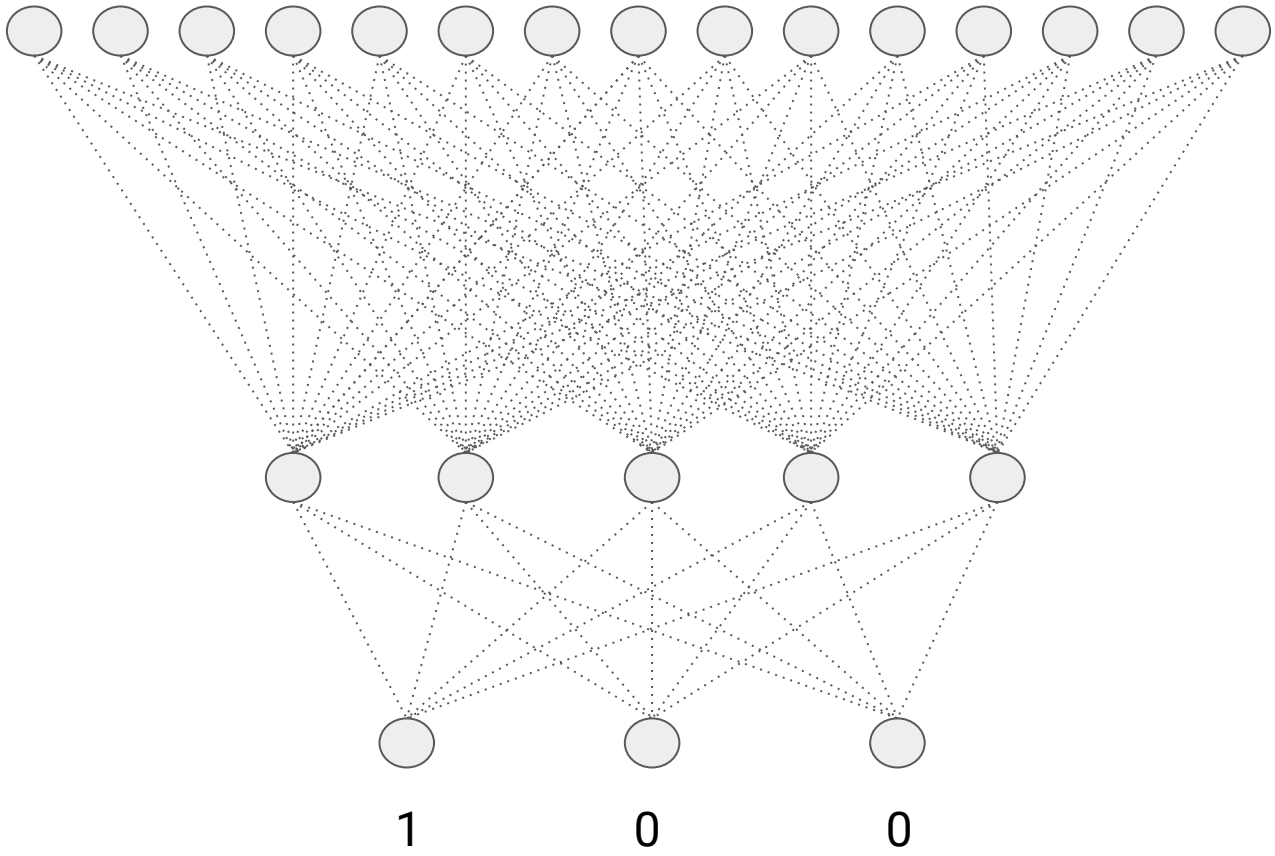
```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

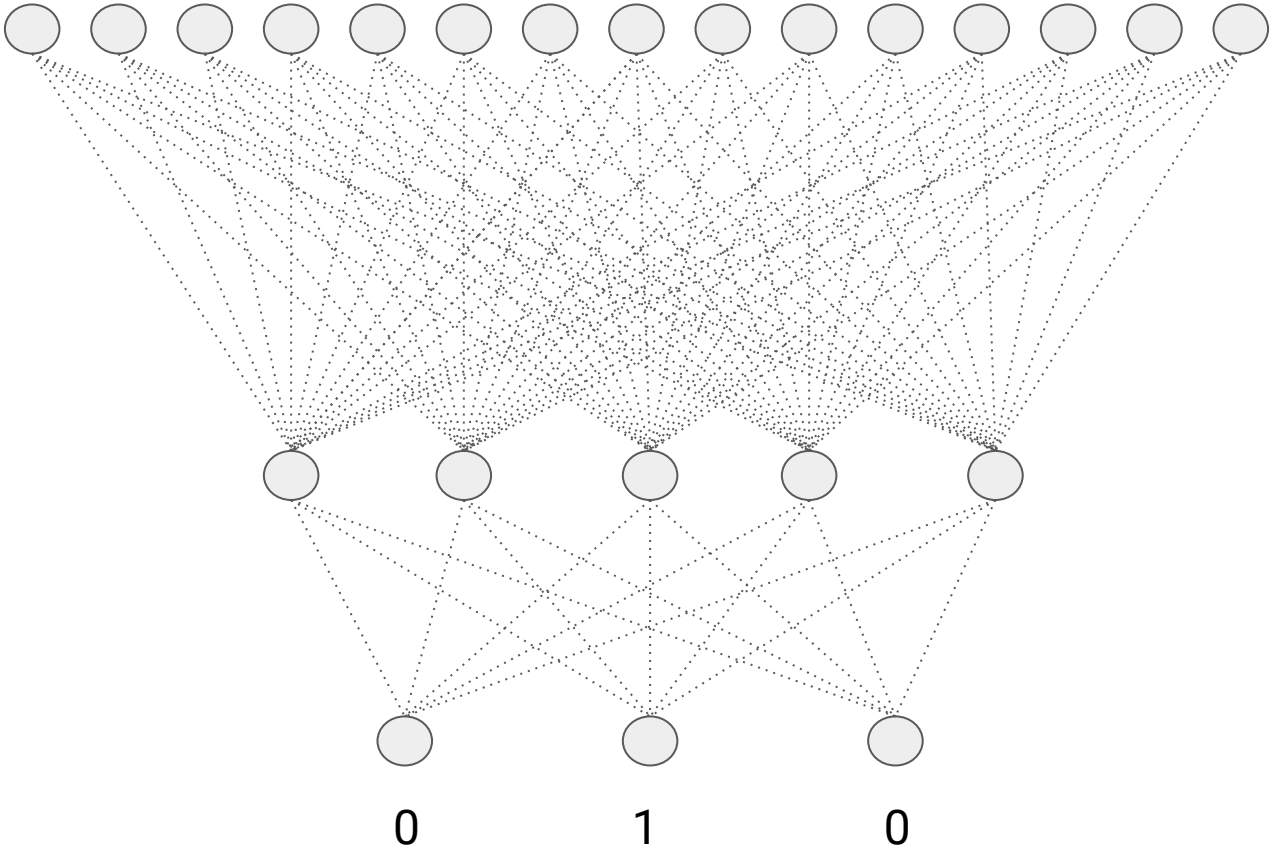
```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

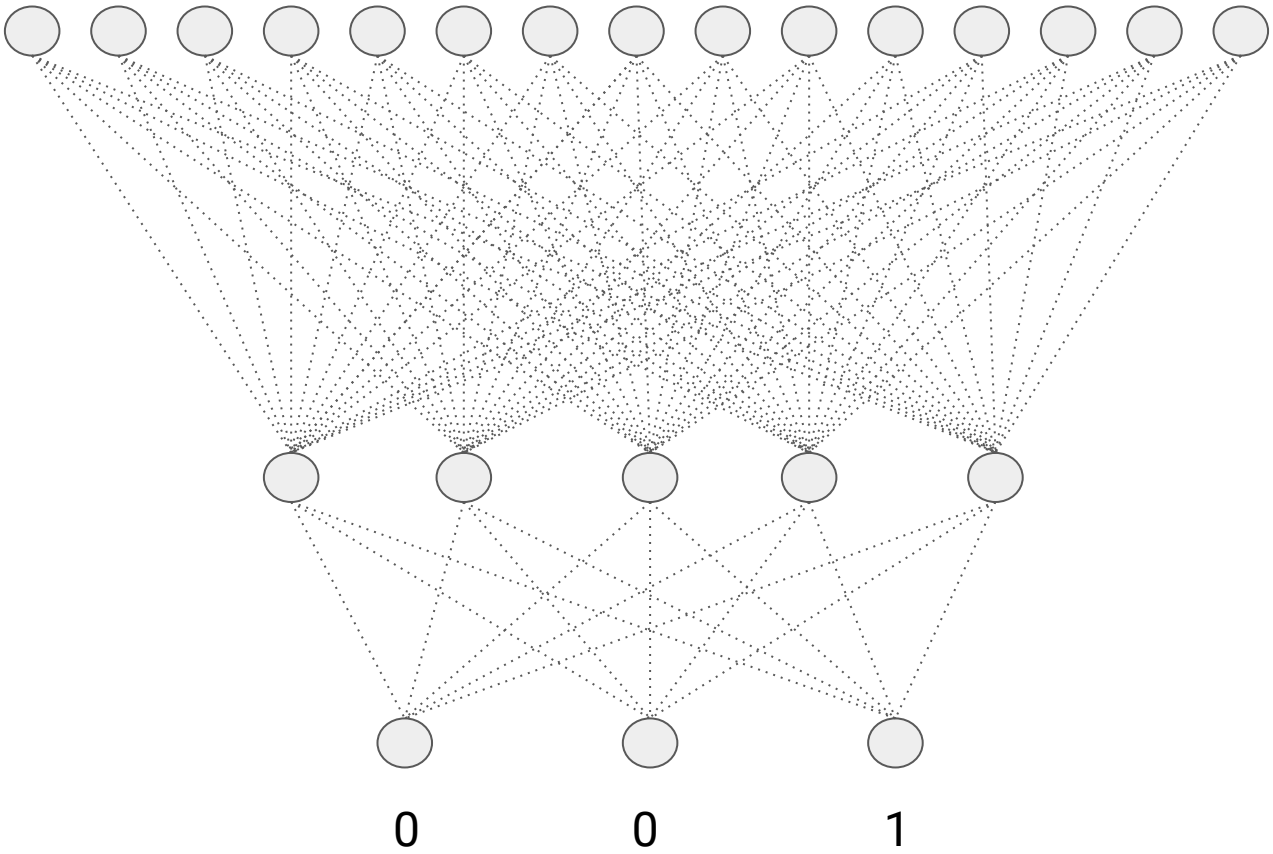
```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```






1
0
0


0
1
0


0
0
1

1 0 0

0 1 0

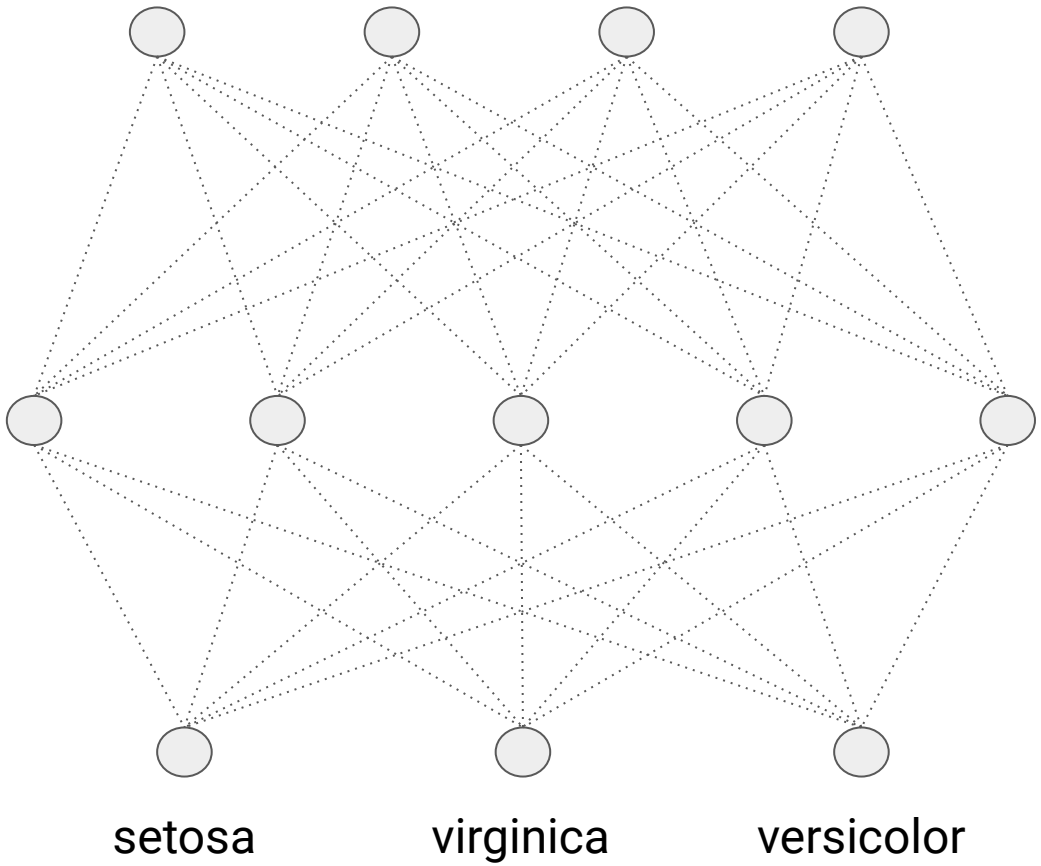
0 0 1

setosa	1	0	0
virginica	0	1	0
versicolor	0	0	1

```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```


setosa
virginica
versicolor

```
const model = tf.sequential();

model.add(tf.layers.dense({
      inputShape: [numOfFeatures],
      activation: "sigmoid", units: 5}))

model.add(tf.layers.dense({activation: "softmax", units: 3}));
```

```
const model = tf.sequential();

model.add(tf.layers.dense({
      inputShape: [numOfFeatures],
      activation: "sigmoid", units: 5}))

model.add(tf.layers.dense({activation: "softmax", units: 3}));
```

```
const model = tf.sequential();

model.add(tf.layers.dense({
      inputShape: [numOfFeatures],
      activation: "sigmoid", units: 5}))

model.add(tf.layers.dense({activation: "softmax", units: 3}));
```

```
const model = tf.sequential();

model.add(tf.layers.dense({
      inputShape: [numOfFeatures],
      activation: "sigmoid", units: 5}))

model.add(tf.layers.dense({activation: "softmax", units: 3}));
```

```
model.compile({
    loss: "categoricalCrossentropy",
    optimizer: tf.train.adam(0.06)})
```

```
await model.fitDataset(
    convertedData,
    {
        epochs:100,
        callbacks:{
        onEpochEnd: async(epoch, logs) =>{
          console.log("E: " + epoch + " Loss: " + logs.loss);
        }
    }
});
```

```
await model.fitDataset(
    convertedData,
    {
        epochs:100,
        callbacks:{
        onEpochEnd: async(epoch, logs) =>{
          console.log("E: " + epoch + " Loss: " + logs.loss);
        }
    }
});
```

```
await model.fitDataset(
    convertedData,
    {
        epochs:100,
        callbacks:{
        onEpochEnd: async(epoch, logs) =>{
          console.log("E: " + epoch + " Loss: " + logs.loss);
        }
    }
});
```

```
await model.fitDataset(
    convertedData,
    {
        epochs:100,
        callbacks:{
        onEpochEnd: async(epoch, logs) =>{
          console.log("E: " + epoch + " Loss: " + logs.loss);
        }
    }
});
```

```
const testVal = tf.tensor2d([5.8, 2.7, 5.1, 1.9], [1, 4]);
const prediction = model.predict(testVal);
alert(prediction);
```

```
const testVal = tf.tensor2d([5.8, 2.7, 5.1, 1.9], [1, 4]);
const prediction = model.predict(testVal);
alert(prediction);
```

```
const testVal = tf.tensor2d([5.8, 2.7, 5.1, 1.9], [1, 4]);
const prediction = model.predict(testVal);
alert(prediction);
```

```
const testVal = tf.tensor2d([5.8, 2.7, 5.1, 1.9], [1, 4]);
const prediction = model.predict(testVal);
alert(prediction);
```