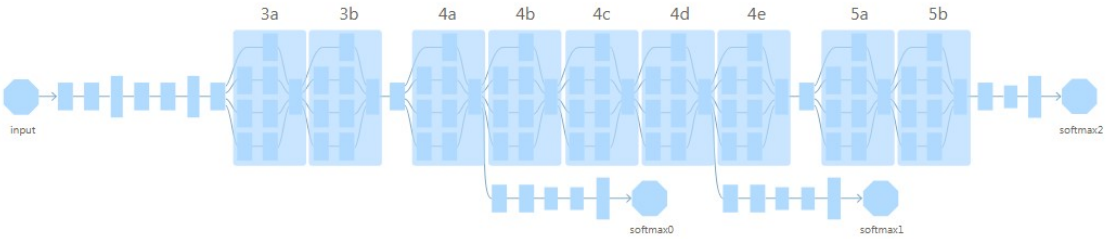
Google TensorFlow.js



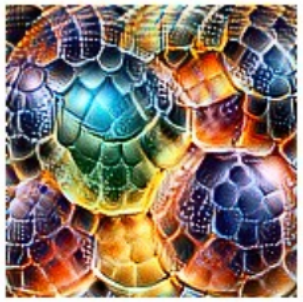
深度学习应用开发
基于TensorFlow的实践

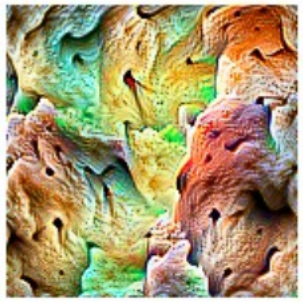
吴明晖 李卓蓉 金苍宏

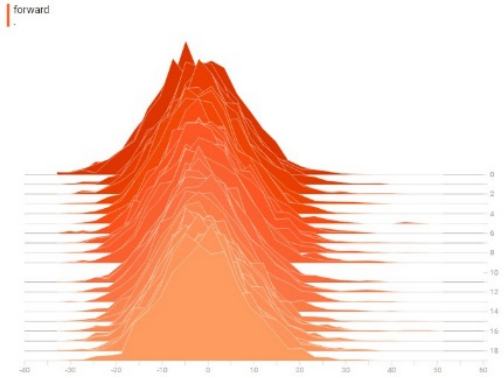
浙江大学城市学院

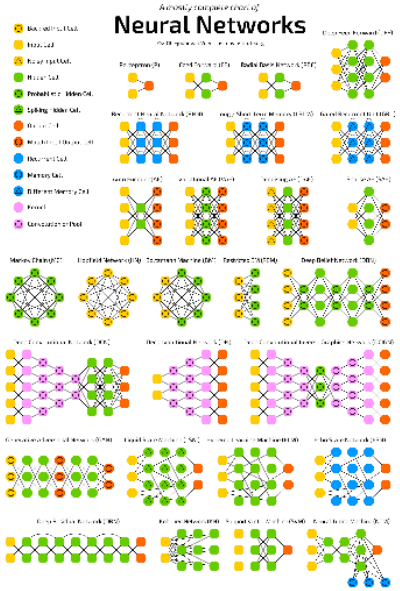
计算机与计算科学学院

Dept. of Computer Science
Zhejiang University City College









运行第一个TensorFlow.js

通过标签运行tfjs

```html
<head>
    <!-- Load TensorFlow.js -->
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.14.1/dist/tf.min.js"> </script>

    <!-- Place your code in the script tag below. You can also use an external .js file -->
    <script>

        // Define a model for linear regression.
        const model = tf.sequential();
        model.add(tf.layers.dense({units: 1, inputShape: [1]}));

        // Prepare the model for training: Specify the loss and the optimizer.
        model.compile({loss: 'meanSquaredError', optimizer: 'sgd'});

        // Generate some synthetic data for training.
        const xs = tf.tensor2d([1, 2, 3, 4], [4, 1]);
        const ys = tf.tensor2d([1, 3, 5, 7], [4, 1]);

        // Train the model using the data.
        model.fit(xs, ys, {epochs: 10}).then(() => {
            // Use the model to do inference on a data point the model hasn't seen before:
            // Open the browser devtools to see the output
            model.predict(tf.tensor2d([5], [1, 1])).print();
        });
    </script>
</head>

<body>
</body>
</html>
```

通过yarn运行tfjs

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="../js/main.js"></script>
</head>
<body>

</body>
</html>
```

通过yarn运行tfjs

```
1
import * as tf from '@tensorflow/tfjs';
3
// Define a model for linear regression.
const model = tf.sequential();
model.add(tf.layers.dense({units: 1, inputShape: [1]}));
7
// Prepare the model for training: Specify the loss and the optimizer.
model.compile({loss: 'meanSquaredError', optimizer: 'sgd'});
10
// Generate some synthetic data for training.
const xs = tf.tensor2d([1, 2, 3, 4], [4, 1]);
const ys = tf.tensor2d([1, 3, 5, 7], [4, 1]);
14
// Train the model using the data.
model.fit(xs, ys, {epochs: 10}).then(() => {
    // Use the model to do inference on a data point the model hasn't seen before:
    model.predict(tf.tensor2d([5], [1, 1])).print();
});
20
```

TensorFlow.js 核心概念

张量和变量

张量（Tensor）

TensorFlow. js中数据的中心单位是张量：一组数值形成一个或多个维度的数组。张量实例具有定义数组形状的形状属性。
在TensorFlow. js中，张量是不变的，一旦创建你就不能改变它们的值。

tf.tensor (values, shape?, dtype?) function

Creates a tf.Tensor with the provided values, shape and dtype.

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>tensors</title>
    <!-- Load TensorFlow.js -->
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.13.3/dist/tf.min.js"
    <script>
        const shape = [2,3];// 可以看做是两行三列组成
        const a = tf.tensor([1.0,2.0,3.0,10.0,20.0,30.0],shape)
        a.print()
```

0阶、1阶和高阶张量

```
// 0阶张量，即标量
tf.scalar(3.14).print(); // 3.140000104904175，  默认dtype 是 float32
tf.scalar(3.14, 'float32').print(); // 3.140000104904175
tf.scalar(3.14, 'int32').print(); // 3
tf.scalar(3.14, 'bool').print(); // 1
```

```
top                                   Filter                Default levels
Tensor                                                                  array_ops.ts:1136
    3.140000104904175
Tensor                                                                  array_ops.ts:1136
    3.140000104904175
Tensor                                                                  array_ops.ts:1136
    3
Tensor                                                                  array_ops.ts:1136
    1
```

0阶、1阶和高阶张量

```
// 1阶张量
tf.tensor1d([1, 2, 3]).print(); // [1, 2, 3]

// 2阶张量
tf.tensor2d([[1, 2], [3, 4]]).print();
tf.tensor2d([1.1, 2.1, 3.1, 4.1], [2, 2],'int32').print();
```

```
top ▼   Filter   Default levels ▼

Tensor                                          array_ops.ts:1136
    [1, 2, 3]
Tensor                                          array_ops.ts:1136
    [[1, 2],
     [3, 4]]
Tensor                                          array_ops.ts:1136
    [[1, 2],
     [3, 4]]
```

变量（Variable）

变量是通过张量进行初始化得到的。不像Tensor的值不可变，变量的值是可变的。使用变量的assign方法分配一个新的tensor到这个变量上，变量就会改变：

```
const initialValues = tf.zeros([5]);
const biases = tf.variable(initialValues); //初始化biases
biases.print(); // 输出: [0, 0, 0, 0, 0]

const updatedValues = tf.tensor1d([0, 1, 0, 1, 0]);
biases.assign(updatedValues); //更新biases的值
biases.print(); // 输出: [0, 1, 0, 1, 0]
```

```
top ▼   Filter   Default levels ▼
Tensor                                  array_ops.ts:1136
    [0, 0, 0, 0, 0]
Tensor                                  array_ops.ts:1136
    [0, 1, 0, 1, 0]
```

模型设计

人工构建模型

- 通过操作（ops）来直接完成模型本身所做的工作

```
function predict(input) {
    // y = a * x + b
    return tf.tidy(() => {
        const x = tf.scalar(input);
        const y = a.mul(x).add(b);

        return y;
    });
}

const a = tf.scalar(2);
const b = tf.scalar(5);

const result = predict(2);
result.print()
```

使用tf.model构建模型

There are two primary ways of creating models.

- Sequential — Easiest, works if the models is a simple stack of each layer's input resting on the top of the previous layer's output.
- Model — Offers more control if the layers need to be wired together in graph-like ways — multiple 'towers', layers that skip a layer, etc.

```
const model = tf.sequential();
model.add(tf.layers.dense({units: 32, inputShape: [50]}));
model.add(tf.layers.dense({units: 4}));
console.log(JSON.stringify(model.outputs[0].shape));
```

内存管理

内存管理

- TensorFlow.js计算过程中使用了显存，因此当tensorflow处理张量和变量时就有必要来管理显存。在TensorFlow.js中，可以通过dispose 和 tf.tidy这两种方法来管理内存。

tf.dispose (container) function source

Disposes any tf.Tensors found within the provided object.

dispose

```
const x = tf.tensor2d([[0.0, 2.0], [4.0, 6.0]]);
const x_squared = x.square();
x_squared.print();
x.dispose();
x_squared.dispose();
x_squared.print();
```

```
Tensor
    [[0 , 4 ],
     [16, 36]]
```

```
▸Uncaught Error: Tensor is disposed.
    at e.throwIfDisposed (tensor.ts:584)
    at e.dataSync (tensor.ts:561)
    at e.toString (tensor.ts:692)
    at Object.print (array_ops.ts:1136)
    at e.print (tensor.ts:614)
    at tensors.html?_ijt=52…touhjaoden5v64kn:71
```

tf.tidy

- 调用dispose来清除张量并释放其内存有时候可能会很麻烦。 TensorFlow.js提供了另一个函数tf.tidy，它对JavaScript中的常规范围起到类似的作用，清除所有创建的中间张量，释放它们的内存。 它不清除内部函数的返回值。
- 使用tf.tidy将有助于防止应用程序中的内存泄漏。它也可以用来更谨慎地控制内存何时回收。

tf.tidy、箭头函数

```
const average = tf.tidy(() => {
    const y = tf.tensor1d([1.0, 2.0, 3.0, 4.0]);
    const z = tf.ones([4]);

    return y.sub(z).square().mean();
});

average.print()
```

```
var func = p => p
```

等于

```
var func = function(p) {
  return p
}
```

完整实例展示

```
import * as tf from '@tensorflow/tfjs';

// 定义一个线性回归模型
const model = tf.sequential();
model.add(tf.layers.dense({units: 1, inputShape: [1]}));

// 模型训练所需要的损失函数和优化器
model.compile({loss: 'meanSquaredError', optimizer: 'sgd'});

// 生成训练数据集
const xs = tf.tensor2d([1, 2, 3, 4], [4, 1]);
const ys = tf.tensor2d([1, 3, 5, 7], [4, 1]);
```

```
15  model.fit(xs, ys, {epochs: 10}).then(() => {
16      // 输出预测结果
17      model.predict(tf.tensor2d([5], [1, 1])).print();
18  });
```

```
top ▼   Filter   Default levels ▼
Tensor                                          array_ops.ts:12
    [[5.9986029],]
```