РАФАЭЛЬ МУН(RAPHAEL MUN)

Статьи по машинному обучению в браузере с использованием фреймворка TensorFlow.js

УЧЕБНЫЕ РУКОВОДСТВА

Перевод: С. Кузнецов, 2023 г.

Articles on machine training in the browser with use of a framework TensorFlow.js

Raphael Mun

2020 · 2001

https://www.codeproject.com/Articles/instafluff#Article

Статьи по машинному обучению в браузере с использованием фреймворка TensorFlow.js

Рафаэль Мун

2020 · 2001

https://www.codeproject.com/Articles/instafluff#Article

Перевод: С. Кузнецов, 30.10.2023

Статья «Начало работы с глубоким обучением в браузере с использованием фреймворка TensorFlow.js»

Статья **«Начало работы с глубоким изучением в браузере с использованием фреймворка TensorFlow.js»(«Getting Started With Deep Learning in Your Browser Using TensorFlow.js»;** https://www.codeproject.com/Articles/5272760/Getting-Started-With-Deep-Learning-in-Your-Browser **)** является статьей из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**

В этой статье я покажу вам, как быстро и легко установить и использовать фреймворк **TensorFlow.js**, чтобы обучить **нейронную сеть(neural network)** делать **прогнозы(predictions)** из точек данных.

Здесь мы рассмотрим темы:

- установка **TensorFlow.js**
- создание данных обучения(тренировки)
- определение модели нейронной сети
- обучение **искусственного интеллекта(Artificial Intelligence; AI)**.

TensorFlow + JavaScript. Самый популярный, ультрасовременный **AI**-фреймворк теперь поддерживает наиболее широко используемый язык программирования на планете, поэтому давайте заставим волшебство произойти посредством **глубокого изучения(deep learning)** прямо в нашем веб-браузере, ускоренном **графическим процессорным устройством(ГПУ; GPU)** через графическую библиотеку **WebGL**, используя фреймворк обучения **TensorFlow.js**!

В этой статье я покажу вам, как быстро и легко установить и использовать фреймворк `TensorFlow.js`, чтобы обучить `нейронную сеть(neural network)` делать `прогнозы(predictions)` из точек данных.

Вы можете найти код для этого проекта вместе с другими примерами путем щелчка по кнопке `'Browse Code'` в меню налево или загрузив `.zip`-файл, присоединенный выше.

Установка фреймворка TensorFlow.js

Первый шаг должен создать `HTML`-файл веб-страницы, такой как `index.html`, и внутри тега `<head>`, и внутри тега скрипта `<script>` включить ссылку на фреймворк `TensorFlow.js`, что позволит нам работать с фреймворком `TensorFlow`, используя объект `tf`.

JavaScript

```
<script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
```

Далее показан шаблон начальной страницы, который мы можем использовать для наших проектов, с разделом, зарезервированным для нашего кода:

HTML

```
<html>
    <head>
        <title>Глубокое обучение в браузере с помощью TensorFlow.js/Deep
Learning in Your Browser with TensorFlow.js</title>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
    </head>
    <body>
        <script>
        (async () => {
         // Ваш код поместите сюда
```

```
        // Your Code Goes Here
      })();
      </script>
    </body>
</html>
```

Весь код, обсужденный в этой статье, будет помещен в асинхронном `async`-разделе обертки веб-страницы.

Чтобы выполнить код, откройте вышеупомянутую веб-страницу в любом современном браузере, затем запустите журнал диалогового отладчика (нажатие клавиши `F12` сработает в большинстве браузеров).

Асинхронный `async`-раздел обертки веб-страницы позволит нам работать с асинхронной функцией `TensorFlow`, используя ключевое слово ожидания `await` без набора условия `.then()` цепочки кода. Это должно быть довольно прямо и понятно; однако, если вы хотите более глубоко познакомиться с этим шаблоном, я рекомендую почитать это руководство guide.

Создание данных обучения(тренировки)

Мы собираемся тиражировать `логический элемент булевой логики XOR(XOR boolean logic gate)`, принимающий на входе два выбора и проверяющий и выводящий результат, чтобы один из них был выбран, но не были выбраны оба одновременно.

Это похоже на эту таблицу:

A	B	Output
0	0	0
0	**1**	**1**
1	0	**1**
1	**1**	0

В коде мы можем установить данные в виде двух массивов чисел, один для входа двух выборов, и один массив для ожидаемого вывода:

pyhton

```
// XOR datac
const data = [ [ 0, 0 ], [ 0, 1 ], [ 1, 0 ], [ 1, 1 ] ];
```

```
const output = [ [ 0 ], [ 1 ], [ 1 ], [ 0 ] ];
```

Затем, мы должны преобразовать значения в данные "тензоры"("tensors"), чтобы подготовить их к использованию в фреймворке TensorFlow. Мы можем сделать это, создавая 1-мерный тензор для каждого элемента и пакетируя их в "стек"("stack") , как в этом коде:

pyhton

```
const xs = tf.stack( data.map( a =>   tf.tensor1d( a ) ) );
const ys = tf.stack( output.map( a => tf.tensor1d( a ) ) );
```

Альтернатива: вот версия кода для цикла, если вы не знакомы с функцией отображения map().

```
const dataTensors = [];
const outputTensors = [];
for( let i = 0; i < data.length; i++ ) {
    dataTensors.push( tf.tensor1d( data[ i ] ) );
    outputTensors.push( tf.tensor1d( output[ i ] ) );
}
const xs = tf.stack( dataTensors );
const ys = tf.stack( outputTensors );
```

Контрольная точка

Теперь ваш код должен выглядеть подобным этому коду:

HTML

```
<html>
    <head>
        <title>Deep Learning in Your Browser with TensorFlow.js</title>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
    </head>
    <body>
        <script>
        (async () => {
            // XOR data
            const data = [ [ 0, 0 ], [ 0, 1 ], [ 1, 0 ], [ 1, 1 ] ];
            const output = [ [ 0 ], [ 1 ], [ 1 ], [ 0 ] ];
```

```
            const xs = tf.stack( data.map( a => tf.tensor1d( a ) ) );
            const ys = tf.stack( output.map( a => tf.tensor1d( a ) ) );
        })();
        </script>
    </body>
</html>
```

Определение модели нейронной сети

Теперь давайте установим нашу сеть для некоторого **глубокого обучения(deep learning)**. Слово **"Глубоко"("Deep")** обозначает сложность нашей сети и обычно определен как три или больше **"скрытых слоя"("hidden layers")** в сети. Давайте построим модель.

```
// Определите нашу модель с помощью нескольких скрытых слоев
// Define our model with several hidden layers
const model = tf.sequential();
model.add(tf.layers.dense( { units: 100, inputShape: [ 2 ] } ) );
model.add(tf.layers.dense( { units: 100, activation: 'relu' } ) );
model.add(tf.layers.dense( { units: 10, activation: 'relu' } ) );
model.add(tf.layers.dense( { units: 1 } ) );
model.summary();
```

Мы определяем нашу **модель последовательно(model sequentially)**. Чтобы соответствовать нашим данным, наша нейронная сеть должна взять входную форму из **2** чисел и вывод из **1** числа.

У первых двух **"скрытых слоев"("hidden layers")** есть **100** узлов или модулей, и третий слой имеет **10** узлов. Поскольку функция активации **relu(Rectified Linear Unit; Очищенный линейный модуль)** учится быстро и выступает хорошо в большинстве случаев, то этот код здесь является хорошим выбором по умолчанию.

Обучение(тренировка) искусственного интеллекта(Artificial Intelligence; AI)

Остался всего один шаг и он должен на наших данных обучить нашу **нейронную сеть(neural network)**. В фреймворке **TensorFlow** это означает, что мы просто компилируем нашу модель и соответствуем нашим данным

специфичным количеством итераций или "эпох"(specified number of iterations or "epochs.").

Python

```
model.compile( { loss: 'meanSquaredError', optimizer: "adam", metrics: [
"acc" ] } );

// Train the model using the data.
let result = await model.fit( xs, ys, {
    epochs: 100,
    shuffle: true,
    callbacks: {
        onEpochEnd: ( epoch, logs ) => {
            console.log( "Epoch #", epoch, logs );
        }
    }
} );
```

Точно так же, как в нашей модели для функции активации мы выбрали relu(Rectified Linear Unit; Очищенный линейный модуль), то мы будем использовать встроенную функцию среднеквадратичной ошибки meanSquaredError и встроенную функцию adam(потеря и оптимизатор), которые соответствуют большинству сценариев.

Поскольку наши данные обучения(тренировки) XOR не зависимы от порядка(не как, например, данные временного ряда, такие как погода или температура в течение времени), мы хотим включить режим shuffle(перестановки), и мы будем обучать в течение 100 эпох.

Тестирование результата

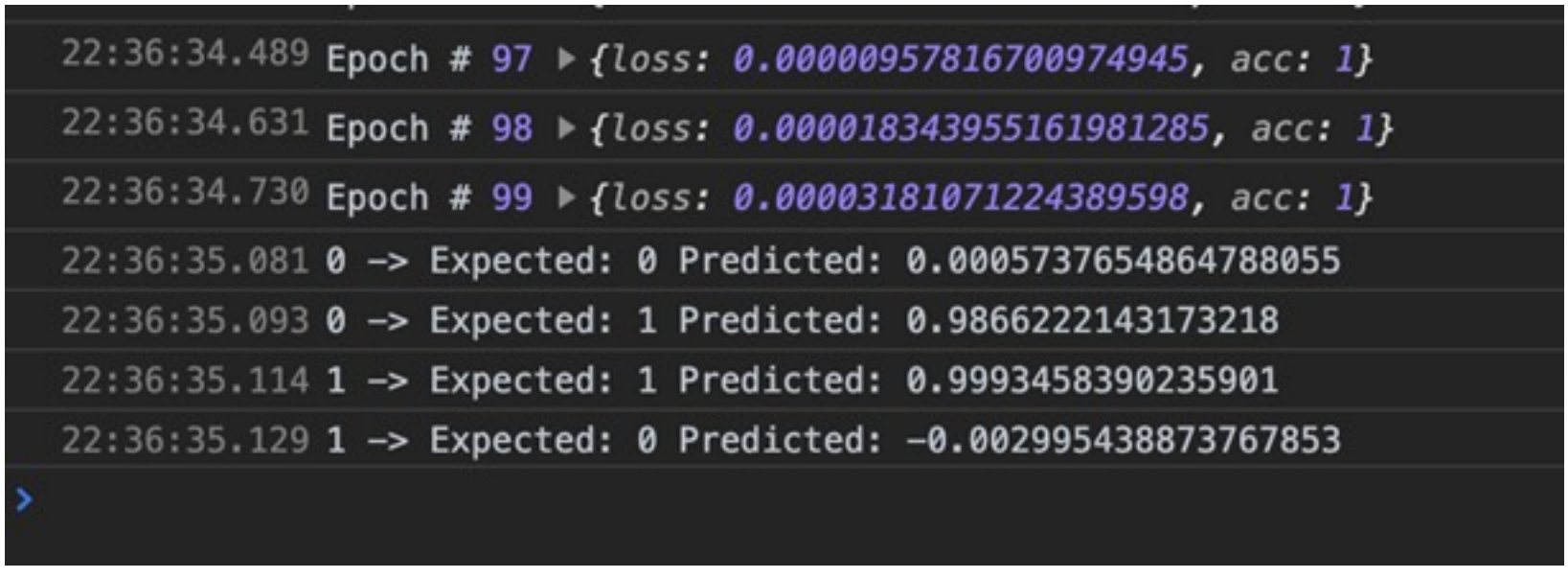
Наконец пора использовать нашу обученную нейронную сеть(trained neural network). Мы должны у модели выполнить функцию предсказания predict() с нашими вводами и данными и затем распечатать выход.

Python

```
for( let i = 0; i < data.length; i++ ) {
    let x = data[ i ];
    let result = model.predict( tf.stack( [ tf.tensor1d( x ) ] ) );
    let prediction = await result.data();
```

```
    console.log( x[ 0 ] + " ->  Ожидаемо:/Expected: " + output[ i ][ 0 ] +
" Предсказано:/Predicted: " + prediction[ 0 ] );
}
```

После того, как обучение(тренировка) завершено, то консоль отладчика браузера покажет вывод, подобный этому:



Финишная черта

Вот полный код скрипта:

HTML

```
<html>
    <head>
        <title>Deep Learning in Your Browser with TensorFlow.js</title>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
    </head>
    <body>
        <script>
        (async () => {
            // XOR data
            const data = [ [ 0, 0 ], [ 0, 1 ], [ 1, 0 ], [ 1, 1 ] ];
            const output = [ [ 0 ], [ 1 ], [ 1 ], [ 0 ] ];

            const xs = tf.stack( data.map( a => tf.tensor1d( a ) ) );
            const ys = tf.stack( output.map( a => tf.tensor1d( a ) ) );

            // Define our model with several hidden layers
            const model = tf.sequential();
            model.add(tf.layers.dense( { units: 100, inputShape: [ 2 ] } )
);
            model.add(tf.layers.dense( { units: 100, activation: 'relu' }
) );
```

```
            model.add(tf.layers.dense( { units: 10, activation: 'relu' } )
);
            model.add(tf.layers.dense( { units: 1 } ) );
            model.summary();

            model.compile( { loss: 'meanSquaredError', optimizer: "adam",
metrics: [ "acc" ] } );

            // Train the model using the data.
            let result = await model.fit( xs, ys, {
                epochs: 100,
                shuffle: true,
                callbacks: {
                    onEpochEnd: ( epoch, logs ) => {
                        console.log( "Epoch #", epoch, logs );
                    }
                }
            } );

            for( let i = 0; i < data.length; i++ ) {
                let x = data[ i ];
                let result = model.predict( tf.stack( [ tf.tensor1d( x ) ]
) );
                let prediction = await result.data();
                console.log( x[ 0 ] + " -> Ожидаемо:/Expected: " + output[
i ][ 0 ] + " Предсказано:/Predicted: " + prediction[ 0 ] );
            }
        })();
        </script>
    </body>
</html>
```

Примечание по использованию памяти

Чтобы сохранить вещи простыми, это учебное руководство не волнуется об использовании памяти тензоров и удалении их позже; однако, это имеет значение для больших, более сложных проектов. Фреймворк `TensorFlow.js` выделяет тензоры в `графическом процессорном устройстве(ГПУ; GPU)`, и мы должны удалить их сами, если мы хотим предотвратить утечки памяти. Мы можем сделать это, используя функцию размещения `dispose()` на каждом объекте тензора. Альтернативно, мы можем позволить фреймворку `TensorFlow.js` автоматически управлять размещением тензора, обертывая наш код в функцию `tf.tidy()` как в этом коде:

Python

```
for( let i = 0; i < data.length; i++ ) {
    let x = data[ i ];
    let result = tf.tidy( () => {
        return model.predict( tf.stack( [ tf.tensor1d( x ) ] ) );
    });
    let prediction = await result.data();
    result.dispose();
    console.log( x[ 0 ] + " -> Expected: " + output[ i ][ 0 ] + "
Predicted: " + prediction[ 0 ] );
}
```

Что далее? Собаки и пицца?

Вы увидели, как просто установить и использовать фреймворк машинного обучения **TensorFlow** в браузере. Не останавливайтесь здесь, есть больше, что изучить! Как насчет того, чтобы на базе прогресса в обучении, сделанного ранее, мы пробуем сделать что-то более интересное, такое как обнаружение животных и объектов в изображении?

В серии статей, смотрите следующую статью **Собаки и Пицца:Машинное зрение в браузере с помощью фреймворка TensorFlow.js(Dogs and Pizza: Computer Vision in the Browser with TensorFlow.js)**.

Эта статья - статья из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**