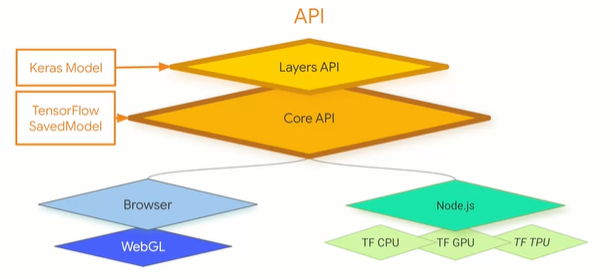
Browser-based Models with TensorFlow.js

Week 1

In this course I’ll go learn how to training and inference using JavaScript. This will allow us to take our knowledge of ML models and use them in the browser as well as on backend servers like Node.js. The design and architecture of TensorFlow.js:



The goals is twofold, first we want to make it easy for you to code against it with a friendly high-level API, but we can also go lower into the APIs and program against them directly too. Its designed to run in the browser and as well as an Node.js server

Simple start to see how to build for TensorFlow.js, so first things first we’re going to need a web page ex: simplest possible one:

<html>

<head></head>

<body>

<h1>First HTML Page</h1>

</body>

</html>

Add a script tag below the head and above the body to load the TensorFlow.js file with this code:

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

Lets do the simple example, and add this above the body tag in your HTML page:

<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>

r1-r6

r2 🡪 defines the model to be a sequential

r3 🡪 simples possible nn is one layer with one neuron, so we’re only adding one dense layer to the sequence, and the units = neuron

r4 🡪 compiled a neural network with loss and optimizer

before the closing script tag add this code:

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]);

r1-r2 🡪 we’re using tensor2d because there is no numpy array in java script, and when using tensor2d you have a two dimensional array or two one-dimensional array. So in this case our training values are in one array and the second array is the shape of those training values. SO in here we’re using 6 x values in one-dimensional array and that’s why the second parameter is 6,1

Training should be an asynchronous function because it will take an indeterminate time to complete. So our next piece of code will call an asynchronous function called doTraining:

doTraining(model).then(() => {

alert(model.predict(tf.tensor2d([10], [1,1])));

});

r1-r3

r1 🡪 as we know training can take an indeterminate amount of time, and we don’t want to lock the browser while this is going on. So it’s better to do it as an asynchronous function that calls us back when it’s done. We call it and pass it the model that we just created. Then when it calls back, the model is trained and at that point we can call model predict and in this example we’ll try to predict the value for 10. We again have to create a tensor 2D with the first dimension being an array containing the value that we want to predict in this case 10, and the second being the size of that array 1,1.

Code for training the model, this code should go at the top of the script block that you’ve been creating:

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);

}

}

});

}

r1-r11

r3 🡪 model.fit is function to do model training

r4-~ 🡪 json list

The full code from start to end for this example:

<html>

<head></head>

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

<script lang="js">

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);

}

}

});

}

const model = tf.sequential();

model.add(tf.layers.dense({units: 1, inputShape: [1]}));

model.compile({loss:'meanSquaredError',

optimizer:'sgd'});

model.summary();

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])));

});

</script>

<body>

<h1>First HTML Page</h1>

</body>

</html>

Reading CSV file with TensorFlow.js

Start with putting an asynchronous function into a JavaScript block:

async function run() {

}

To load the data from the CSV you’ll use code like this:

const csvUr1 = ‘iris.csv’ :

const trainingData = tf.data.csv(csvUr1, {

columnConfigs: {

species: {

isLabel: true

}

}

});

r1-r8

r1 🡪 the CSV is at a URL, we don’t have the server or protocol details, which means it’s going to try to load it from the same directory as the web page it’s hosting it. It is loading from the file system directly.

r2 🡪 it uses the tf.data.csv class to handle loading and parsing the data. Tf.data.csv takes care of CSV management for you

the data is define as dictionary we want to convert it into array. We also want to convert the strings defined in the labels into a one hot encoded array of label values

Const convertedData =

trainingData.map(({xs, ys}) => {

const labels = [

ys.species == “setosa” ? 1 : 0,

ys.species == “virginica” ? 1 : 0,

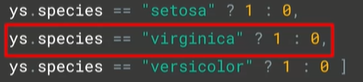
ys.species == “versicolor” ? 1 : 0 ]

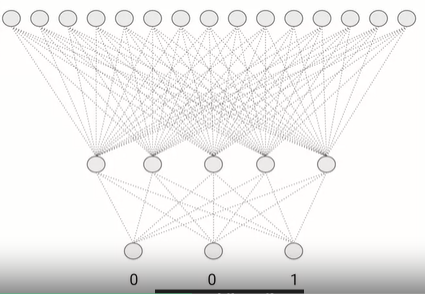
return{ xs: Object.values(xs), ys:Object.values(labels)};

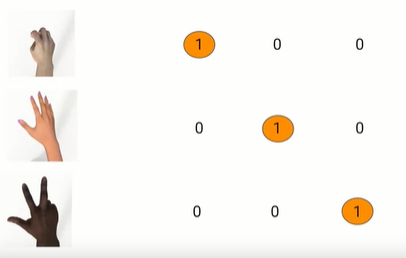
}).batch(10);

we hard encoded it using the const labels. The values that weren’t flagged as labels are in the x set in object.values

one-hot encoding 🡪 one of the values in the array is the hot value:

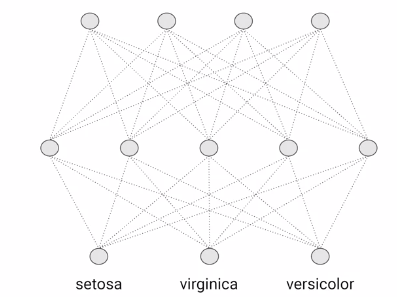








So for the irs case well designed it like this:



And in code it will look like this:

const model = tf.sequential();

model.add(tf.layers.dense({

inputShape: [numOfFeatures],

activation: “sigmoid”, units: 5}))

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

p1-p3:

p2 🡪 hidden layers with five neurons

p3 🡪 3 neurons at the bottom activating them with softmax function

compile:

model.compile({

loss: “categoricalCrossentropy”,

optimizer: tf.train.adam(0.06)}

training:

await model.fitDataset(

convertedData,

{

Epochs:100,

Callbacks:{

onEpochEnd: async(epoch, logs) =>{

console.log(“E: “ + epoch + “ Loss: “ + logs.loss);

}

}

});

Predict:

Const testVal = tf.tensor2d([5.8, 2.7, 5.1, 1.9], [1, 4]);

Const prediction = model.predict(testVal);

Alert(prediction);

Week 2

This wekk we’ll take a look at some training convolutional neural networks for image classification in the browser and then writing abrowser app that takes these images and passes them to the classifier, we’ll start by creating the model using java script like this:

model = tf.sequential();

model.add(tf.layers.conv2d({inputSHape: [28, 28, 1], kernelSie: 3, filters: 8, activation: ‘relu’}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16, kernelSize: 3, activation: “relu}));

model.add(tf.layers.maxPooling2d({poolSie:[2, 2]}));

model.add(tf.layers.flattern());

model.add(tf.layers.dense({units: 128, activation: ‘relu’}));

model.add(tf.layers.dense({units: 10, activation: ‘softmax’}));

r2 🡪 kernel size = 3 we are specifying that we want to use three by three filters, and for filters = 8 mean we have a set of 8 filters that we will that attempt to learn convolutions from.

Compile:

model.compile({ optimizer: tf.train.adam(), loss: ‘categoricalCrossentropy’, metrics: [‘accuracy’]});

Training:

Model.fit(trainXs, trainYs, { batchSize: BATCH\_SIZE, validationData: [testXs, testYS], epochs: 20, shuffle: trie, callbacks: fitCallbacks});

In tensorflow.js there is a cool library called tf-vis that we ca use too render the outpurs of your callbacks. With this code!:

We must include the library called tfjis-vis in our code with this script

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

Note here’s the source link: <https://github.com/tensorflow/tfjs/tree/master/tfjs-vis>

To use the tf-vis libraries with fitcallbacks (from before code), we must declare it to be return tfvis.show.fitCallbacks:

const metrics = [‘loss’, ‘val\_loss’, ‘acc’, ‘val\_acc’];

const container = {name: ‘Model Training’, styles: { height: ‘1000px’ } };

const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);

Sprite sheet for training in web

Export class MnistData {

…

Async load() {

}

nextTrainBatch(){

}

nextTestBatch(){

}

}

In order to initialize the data class and load the sprite getting it ready for batching we need this code:

Const data = new MnistData();

Await data.load();

Once we loaded instance of data we can now get the batches and resize them

Const [trainXs, trainYs] = tf.tidy(() =>{

Const d = data.nextTrainBatch(TRAIN\_DAtA\_SIZE);

Return [

d.xs.reshape([TRAIN\_DATA\_SIZE, 28, 28, 1]),

d.labels

];

});

r1 🡪 we want to create an array containing the set of training Xs and training Ys, so this function will handle that.

r2 🡪 getting the next training batch from the data source, by default with MNIST, the train data size is 5500 so its basically getting 5500 lines of 784 bites

r4 🡪 reshape the data into a four dimensional tensor with 5500 in the first dimension then 28 by 28 representing the image, and the one representing the color depth

r5 🡪 as the labels are already one hot encoded, it will return them as the second element in the array

tf.tidy? something that helps our code be a good citizen within the browser TensorFlow apps, by their nature, tend to use a lot of memory. So the idea of tf.tidy is that one the execution is done it cleans up all those intermediate tensores, except those that it returns