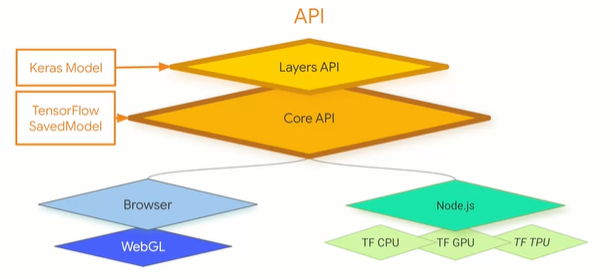
Browser-based Models with TensorFlow.js

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,

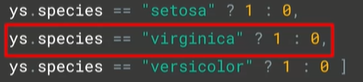
ys.species == “versicolor” ? 1 : 0 ]

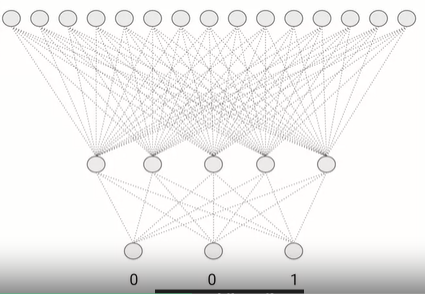
return{ xs: Object.values(xs), ys:Object.values(labels)};

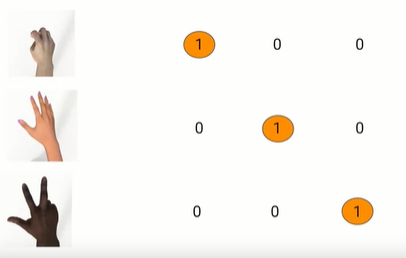
}).batch(10);

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:

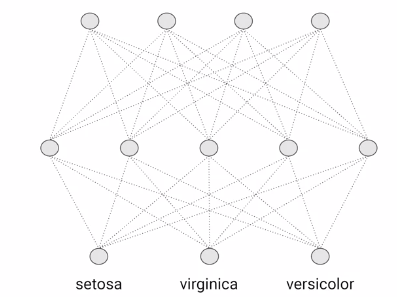








So for the irs case well designed it like this:



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

Week 3

Training in the browser requires some kind of compromise, in this week instead of training models we will use existing of the shelf ones in the browser.

Pre trained tf.js models: <https://github.com/tensorflow/tfjs-models>

So how do we get started?

Add this script first:

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

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

How de we search the model? Its kinda hard so the rule of thumb that Lawrence would recommend is to take a look at the URL like this and then just take the name of the model at the end

So the code will be

<html>

<head>

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

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

<script>

// our code

</script>

</head?

<body></body>

</html>

Now the first thing that we need when using toxicity is threshold:

const threshold = 0.9;

#this value is the minimum prediction confidence namely, if a prediction comes in as over this value, we will match it

How to do a prediction on a sentence:

Toxicity.load(threshold).then(model => {

Const sentences = [‘you suck!’];

Model.classify(sentences).then(prediction =>{

// Handle Results

});

});

r1 🡪 load the model, passing it the threshold value. Then once it loaded, we’ll have a model

r2 🡪 we’ll create an array of sentences to classify

r3 🡪 we’ll then call model.classify passing it the sentences and the we’ll get a set of prediction

MobileNet 🡪 small latency, low-power model, parameteried to meet the resource constraints of a variety of uses cases.

Mobilenet label: <https://github.com/https-deeplearning-ai/tensorflow-2-public/blob/main/C1_Browser-based-TF-JS/Misc/labels.txt>

Mobilenet script:

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

<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0"> </script>

Body need an image tag and a div to contain the output text:

<body>

<!-- Feel free to replace the quotes in src with these

"/image/coffee.jpg"

"/image/cat.jpg"

"/image/pig.jpg"

-->

<img id="img" src="/image/pig.jpg" style="height:700px"></img>

<div id="output" style="font-family:courier;font-size:24px;height:300px"></div>

</body>

Script that passes the image to mobilenet and gets a set of classification back

const img = document.getElementById('img');

const outp = document.getElementById('output');

mobilenet.load().then(model => {

model.classify(img).then(predictions => {

console.log(predictions);

for(var i = 0; i<predictions.length; i++){

outp.innerHTML += "<br/>" + predictions[i].className + " : " + predictions[i].probability;

}

});

});

r1 🡪 create a variable representing the image tag

r2 🡪 load the mobilenet

r3 🡪 to use it we pass the image as a parameter to the models classisfy function and prediction

Converting models (python) to javascript

Install this code first:

!pip install tensorflowjs

Next we’ll create this:

import numpy as np

import tensorflow as tf

from tensorflow import keras

print(tf.\_\_version\_\_)

model = tf.keras.Sequential([keras.layers.Desne(units=1, input\_shape=[1])])

model.compile(optimizer = ‘sqg’, loss = ‘mean\_squared\_error’)

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=floar)

ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0 ,7.0, dtype=float)

model.fit(xs, ys, epochs =500)

#predict

print(model.predict([10.0]))

After we have that we need to save our model, we’re going to do it as a save model:

import time

saved\_model\_path = ‘/tmp/saved\_models/{}’.format(int(time.time()))

#for tf 2.0

#tf.keras.experimental.export\_saved\_model(model, saved\_model\_path)

#for tf 1.

#tf.contrib.saved\_model.save\_keras\_model(model, saved\_model\_path)

- Generating directory to save the file in using the timestamp, so we’ll import time, get the current timestamp and save the model and the path

Output is the direccotry and we need that directory

Here’s the command to convert a saved model into the tf.js and its called model.json

!tensorflowjs\_coverter \

--input\_format=keras\_saved\_model \

/tmp/saved\_models/1554528640/1554528642 \

/tmp/linear

R2🡪directory contained the save model

R3 🡪 output directory where we want the json to be saved

Weneed to download all the files and put them in the same directory as the HTML page that will host them

Now the html code where the model hosted in it

<html>

<head>

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

<script>

async function run(){

const MODEL\_URL = 'http://127.0.0.1:8887/model.json';

const model = await tf.loadLayersModel(MODEL\_URL);

console.log(model.summary());

const input = tf.tensor2d([10.0], [1,1]);

const result = model.predict(input);

alert(result)

}

run();

</script>

</head>

<body></body>

</html>

MODEL\_URL need to be the same address as the mosel.json

Week 4

In this week, we’re going to get an existing pre-trained model mobile net and you’ll freeze some of its layers training a new neural network using the features that you’ve extracted from the mobile net. You’ll capture images in the browser using the webcam, sort these into your desired classes. Adn then with transfer learning, you’ll build a new model that classifies only those images. So let’s get started.

Build a simple web page

<html>

<head>

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

<script src="webcam.js"></script>

</head>

<body>

<div>

<div>

<video autoplay playsinline muted id="wc" width="224" height="224"></video>

</div>

</div>

</body>

<script src="index.js"></script>

</html>

This page will render a live stream of the webcam, it will also initialize everything you need to start capturing from the webcam and converting that data into tensors, which will then be used to train the network

-The video tag defines aa video area on the page, and it’s given the id wc

-index.js is where we’ll write most of our code

How we start the index.js file:

let mobilenet;

let model;

const webcam = new Webcam(document.getElementById(‘wc’));

async function init(){

await webcam.setup();

}

init();

next step add this in index.js:

async function loadMobilenet() {

const mobilenet = await

tf.loadLayerModel(‘mobilenet link’);

const layer = mobilenet.getLayer(‘conv\_pw\_13\_relu’);

return tf.model((input: mobilenet.inputs, outputs: layer.output));

}

- the const layer we cant get one of the output layers from the preloaded mobilenet, in here we’re selecting the layer called conv pw 13 relu as the one above everything will freeze.

-we’ll then use the tf.model class to make a new model and its constructor can tak inputs and outputs, which we will set to take the mobilenet inputs namely the top of the mobilenet and the conv pw 13 relu as output

In order to make this work, we’ll update our init function with this code:

async function init(){

await webcam setup();

mobilenet = awit loadMobilenet();

tf.tidy(() => mobilenet.predict(webcam.capture()));

}

Instead of adding a new densely connected set of layers underneath the frozen layers from the original model, we will create a new model. With its input shape being the output shape of the desired mobile net layer. We then treat this as a separate model that we train. At prediction time, we’ll then get a prediction from our truncated mobile net up to the layer that we wanted to gice us a set of embeddings. We’ll then pass those embeddings through the new model in order to get a prediction that the new model was trained on. Code:

async function train() {

model = tf.sequential({

layers: [

tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),

tf.layers.dense({ units: 100, activation: ‘relu’}),

tf.layers.dense({ units: 3, activation: ‘softmac’})

]

});

}

const embeddings = mobilenet.predict(img);

const predictions = model.predict(embedding);

code that we need to capture the data that will be used to retrain the network:

<button type="button" id="0" onclick="handleButton(this)" >Rock</button>

<button type="button" id="1" onclick="handleButton(this)" >Paper</button>

<button type="button" id="2" onclick="handleButton(this)" >Scissors</button>

<div id="rocksamples">Rock Samples:</div>

<div id="papersamples">Paper Samples:</div>

<div id="scissorssamples">Scissors Samples:</div>

<button type="button" id="train" onclick="doTraining()" >Train

Handlebutton works:

function handleButton(elem){

switch(elem.id){

case "0":

rockSamples++;

document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;

break;

case "1":

paperSamples++;

document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;

break;

case "2":

scissorsSamples++;

document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;

break;

}

label = parseInt(elem.id);

const img = webcam.capture();

dataset.addExample(mobilenet.predict(img), label);

}

Data set file in:

<script src="rps-dataset.js"></script>

And here’s the rpsdataset in rps dataset.js:

class RPSDataset {

constructor() {

this.labels = []

}

addExample(example, label) {

if (this.xs == null) {

this.xs = tf.keep(example);

this.labels.push(label);

} else {

const oldX = this.xs;

this.xs = tf.keep(oldX.concat(example, 0));

this.labels.push(label);

oldX.dispose();

}

}

encodeLabels(numClasses) {

for (var i = 0; i < this.labels.length; i++) {

if (this.ys == null) {

this.ys = tf.keep(tf.tidy(

() => {return tf.oneHot(

tf.tensor1d([this.labels[i]]).toInt(), numClasses)}));

} else {

const y = tf.tidy(

() => {return tf.oneHot(

tf.tensor1d([this.labels[i]]).toInt(), numClasses)});

const oldY = this.ys;

this.ys = tf.keep(oldY.concat(y, 0));

oldY.dispose();

y.dispose();

}

}

}

}

Training the network:

async function train() {

dataset.ys = null;

dataset.encodeLabels(3);

model = tf.sequential({

layers: [

tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),

tf.layers.dense({ units: 100, activation: 'relu'}),

tf.layers.dense({ units: 3, activation: 'softmax'})

]

});

const optimizer = tf.train.adam(0.0001);

model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});

let loss = 0;

model.fit(dataset.xs, dataset.ys, {

epochs: 10,

callbacks: {

onBatchEnd: async (batch, logs) => {

loss = logs.loss.toFixed(5);

console.log('LOSS: ' + loss);

}

}

});

}

Nexr we’ll se how to poll frames from the webcam and pass the to the model for inference to see if the models sees rock paper or scissors.

First edit:

<div id="dummy">Once training is complete, click 'Start Predicting' to see predictions, and 'Stop Predicting' to end</div>

<button type="button" id="startPredicting" onclick="startPredicting()" >Start Predicting</button>

<button type="button" id="stopPredicting" onclick="stopPredicting()" >Stop Predicting</button>

<div id="prediction"></div>

Start and stop predicting:

function startPredicting(){

isPredicting = true;

predict();

}

function stopPredicting(){

isPredicting = false;

predict();

}

Code to rea a frame from the webcam

const predictedClass = tf.tidy(() => {

const img = webcam.capture();

const activation = mobilenet.predict(img);

const predictions = model.predict(activation);

return predictions.as1D().argMax();

});

Update the ui:

const classId = (await predictedClass.data())[0];

var predictionText = "";

switch(classId){

case 0:

predictionText = "I see Rock";

break;

case 1:

predictionText = "I see Paper";

break;

case 2:

predictionText = "I see Scissors";

break;

}

document.getElementById("prediction").innerText = predictionText;

tidy up using:

predictedClass.dispose();

await tf.nextFrame();