---

layout: post title: "Basic Tutorial with TensorFlow.js: Linear Regression" image: assets/images/tensorflow-linear.gif image_alt: "Stylized TensorFlow.js Logo" published: true

Fresh off of the OpenAI Retro contest, I wanted to keep exploring more AI topics. Somebody told me that the best way to learn was reproducing other people's papers. Not wanting to learn any more Python than I had to, I decided to try to tackle some existing work with TensorFlow.js.

I first tried to run, but I realized it might be better to crawl first since I am coming from a pretty fresh background. I was able to find a series of basic TensorFlow examples that I felt would let me ladder up my TensorFlow.js skills in aymericdamien/TensorFlow-ExamplesTODO link to past blog posts?

Hello World

I'm not sure what the right Hello World with TensorFlow would be, so I printed out my first 1 dimensional Tensor:

```
const tf = require('@tensorflow/tfjs');
require('@tensorflow/tfjs-node');

tf.scalar(3.14).print();

// Tensor
//     3.140000104904175
```

and hey, looks like floating point percision problems exist in the TensorFlow world too!

Basic Operations

Getting a handle on the basic operations seems important! On of the biggest surprises here was that methods to display the values inside of a Tensor seem somewhat limited. `tf.print()` and `tf.toString()` both add the word `Tensor` with a newline to the text, making it a bit hard to easily display values.

```
let a = tf.scalar(2);
let b = tf.scalar(3);

console.log('a: '+a.dataSync()[0]+', b: '+b.dataSync()[0]);
console.log('Addition with constants:'+ a.add(b).dataSync()[0]);
console.log('Multiplication with constants:'+ a.mul(b).dataSync()[0]);

// a: 2, b: 3
// Addition with constants:5
// Multiplication with constants:6
```

Basic Operations with a model

This was the same as above, but in Python using a placeholder input and feeding in values when the session ran. With the idiomatic differences between Python TensorFlow and TensorFlow.js, I couldn't see a useful way to do this besides creating functions that took tensors as input.

Matrix Mulitplication

Again, another basic Tensor operation, the dot product.

```
let a = tf.tensor2d([[3, 3]]);
let b = tf.tensor2d([[2], [2]]);

a.matMul(b).print();
//Tensor [[12],]
```

Linear Regression

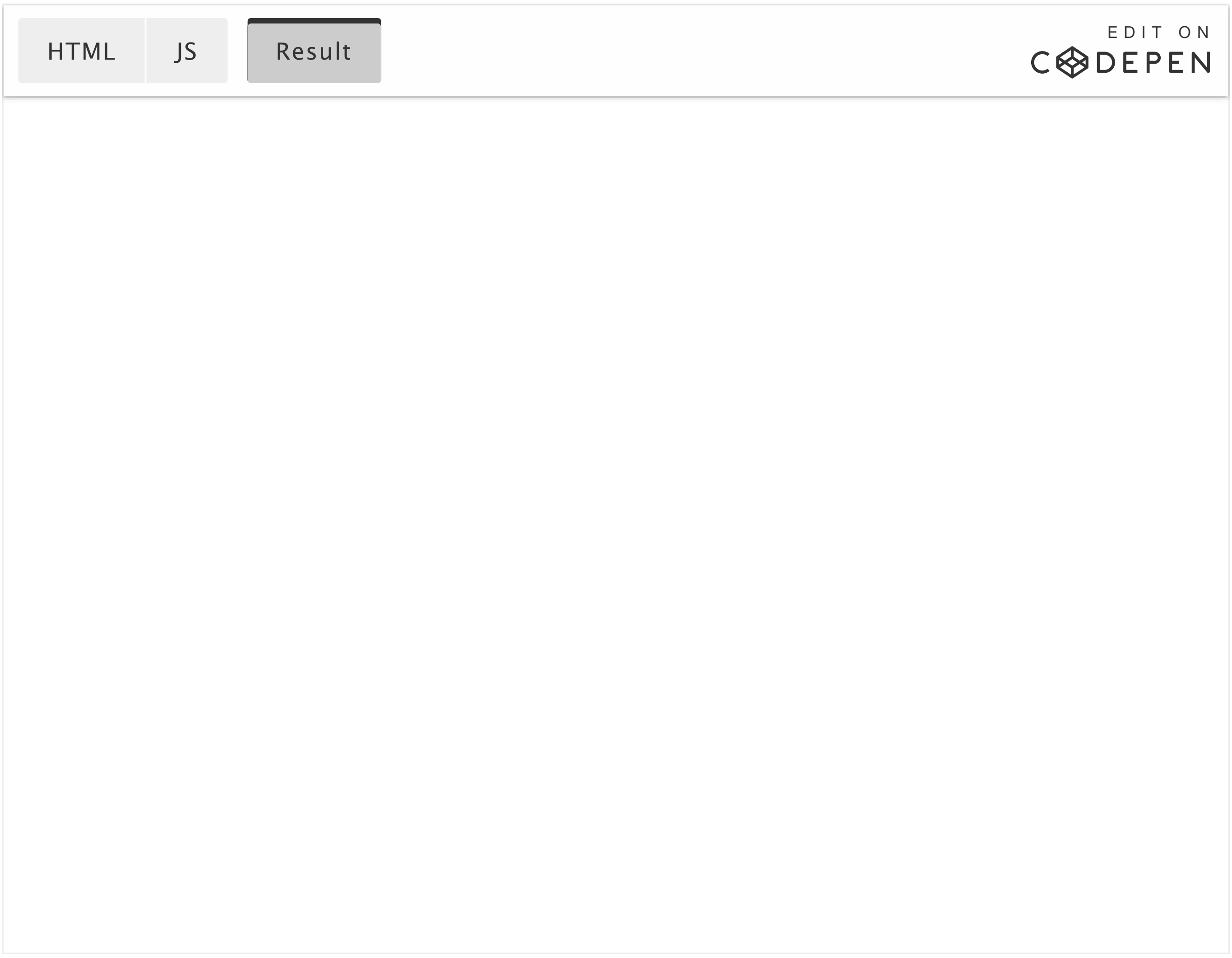
Now it's time to start making a model that can actually do things. One of the simplest problems to solve is finding the values for a linear regression. If you recall from algebra, lines generally can be modeled in the x/y space with `y = mx + b`, `m` being the slope of the line and `b` a scalar offset in the y direction. TensorFlow can do the hard work of finding out the best `m` & `b` are for a given set of data.

We'll start our code out with our training data and initializing variables that will hold our results. The `m` & `b` variables are 1 dimensional tensors with only one value, so we can use the helper function `tf.scalar()` when creating them.

```
const trainX = [3.3, 4.4, 5.5, 6.71, 6.93, 4.168, 9.779, 6.182, 7.59,
2.167, 7.042, 10.791, 5.313, 7.997, 5.654, 9.27, 3.1];
const trainY = [1.7, 2.76, 2.09, 3.19, 1.694, 1.573, 3.366, 2.596, 2.53,
1.221, 2.827, 3.465, 1.65, 2.904, 2.42, 2.94, 1.3];

const m = tf.variable(tf.scalar(Math.random()));
const b = tf.variable(tf.scalar(Math.random()));
```

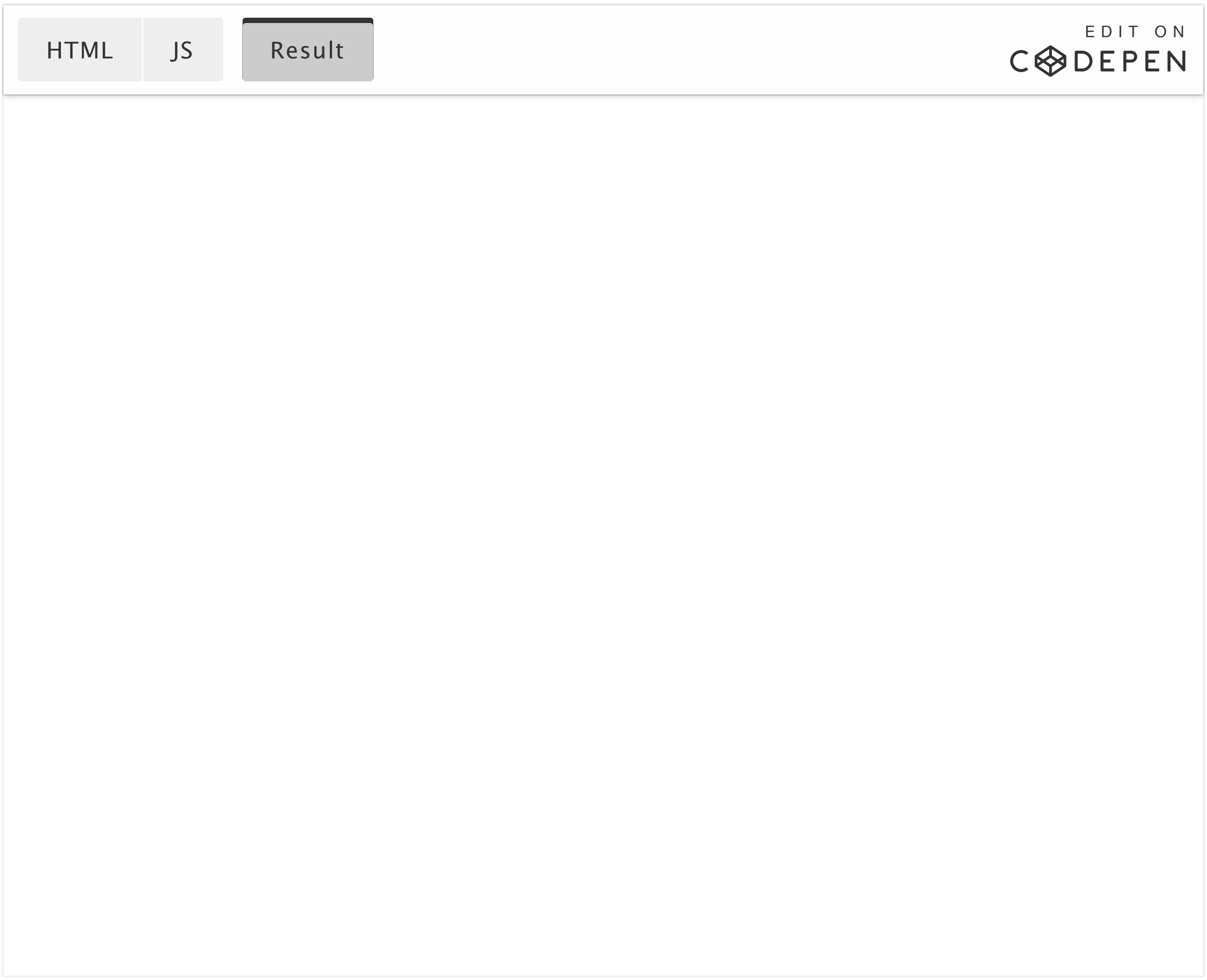
Right now we can visualize our data, but because we only have the randomly initialized numbers to define our line, we don't have a very good fit.



Even though we know that we don't have a good model, we can use it to predict the Y values of the points. We'll create a new function (`predict()`) to efficiently run the model. `predict()` takes a tensor of x values in as an argument, then multiplies them by `m` and adds them to `b`, returning the result. `tf.tidy()` is a helper function to make sure that any intermediate Tensors are effectively garbage collected.

```
function predict(x) {
  return tf.tidy(function() {
    return m.mul(x).add(b);
  });
}
```

As you would expect, all of the points are on the line that describes the slope (`m`) and y-offset (`b`).



Let's see if we can actually get a good fit. We are still missing a couple pieces of the puzzle to do that though. One is the loss function. The loss function will be what our model uses to determine how well the model is performing. I think it could just as easily be described in the positive and be called a fit or ***goodness*** function but loss seems to be what the data science community has picked.

Our loss function will use the mean squared error of all of the model's predicted Y values, and the Y values that our data points have. Our loss function takes in Tensors of both the predicted and actual Y values and then outputs a single value describing how far off the predicted values are.

```
function loss(prediction, actualValues) {
  // Having a good error function is key for training a machine learning
model
  const error = prediction.sub(actualValues).square().mean();
  return error;
}
```

We will also need an optimizer. TensorFlow.js has one built in that we will use called Stochastic Gradient Descent. Google has a good explanation of what that means, but this bit of code will look at the loss, and choose new values of `m` & `b` to us in the model that will hopefully be better fits. It takes a single parameter, the learning rate. The higher the learning rate, the faster the model will reach better solutions, but also the greater the risk that the learning rate is too high and your model will run out of control. Definitely check out the explanation above if you haven't seen this before.

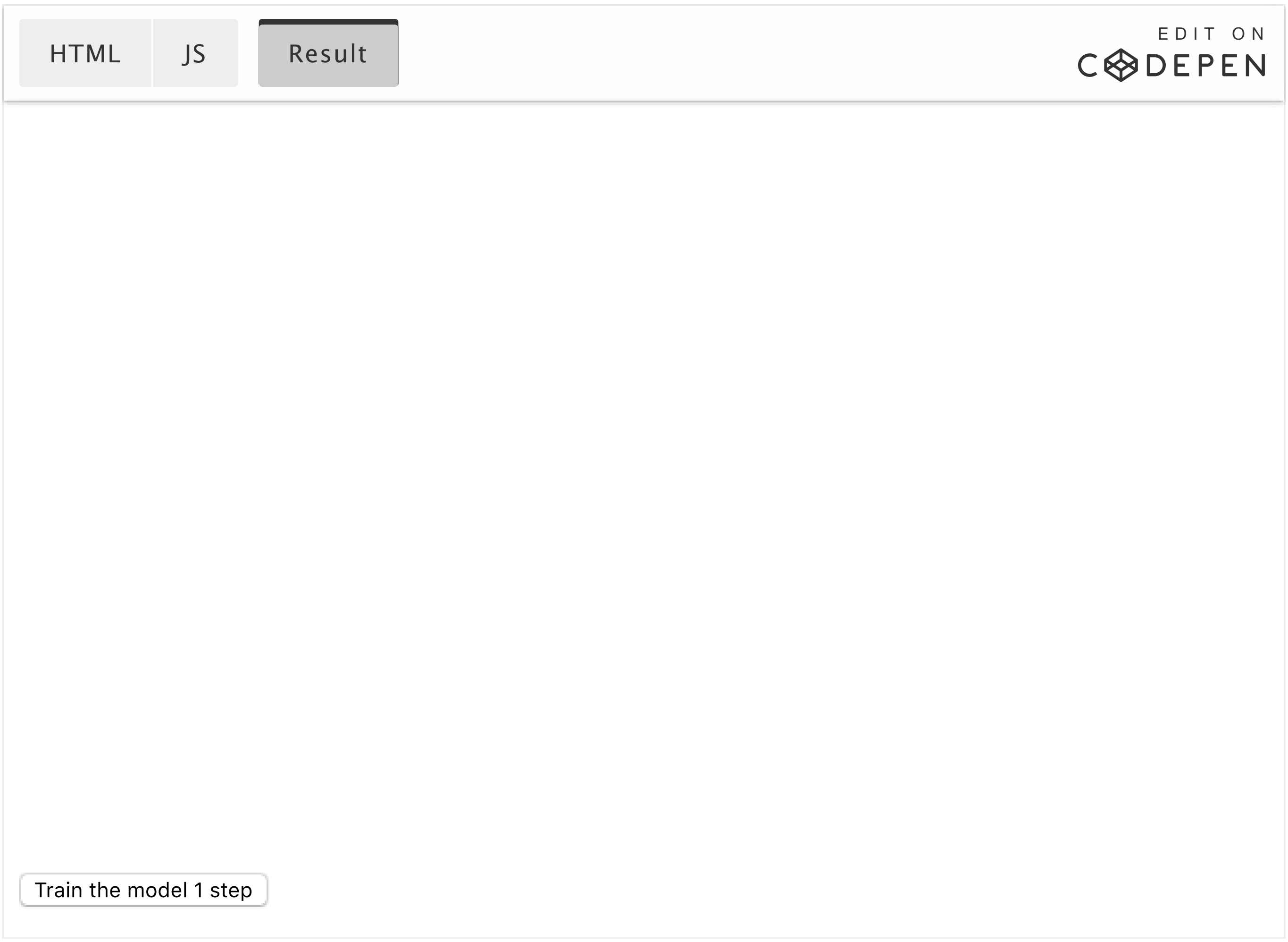
```
const learningRate = 0.01;
const optimizer = tf.train.sgd(learningRate);
```

I'll create a new function to train the model. My goal is to have this function run on the click of a button and to update the graph with each newly updated model. It predicts Y values using the model (as defined in `predict()`) and then calculates the loss, returning it back to the optimizer so that the optimizer can better choose new model values. It also runs `plot()`, a function that updates the chart.

```
function train() {

    optimizer.minimize(function()  {
      const predsYs = predict(tf.tensor1d(trainX));
      stepLoss = loss(predsYs, tf.tensor1d(trainY))
      console.log(stepLoss.dataSync()[0])
      return stepLoss;
    });
    plot();
}
```

Depending on what learning rate you use, you should be able to get to a very good solution in just a few iterations. Try it out in the pen below!



I hope you enjoyed this tutorial with TensorFlow.js! You can find the complete code in all of the codepens, as well as in this gist. If you are looking for more TensorFlow.js content, check out the official curve fitting example & follow me for the next thing I'll write.

TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.