LAB 4.1

Beskrivelse

4.1 Prepare the Data

In this lab you will import and prepare data ready for training a linear regression model with TensorFlow.js.

First, create a HTML page with TensorFlow.js imported as we did in the last section.

Now try each of the following exercises, referring to the TensorFlow.js API docs as a reference.

Get the data

1. Download the **King County House** Sales data set
    - You can get it from Kaggle but you will first need to sign up for a (free) account.
    - Alternatively, download the copy of the data attached as a resource for this lesson.
2. Unzip the data to find the CSV file `kc_house_data.csv`.
3. Open `kc_house_data.csv` in your editor (eg. Atom) and familiarise yourself with the columns.

Set up the environment

If you are adapting this lab to Node.js, skip this part. You can run this lab as a command line program.

1. Create a HTML page with TensorFlow.js imported as we have done previously, and save it as `linear.html`.
2. Rather than open it directly in your browser (which uses the `file://` protocol), we need to serve it over

Oplysninger

Udgivet 24. februar 2026 af Morten Sabro Damgaard

Deadline: 26. februar 2026 23:59

Denne arbejdsopgave er obligatorisk

HTTP (so that we can later load the CSV over HTTP).

- If you already have a web server installed (such as Apache), you could use that.
- If you don't have a web server installed, you can set up a basic server by installing a node package:

  a. On the command line, run `npm install http-server -g` (or `sudo npm install http-server -g` if necessary)
  b. Navigate to the folder where you saved `linear.html` using the `cd` command
  c. Run `http-server` and the web server will start
  d. Open one of the `http://` addresses listed.
  e. You will see a directory listing. Click on `linear.html`.
  f. When you later want to stop the web server, press **control-C** on your keyboard when the command line is active.

3. Ensure you are able to view `linear.html` in your browser over the HTTP protocol. The address bar should look similar to this: `http://127.0.0.1:8080/linear.html`.

Import from CSV

1. Place `kc_house_data.csv` in the same folder as `linear.html`
2. In the script tag, create an `async` function named `run` and call the function below with `run()`. You will add code inside the function so that you can use the `await` keyword.
3. The first line should `await tf.ready()`. tf.ready() ensures that the TensorFlow.js backend has been

initialized. Generally, this happens quickly and you could do without this line, but it's safest to use it just in-case TensorFlow.js takes longer than usual to initialize.

4. Import the data with `const houseSalesDataset = tf.data.csv('./kc_house_data.csv')`.

   - *If you are adapting this lab to Node.js, you can import the CSV using the `file:///` protocol. You would need to specify the full path so can use `` `file:///${__dirname}/kc_house_data.csv` `` to find a file in the same directory as the current source code file.*

5. You may try out the methods available on the tf.data.CSVDataset object.

Visualise the data

If you are adapting this lab to Node.js, you cannot visualise the data with tfjs-vis. Alternatively, you could write a function to export an scatter plot image with another library such as chartjs-node

1. Add a script tag to `linear.html` to import tfjs-vis: `<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-vis"></script>`
2. Add the following snippet to extract x and y points from the dataset:

   a. const pointsDataset = houseSalesDataset.map(record => ({ x: record.sqft_living, y: record.price }));
   b. const points = await pointsDataset.toArray();

3. Add the following function to visualise the points:

   a. function plot (points, featureName) {
   b.   tfvis.render.scatterplot(
   c.   {name: `${featureName} vs House Price`},

   d. {values: [points], series: ['original']},
   e. {
   f. xLabel: featureName,
   g. yLabel: 'Price',
   h. }
   i.);
   j. }

4. Call the function:

   a. plot(points, 'Square Feet');

5. Refresh linear.html in the browser and observe the scatter plot. You will see there is a correlation.

Extra challenge

Try out different x values instead of sqft_living. Can you find other correlated x values? If you like, you can use that instead of sqft_living throughout the labs.

Extract features and labels

The x values will be our **feature** (**input**). The y values will be out **label** or **output**.

1. Extract the feature and store it in a tensor:

   a. const featureValues = points.map(p => p.x);
   b. const featureTensor = tf.tensor2d(featureValues, [featureValues.length, 1]);

2. Extract the label and store it in a tensor:

   a. const labelValues = points.map(p => p.y);
   b. const labelTensor = tf.tensor2d(labelValues, [labelValues.length, 1]);

Extra challenge

The documentation advises against using Dataset.toArray(). We have done so to demonstrate manual tensor creation and because this data set is reasonably small. How else can you create these Tensors? Try doing so with the Dataset.batch(...)

and Dataset.forEachAsync(...) methods. You will need a good understanding of asyncronous methods in order to manage this!

Normalise features and labels

1. Add the following function for **min-max normalisation**:
    a. function normalise (tensor) {
    b.   const max = tensor.max();
    c.   const min = tensor.min();
    d.   const normalisedTensor = tensor.sub(min).div(max.sub(min));
    e.   return { tensor: normalisedTensor, min, max };
    f. }
2. Normalise both the features and labels:
    a. const normalisedFeature = normalise(featureTensor);
    b. const normalisedLabel = normalise(labelTensor);
3. Add this denormalisation function for later use:
    a. function denormalise (tensor, min, max) {
    b.   return tensor.mul(max.sub(min)).add(min);
    c. }

Shuffle the dataset

Before splitting data into sets, we should shuffle the dataset to ensure a random order. We need to shuffle both features and labels together, so it's best to do this earlier.

1. Near the top of your code after you created the points array, use tf.util.shuffle() to shuffle the array.

Split into testing and training sets

We need to split into two sets: one for training, another for testing.

1. Split the dataset:

   a. const [trainingFeatures, testingFeatures] = tf.split(normalisedFeature.tensor, 2);
   b. const [trainingLabels, testingLabels] = tf.split(normalisedLabel.tensor, 2);
   c.

Extra challenge

The above splits into 2 evenly sized sets (for both features and labels). Check the documentation for tf.split(...), and adjust to use 80% for training and 20% for testing.

Clean up memory usage

1. Use tf.memory() to check memory usage at the end of the code.
2. Wrap all the code with tf.tidy() and check memory usage again to ensure tensors are removed from memory.

We're done! You've imported, visualised, and prepared data for machine learning!

In the next lab, we'll create a linear regression model to train.