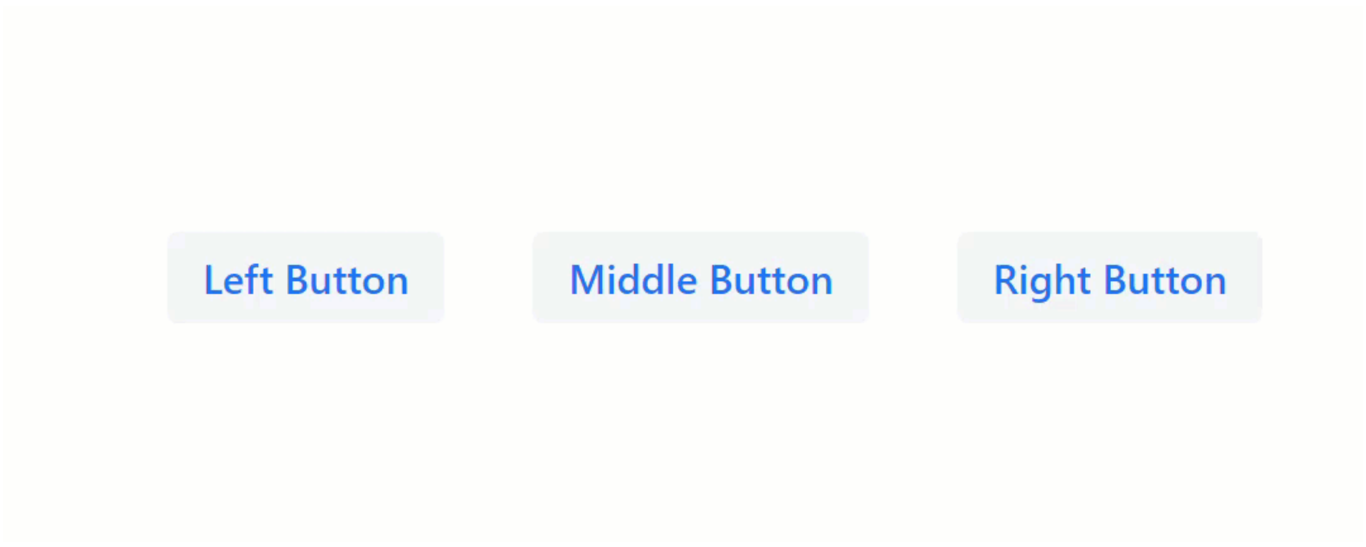
---

title: Improving User Experience with TensorFlow.js published: false description: tags: tensorflow.js, ux, ml

Improving User Experience with TensorFlow.js

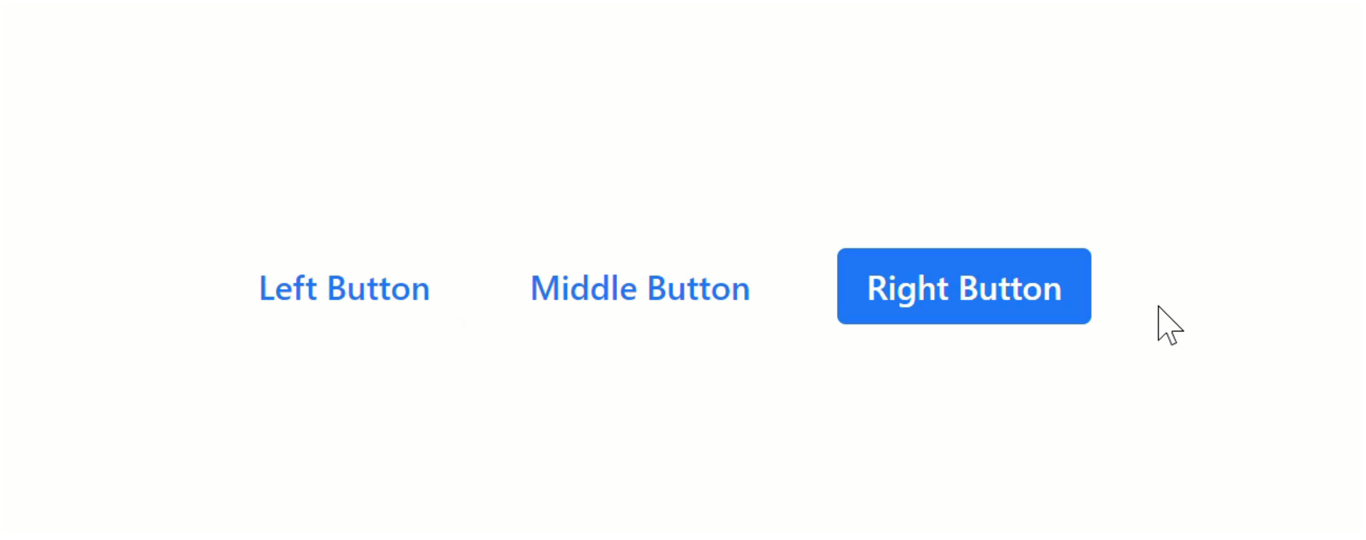
Imagine you have an application with some buttons or other interactive elements. To make clear which button you would click they often highlight when you hover over them. Much like this



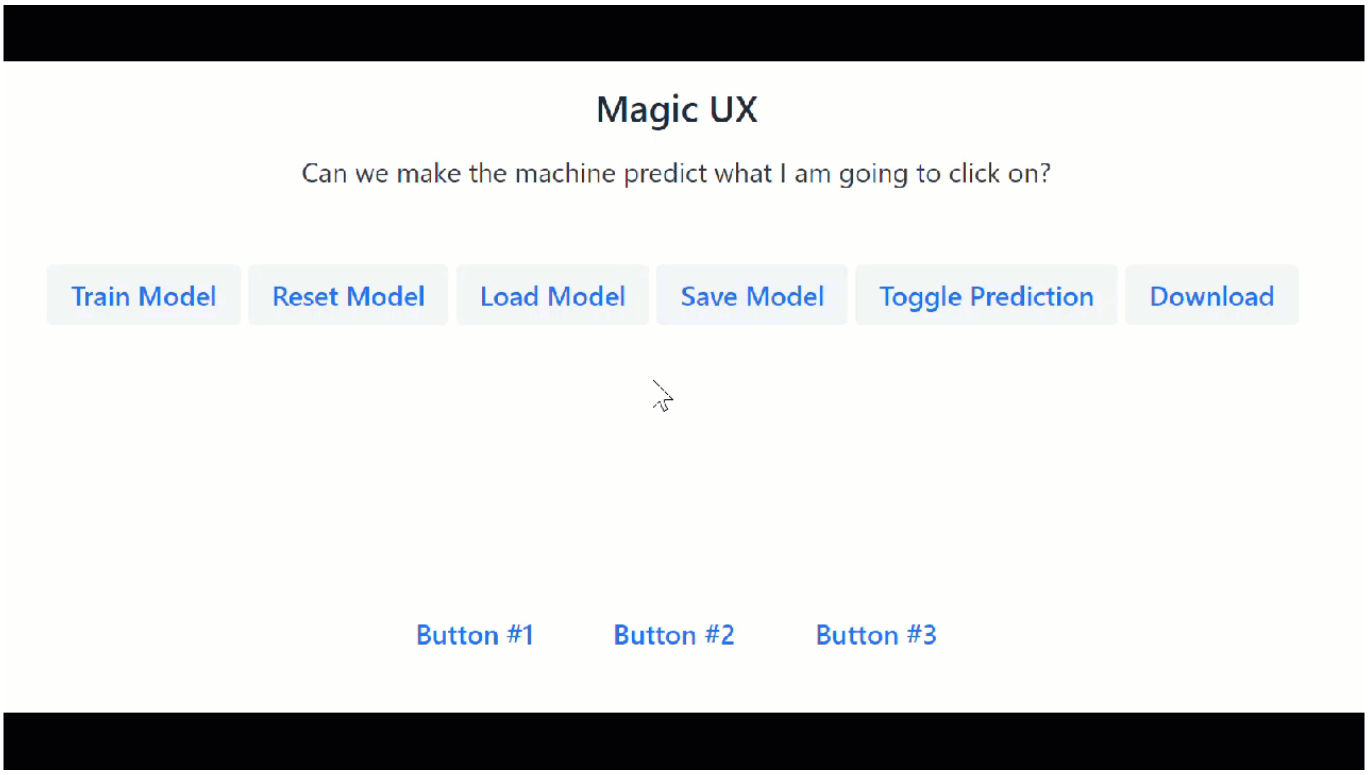
Now imagine we would be able to highlight the interactive element you are about to click *before* you hover over it. This might be useful for at least two reasons

1. Prepare resources in advance
    - toggle lazy loading of images or iframes to eager earlier (https://dev.to/ben/native-lazy-loading-for-img-and-iframe-is-coming-to-the-web-55on)
    - Serverless function pre-start
    - Module pre-load
    - AR: where would people look at soon? render more details
2. (unnoticeably) highlight button to make it easier to access for user, thus giving them a better user experience without them necessarily knowing why

Such a highlighting system could look like this. We did not make the highlight very subtle, to make clear what is going on:



This is just one way of predicting the button, but there are ways of prediction that take the path you are moving on into account much more, like this one



This way the system would have a solid prediction earlier than the first system, however at the cost of less certainty.

But how can this be done?

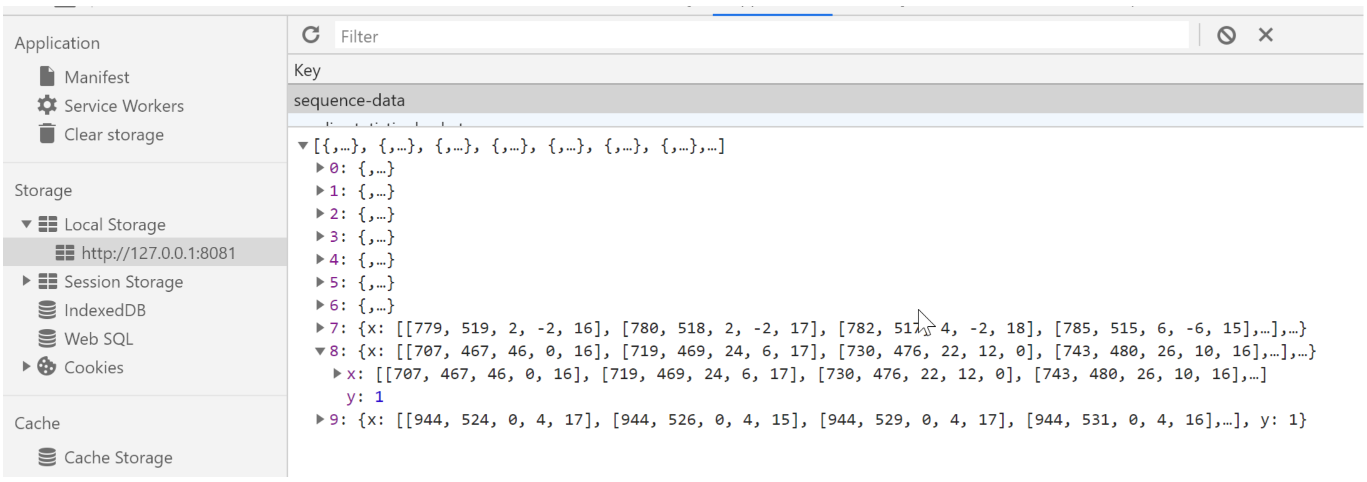
I guess there could be heuristic that either highlights a button when we are close to it and at the same time closer than to any other button or one that does a simple regression drawing a line through your mouse movements and sees what button you are pointing to.

This can be tricky, though, because different computers would have different resolutions, different track pads or mouses. And - more importantly - different types of users would use their devices very differently based on level of experience and personality. I, for example, like track pads with extremely high sensitivity that sometimes make me jump over a button requiring to go back a little bit.

Machine Learning to the rescue

The idea of machine learning is that we collect some information about how a certain user on a specific machine clicks those buttons. Using that data we hope to create a more general model that can predict future uses of that user interface.

In our case we can very easily collect mouse events that give us a sequence of mouse positions and a delta from the previous position. Additionally, we can add a time delta between the events. This effectively gives a us a sequence of five data points over time that you can see in this developer tools screenshot in the browser:



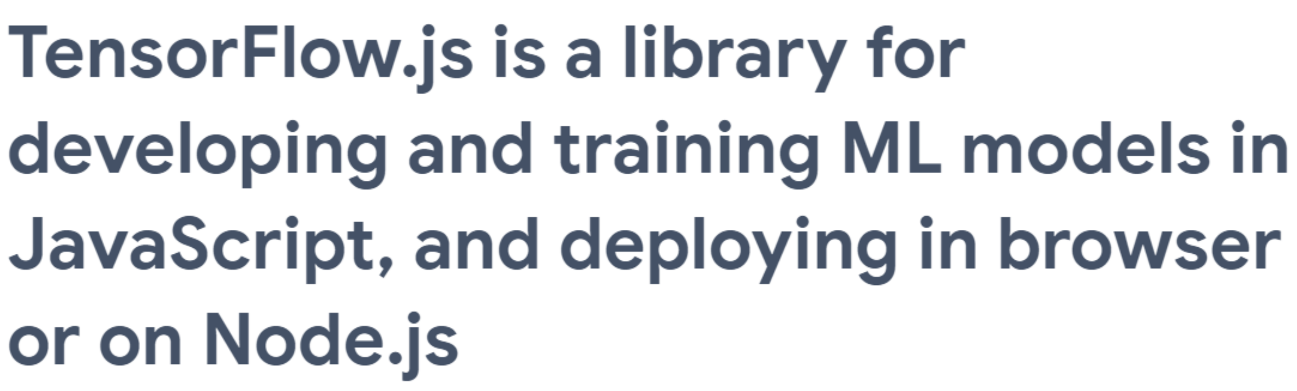
```
▼[{,…}, {,…}, {,…}, {,…}, {,…}, {,…}, {,…},…]
  ▶0: {,…}
  ▶1: {,…}
  ▶2: {,…}
  ▶3: {,…}
  ▶4: {,…}
  ▶5: {,…}
  ▶6: {,…}
  ▶7: {x: [[779, 519, 2, -2, 16], [780, 518, 2, -2, 17], [782, 517, 4, -2, 18], [785, 515, 6, -6, 15],…],…}
  ▼8: {x: [[707, 467, 46, 0, 16], [719, 469, 24, 6, 17], [730, 476, 22, 12, 0], [743, 480, 26, 10, 16],…],…}
    ▶x: [[707, 467, 46, 0, 16], [719, 469, 24, 6, 17], [730, 476, 22, 12, 0], [743, 480, 26, 10, 16],…]
     y: 1
  ▶9: {x: [[944, 524, 0, 4, 17], [944, 526, 0, 4, 15], [944, 529, 0, 4, 17], [944, 531, 0, 4, 16],…], y: 1}
```

We store this sequence under the name of `x` and the a code for the button that has been clicked following such a sequence of mouse movements as `y`. In this example `1` might stand for the first button. That means we have a number of sequences, one for each mouse movement that resulted in a button being pressed, ten in our example.

Privacy

Before we talk about how we will use that data, let us talk about privacy. How would you like every movement of your mouse to leave your computer to create a model of your very personal behavior? While some people arguably would not mind, some - including myself - would. Our solution is to keep the data in your browser, on your disk and train the model right in the browser and also leave that in your browser as a default. So, neither the data nor the trained model are required to ever leave your browser.

TensorFlow.js


TensorFlow.js is a library for developing and training ML models in JavaScript, and deploying in browser or on Node.js

Exactly this is possible using the JavaScript implementation of Google's famous TensorFlow library called TensorFlow.js. We need to define a neural network that matches our data format, then we train that network, and finally we use that network to predict the button being clicked next. This prediction will be based on new, previously unknown data. Even that data can be used to improve our model at any time. All that can happen transparently in the background. The only thing the user might notice is the their machine is getting hot or the battery level going down as training is executed on the users GPU that takes quite some energy.

The actual model

We need to specify a model that is simple and thus fast to train. It also needs to give reasonable results with only very little training data. Even a power user will likely generate less than 100 clicks in a minute, so we should be able to get at least some results with that amount of data.

The most basic type of network layer that can handle sequential data is the vanilla Recurrent Neural Network. As input it will take a number of mouse events (SEGMENT_SIZE) having 5 (N_FEATURES) entries each `(posX, posY, deltaX, deltaY, deltaT)` as explained above. We also need to specify how many neurons we want - the more the more powerful and the more resource hungry - se we specify 50 of them, which is not many.

And here the *input* of the neural network is expressed in terms of TensorFlow.js which will be just readable if you are at familiar with the famous Keras API:

```
input = tf.layers.simpleRNN({
    inputShape: [SEGMENT_SIZE, N_FEATURES],
    units: 50
})
```

As we need to make that network predict one of the buttons the user will click on, we encode each button with a neuron that outputs a likelihood for a button as the *output* of the neural network:

```
output = tf.layers.dense({
    units: 3,
    activation: "softmax"
})
```

In this case we have three buttons and thus three neurons. `softmax` specifies that the neurons spit out a probability that adds up to 1.0.

We now need to combine these two layers in a sequential fashion, i.e. one after the other ...

```
model = tf.sequential()
model.add(input)
model.add(output)
```

... to configure the resulting model. We do this by telling the model how to determine how well it is doing (`loss`), telling it how to improve on that (`optimizer`) and how to report how well it is doing (`metrics`) back to us in a declarative manner. In this case the loss is determined by comparing the button the user has clicked to the prediction of the model. With respect to the output of the model `sparseCategoricalCrossentropy` is a way of expressing that. `adam` is just the default way of having a model optimized and `accuracy` reports the fraction of datasets that would be predicted correctly.

```
model.compile({
    loss: "sparseCategoricalCrossentropy",
    optimizer: "adam",
    metrics: ["accuracy"]
})
```

Training our model with the click data of the user

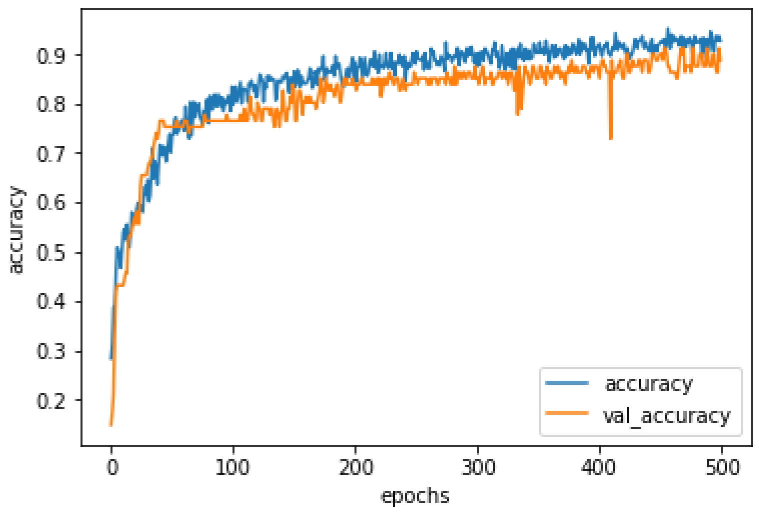
We need a little pre-processing and data coversion that is beyond the scope of this article but that results in all the mouse position sequences being stored in a variable called `X` and all the matching button that end up being clicked in the variable `y`. We pass those data sets through the model 200 times and with each iteration, also called epoch, it hopefully learns how to make model match the data a little better:

```
model.fit(X, y, {
    epochs: 200,
    validationSplit: 0.2,
})
```

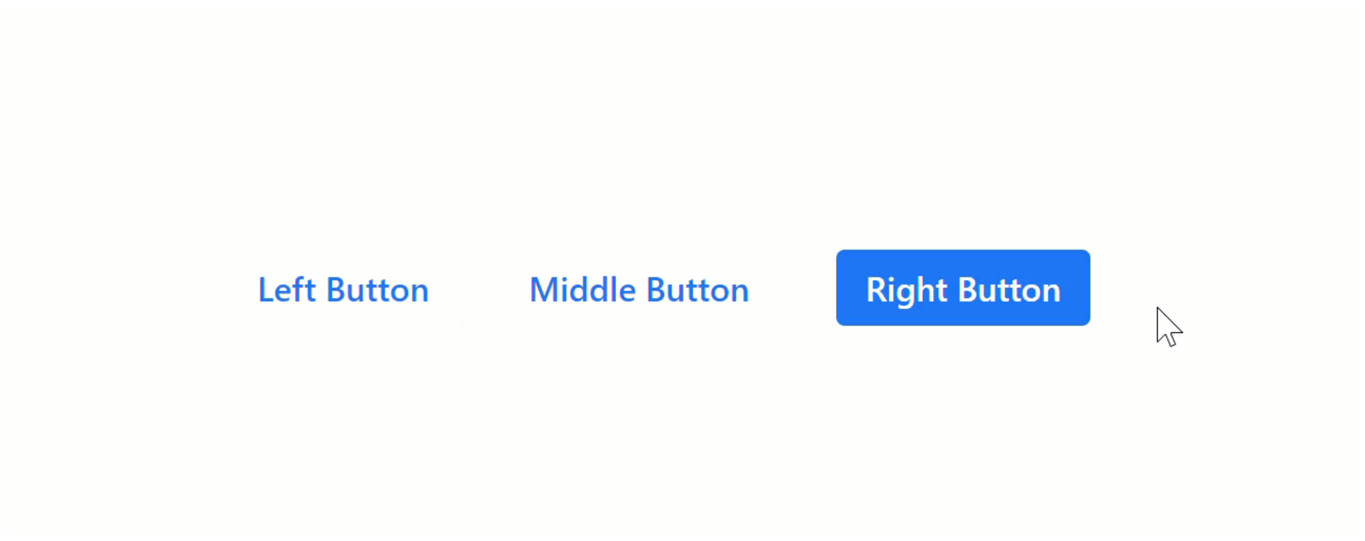
How well is the model actually doing?

There is one more trick, though: since we are really interested in how well the model does on mouse sequences that it has not seen before, we exclude some of the known data - in our case 20% - from being used in the training process. Instead we use it to constantly monitor the performance of the model. Obviously, the model might still perform worse in its real use, but this is the best we can do.

As requested in the metrics TensorFlow.js reports accuracy for training and validation data over time while training:



Indeed we see the rise we were hoping for, both in the training, as well as in the validation data. Depending on how well the data that we trained on represents what the user actually does, this can be quite a successful model. We have already seen how this works in the application in the second image of this article which we repeat here:



Making predictions

We use the model by feeding in actually mouse movements of the user in real time. Next to the privacy concern this is one more thing that only TensorFlow.js allows us to do, as it trains and predicts directly in the browser allowing for the required responsiveness. This is how we make such a prediction:

```
probabilityArray = model.predict(X)
y = probabilityArray.argMax();
```

Calling `predict` on the model gives us the probabilities for all existing buttons. For our three buttons those might be `[0.5087714195251465, 0.31418585777282715, 0.17704263329505920]`. To match the data we were training with originally, we just want to know which of those probabilities is the highest and get back a single number indicating the button to highlight. In this case it would be `0`. You can also take this additional information into account and only highlight a button when you are really sure, in our case we only do it when the probability is over 50%.

Conclusion

For the task at hand only machine learning that is running in the browser was a good fit. Primarily this is because of

- a fit to the individual user and machine
- privacy
- real time responses
- and zero installation

As a nice benefit this also works offline and puts no load on any servers.

This is just one use case for using TensorFlow.js in the browser, but there must be many more and I hope you to come of with one. To get you started have a look at https://www.tensorflow.org/js/tutorials and the complete project I have put on Github https://github.com/DJCordhose/ux-by-tfjs.

Also watch the video https://youtu.be/kZ8sXFIJQyg that explains how to use the demo at http://djcordhose.github.io/ux-by-tfjs/dist that allows you to try this out yourself without any installation directly in the browser (needs a modern browser, though).