Stock Market Predictions Using Tensorflow LSTM

Hunter Miller

Putnam County 4H

Basic Information

All of the files are available at https://github.com/Nerdman42/Nerdman42.github.io, and the working website can be found at https://nerdman42.github.io/. Both links can be found in the README file on the USB drive. Enjoy!

Disclaimer: This project is purely for educational purposes, and should not be trusted to predict actual stocks, especially in today's highly volatile market.

I have always been innately fascinated by artificial intelligence. The idea that humans can create something in our image, with the capacity to learn and the ability to generate creative 'thought', is astounding. Today, AI is one of the fastest-growing industries in the world, finding applications in every faculty of life. Wanting to join the movement, I used this project as my diving board off the deep end, immersing myself in the world of machine learning (ML). Searching for a project that fit my interests, I finally settled on stock market prediction with TensorFlow.js, based on the work of Jinglescode, which can be found at https://github.com/jinglescode/time-series-forecasting-tensorflowjs. My project is completely modeled after that of Jinglescode, but I did everything myself--I just used his work to help me better grasp the interactions between cells, layers, and models.

Like all my previous projects, this started grudgingly slow, with a ridiculously large amount of time being spent just selecting the tools I would use. I started off trying to make the website through repl.it, which failed; I started--and quickly gave up on--making my own graphing software, and I went through three APIs before I found one that I liked. Even once I started coding, the pace was still dragging, with early sessions composed of 90% troubleshooting and 10% progress. But, like a snowball rolling down a hill, I picked up speed as I increased my coding intuition, learning the ins and outs of the project. I've really grown as a coder through this website, sharpening my logical thinking and adding the fundamentals of machine learning to my skills. I can't wait to do more with ML--maybe even with this project. I'd love to develop the model further, using a functional (rather than sequential) model that can take multiple inputs, allowing for the use of technical indicators for a more in-depth analysis. I'd also like to try another ML model, such as a convolutional network, and compare its accuracy to my existing model. Regardless of *what* I do, however, I'll be coding something, and continuing to hone my CS skills.

Components

- Axios is a JavaScript library that functions as an HTTP client, and it can be used in both node.js and the browser. Here, Axios is used to access and return data from YH Finance.
- YH Finance, from RapidAPI, returns financial information about a ticker. While it also has a versatile library, including the ability to give relevant news articles, quarterly breakdowns, and insider transactions, here it is only used to return historic stock price data.
- Chart.js is one of many HTML5 js graphing libraries, but is only one of few that is offered for free. Using the HTML canvas element, Chart.js displays data directly to a webpage, seamlessly creating labels, axes, and a legend. The library also allows for customization, so I also installed the chartjs-plugin-crosshair, which allows for zooming in on the chart.
- Tensorflow.js is a machine learning (ML) library that centers around the use of a datatype, tf.tensors, in ML models.
- The program was compiled using Browserify, allowing for the use of the libraries above. While all of these libraries could have been linked in the HTML code, I had issues getting Chart.js to work, and ended up settling on Browserify instead, which never gave me a hiccup.
- The source code was put online through Github Pages, allowing others to view and learn from my successes and failures.

Axios and YH Finance

Before I could get to the 'fun stuff,' or the ML portion of the project, I had to both import data and make that data into something meaningful. Using Axios, I downloaded historical stock data, parsing through its response to get the data I needed. The timestamps came in seconds since epoch, which I converted into mm/dd/yyyy format. I used a helper function to selectively remove the final date, as it didn't always align with the weekly cycle. I pushed both stock value and timestamp data to their respective arrays to be used for both the chart and calculating the SMA (Simple Moving Average).

```
function initData(ticker, n_steps, freq, trainSize){

  axios.request({//fetches historical stock data
  url: 'https://yh-finance.p.rapidapi.com/stock/v2/get-chart',
  params: {
    interval: freq,
    symbol: ticker,
    range: '10y',
    region: 'US',
  },
  headers: {
    'X-RapidAPI-Key': '8798a229ffmshad68daf26d2dc80p18d220jsnb927abdecbe2',
    'X-RapidAPI-Host': 'yh-finance.p.rapidapi.com'
  }}).then(function(response) {
    var stock_data = response.data.chart.result['0'];
    console.log(response.data);
    stock_data.timestamp.forEach(function (item, index) { //converts timestamp data from seconds since epoch to US date format
      const unixTime = item;
      const date = new Date(unixTime*1000);
      xValues.push(date.toLocaleDateString("en-US")); //pushes dates to an array --> used for labels in chart
    });

    stock_data.indicators.quote[0].close.forEach(function (item, index) {
      yValues.push(item); //pushes stock values to array
    });
    if(freq==="1wk"){ //if data collected weekly..
        dateCorrection(); //corrects issue of last data point not being in weekly cycle
    }
    var smaArr = get_sma(stock_data.indicators.quote[0].close, n_steps);
    smaValues = smaArr[0]; //creates array of SMA data
    smaSet = smaArr[1];
    trainNum = Math.floor(trainSize / 100 * smaSet.length); //dividing num between training set and validation set
    smaDisp = smaValues.slice(0, Math.floor(trainSize / 100 * smaValues.length));

  }).catch(function(error) {
    console.error(error);
  });
}
```

This is the helper function used to correct the final date and stock.

```
function dateCorrection(){
    var lastDate = xValues[xValues.length-1];
    var nextLast = xValues[xValues.length-2]; //gets last two dates
    var lDate = new Date(lastDate);
    var nDate = new Date(nextLast); //converts to Date objects
    const weekMs = 60*60*24*7*1000;
    var dateBack = lDate - weekMs;
    var diff = dateBack - nDate; //checks if last two dates are 7 days apart
    if(!diff==0){
      xValues.pop() //if not, the data is excluded
      yValues.pop()
    }
  }
```

The SMA is the average of the last n-prices, and is used to smooth out the graph of our stock data. It also keeps an array of the set of values used for each average, which is used later to train the model.

The function below creates both an array of SMA values and an array of SMA sets, each n-prices long. At the beginning of the function, the arrays are also offset by n-values, in order to align with the chart labels. The function returns the completed arrays to the initData function, where they are set to global variables.

```
function get_sma(data, window_size){ //creates SMA, or Simple Moving Average, along with SMA datasets
  let avg_idx = [];
  let avg_set = [];
  for(let i=0;i<window_size;i++){ //push null data so the dataset aligns with the labels already on the chart
    avg_idx.push(null);
    avg_set.push(null);
  }
  for(let i = window_size; i< data.length; i++){ //goesh through each sma value
    let avg = 0.00, t = i - window_size;
    let temp_set =[];
    for(let k = t; k < i && k <data.length; k++){//goes through each value for the sma
      avg += data[k] / window_size;
      temp_set.push(data[k]);
    }
    avg_idx.push(avg); //array of n-size averages
    avg_set.push(temp_set); //array of n-size arrays for each average
  }
  return [avg_idx, avg_set];
}
```

The library that fetches the data, Axios, works asynchronously, but I needed to graph the stock data in the very next step. I ended up using a setTimeout in my main function to combat the issue, allowing for the data to be initialized before chart.js worked its magic.

```
initData(tick, winSize, freq);
setTimeout(() => {  mein(winSize, distance, epochNum, dataSplit, hidLayers, LearningRate, freq); }, 5000); //set delay for axios to grab data
```

Chart.js

With two data arrays- one with normal stock data, one with SMA data- and a label array of dates, I was ready to graph. Chart.js streamlines the graphing process-- all I have to do is insert my data, pick some options, and let it run. The interaction determines what points you see when you hover over the chart, and the X:ticks options determine how many label values are present on the X-axis.

```
function showChart(){ //initializes chart
  var ctx = document.getElementById("myyChart");
  aaChart = new Chart(ctx, {
    type: "line",
    data: {
      labels: xValues,
      datasets: [
        {
          label: 'Stock', //controls for displaying stock
          data: yValues,
          borderColor: "rgb(0,0,255)",
          backgroundColor: "rgb(0,0,255)",
          fill: false,
          lineTension: 0,
          borderWidth: 1,
          spanGaps: true,
          pointRadius: 0,
        },
        {
          label: 'SMA', //controls for displaying stock
          data: smaDisp,
          borderColor: "rgb(255,0,0)",
          backgroundColor: "rgb(255,0,0)",
          fill: false,
          lineTension: 0,
          borderWidth: 1,
          spanGaps: true,
          pointRadius: 0,
      }]
    },
    options: {
      interaction: {intersect: false, mode:'nearest', axis: 'x'}, //how the cursor interacts with the datapoints
      legend: {display: false},
      scales: {
          x: {
          ticks: {
            autoSkip: true, //together, creates fewer X values on axis
            maxTicksLimit: 10
          }
        }
      }
    }
  });
}
```

On the HTML side, the only thing needed to display the chart was a canvas element.

```html
<canvas id="myyChart" style=" flex:1; display: block; padding:0; width:800px; outline:1px solid □white; border:3.5px solid □#ffffff;"></canvas>
```

Chart.js also allows for the use of plugins, allowing for charts to be reactive. The chartjs-plugin-crosshair, with options below, allows for horizontal selection and zooming on the page. While it is not perfect--it's difficult to select the edge portions of the chart, and if you zoom in twice, you can't zoom out to the original chart defaults--it gets the job done, allowing for users to zoom in and see the predicted results.

```javascript
options: {
  plugins: {
    title: {
      display: true,
      text: tick + ' Stock Prices and Prediction'
    },
    tooltip: {
    },
    crosshair: {
      line: {
        color: '#F66',  // crosshair line color
        width: 1        // crosshair line width
      },
      sync: {
        enabled: true,            // enable trace line syncing with other charts
        group: 1,                 // chart group
        suppressTooltips: false   // suppress tooltips when showing a synced tracer
      },
      zoom: {
        enabled: true,                                      // enable zooming
        zoomboxBackgroundColor: 'rgba(66,133,244,0.2)',     // background color of zoom box
        zoomboxBorderColor: '#48F',                         // border color of zoom box
        zoomButtonText: 'Reset Zoom',                       // reset zoom button text
        zoomButtonClass: 'reset-zoom',                      // reset zoom button class
      },
      callbacks: {
        beforeZoom: () => function(start, end) {                  // called before zoom, return false to prevent zoom
          return true;
        },
        afterZoom: () => function(start, end) {                   // called after zoom
        }
      }
    }
  },
  interaction: {intersect: false, mode:'nearest', axis: 'x'}, //how the cursor interacts with the datapoints
```

TensorFlow.js

Finally, the meat of the project. From the outside perspective, ML has always seemed daunting, something people need years of coding experience to understand and work with. TensorFlow.js changed my mind about all this, giving me easy-to-use syntax and code that followed a logical flow.

- tf.tensor: a datatype very similar to a multidimensional array, taking both values and a shape upon initialization. Tensors are the data transferred between layers of a tf.model, making it the main object of TensorFlow.js.
- Tf.layer: a function with trainable variables which takes in a tensor and outputs a tensor. Typically has the parameters *units* (dimensionality of a layer) and *inputShape* (shape of the incoming tensor). For my project, tensors came in three flavors:
  - Dense: a 'fully connected' layer, meaning each neuron receives input from each neuron of the previous layer. I use one at the beginning, to take the SMA set data into the model, and one at the end, as a culminating layer, receiving input from the LSTM layers and reducing the output to one unit.
  - Reshape: Used to reshape inputs
  - RNN: Standing for Recurrent Neural Network, it is the workhorse of the model. Instead of having a *units* parameter, it has a *cells* parameter. In this program, I use Long Short-Term Memory, or LSTM, cells. The RNN layer is, itself, made of layers, as multiple lstmCell layers are passed to it.
- LSTM cells pass the typical SMA input, but goes further, using a second, 'hidden' state, using values returned from previous cells. The *units* parameter defines the dimensionality of the hidden states, and the number of values returned. Then, it combines LSTM cells, as hidden layers, into an array, which is used as the *cells* parameter for the RNN layer. A great explanation of how LSTM works can be found at https://tung2389.github.io/coding-note/unitslstm.
- Adam: an optimization algorithm great at working with large amounts of data and/or parameters. Allows for the customization of the learning rate, which is how the algorithm reacts to and corrects errors in the model.

The function first splits the data, taking only the portion designated for training, and converts the data into normalized tensors. The prediction offset is the distance between the SMA set and the

SMA value, allowing for farther-out predictions, and is determined by the user. Then, it creates the model, adding layers and shaping the data.

```
async function trainModel(n_size, trainSize, hidLayers, LearningRate, epochNum, dist){ //the workhorse--this creates the model

  try{
    const batch_size = 32;

    var X = smaSet.slice(n_size, trainNum); //creates training set
    var Y = smaValues.slice(n_size, Math.floor(trainSize / 100 * smaValues.length));
    for(let i=0; i<dist;i++){ //creates prediction offset
      X.pop();
      Y.shift();
    }
    var Xt = tf.tensor2d(X, [X.length, X[0].length]);//training set to tensor
    var Yt = tf.tensor2d(Y, [Y.length, 1])


    var inputs
    var outputs
    [inputs, xMax, xMin] = normalizeTensorFit(Xt); //normalizes training data
    [outputs, yMax, yMin] = normalizeTensorFit(Yt);


    const model = tf.sequential();

    model.add(tf.layers.dense({units: 64, inputShape: [n_size]})); //these two layers set the shape for the model input and allow transition
    model.add(tf.layers.reshape({targetShape: [16,4] }));//to the LSTM layers


    let lstm_cells = [];
    for (let index = 0; index < hidLayers; index++) {  //creates hidden layers
        lstm_cells.push(tf.layers.lstmCell({units: 16}));
    }

    model.add(tf.layers.rnn({ //adds hidden layers to model
      cell: lstm_cells,
      inputShape: [16,4],

    }));

    model.add(tf.layers.dense({units: 1, inputShape: [16]})); //a sort of culminating layer, the fully-connected 'dense' layer
    //is connected to every neuron of of preceding LSTM layer

    model.compile({ //compiles layers
      optimizer: tf.train.adam(LearningRate),
      loss: 'meanSquaredError'
    });
```

Below is the helper function for normalizing the training data.

```
function normalizeTensorFit(tens){ //creates tensor of data normalized between 0 and 1
  const maxval = tf.max(tens);
  const minval = tf.min(tens);

  const normalizedTensor = tf.sub(tens, minval).div(tf.sub(maxval,minval));
  return [normalizedTensor, maxval, minval];
}
```

Next, the program validates the model. In the mein function, the SMA data is again split into the training and validation portion, which are both then normalized and fed back into the model. The model process the data, and the data is un-normalized and returned as arrays of predicted values.

```
async function mein(winSize, distance, epochNum, dataSplit, hidLayers, LearningRate, freq, tick){
  showChart(tick); //displays ticker data
  let mod = await trainModel(winSize, dataSplit, hidLayers, LearningRate, epochNum, distance);

  if(!modelStoppedThrown){
    var trainX = smaSet.slice(winSize, trainNum-distance); //creates data sets for validation
    var validX = smaSet.slice(trainNum- distance);
    let trainY = validateModel(mod, trainX, winSize, distance); //creates validation
    let validY = validateModel(mod, validX, winSize, 0);
    updateChart(trainY, winSize, false) //updates chart
    updateChart(validY, winSize, true)
    predictData(freq, distance);
  }
  modelStoppedThrown = false;
}

function validateModel(model, dataX, winSize, dist){ //uses model to predict values based on n-length datasets
  let dataTen = tf.tensor2d(dataX, [dataX.length, winSize])
  let dataNorm = tf.sub(dataTen, xMin).div(tf.sub(xMax, xMin));//normalizes data
  let dataOut = model.predict(dataNorm);
  let outNorm = tf.add(dataOut.mul(tf.sub(yMax, yMin)), yMin); //un-normalizes data
  let dataY = Array.from(outNorm.dataSync());
  for(let i=0; i<dist;i++){ //creates prediction offset
    dataY.unshift(null);
  }
  return dataY; //returs array of predicted values
}
```

The predicted data is then offset to match the chart labels, and the updated arrays are added to new datasets in the chart. New labels are also created to match by converting the last date in the labels array to ms since epoch, adding a week’s worth of ms (or days, depending on frequency of data chosen), and pushing the new date to the end of the array. The chart is updated, and the new data is displayed.

```
function updateChart(newData, n_size, valid){ //add modeled data to chart
  if (valid){
    for(let i=0; i<trainNum-1; i++){
      newData.unshift(null); //shifts data to align with chart labels
    }
    var newDataset = {
      label: 'Predicted Result',
      data: newData,
      borderColor: "rgb(0,0,0)",
      backgroundColor: "rgb(0,0,0)",
      fill: false,
      lineTension: 0,
      borderWidth: 2,
      spanGaps: true,
      pointRadius: 0,
    }
  }else {
    for(let i=0; i<n_size-1; i++){
      newData.unshift(null);
    }
    var newDataset = {
      label: 'Training Result',
      data: newData,
      borderColor: "rgb(0,240,0)",
      backgroundColor: "rgb(0,240,0)",
      fill: false,
      lineTension: 0,
      borderWidth: 2,
      spanGaps: true,
      pointRadius: 0,
    }
  }
  aaChart.data.datasets.push(newDataset); //Adds newly created dataset to list of `data`
  aaChart.update(); //Updates the chart
}

function predictData(freq, dist){
  var dateMs = finalMS;
  let weekMs = 60*60*24*7*1000;
  let dayMs = weekMs/7;

  if(freq==="1wk"){
    for(let i=0;i<dist;i++){

      let newDate = new Date(dateMs + weekMs);
      aaChart.data.labels.push(newDate.toLocaleDateString("en-US"));
      dateMs += weekMs;
    }
  }else{
    for(let i=0;i<dist;i++){
      let newDate = new Date(dateMs + dayMs);
      aaChart.data.labels.push(newDate.toLocaleDateString("en-US"));
      dateMs = newDate;
    }
  }
  aaChart.update();
```

Cleaning Up/HTML Reactivity

Of course, it is no fun if you can't *see* what the program is doing, so the HTML is updated and styled to display the user inputs and data. The user inputs are done using a form, and the js backend grabs values from each input to control the program when the submit button is clicked.

```html
<form id="something" style="padding:25px; text-align:center">
  <div style="display: flex; flex-direction: row;">
    <div style="flex:1; text-align:right;">
      Enter ticker:          <input type="text" name="search" value="VLO" placeholder="Enter ticker..." style="width: 200px"><br>
      Window size for training: <input type="number" name="window_size" value="20" placeholder="Enter window size for training..." style="width: 200px"><br>
      Prediction distance:      <input type="number" min="1" max="5" name="distance" value="1" placeholder="Prediction distance..." style="width: 200px"><br>
      Number of epochs:         <input type="number" min="1" max="20" name="epochNum" value="10" placeholder="Number of epochs..." style="width: 200px">

    </div>
    <div style="flex:1; text-align:right;">
      Get historical stock data from:
        <select name="freq" id="freq">
          <option value="1wk">Each week</option>
          <option value="1d">Each day</option>
        </select><br>
      Training data split:    <input type="number" min="0" max="249" name="dataSplit" value="95" placeholder="Training data split..." style="width: 200px"><br>
      Number of hidden layers:<input type="number" min="1" max="8" name="hidLayers" value="3" placeholder="Number of hidden layers..." style="width: 200px"><br>
      Learning rate:          <input type="number" min="0.001" max="0.1" step="0.001" name="LearningRate" value="0.01" placeholder="Learning rate..." style="width: 200px">

    </div>
  </div>
  <input type="submit" value="submit" name ="submit" style="margin:7px">
</form>
```

```javascript
function search(query){
 //othe portions left out for readability
  let tick = query.target.elements["search"].value; //gets data from HTML
  let winSize = parseInt(query.target.elements["window_size"].value);
  let distance = parseInt(query.target.elements["distance"].value);
  let epochNum = parseInt(query.target.elements["epochNum"].value);
  let freq = query.target.elements["freq"].value;
  let dataSplit = parseInt(query.target.elements["dataSplit"].value);
  let hidLayers = parseInt(query.target.elements["hidLayers"].value);
  let LearningRate = parseFloat(query.target.elements["LearningRate"].value);
  initData(tick, winSize, freq, dataSplit);

  setTimeout(() => {  query.target.elements["submit"].disabled = false; }, 5000);
  setTimeout(() => {  mein(winSize, distance, epochNum, dataSplit, hidLayers, LearningRate, freq, tick); }, 5000); //set delay for axios to grab data
}

something.addEventListener("submit", search, false)
```

The program also displays the epoch and loss from the model.fit sequence to the user in real-time. An epoch is one run-through of the entire dataset through the model, forwards and backward. Loss represents the penalty for the model being wrong--so the lower, the better. The code snippet is the second half of trainModel. After the end of each epoch, the epoch number and loss data are formatted and appended to the HTML, and a loading animation is shown

```
    model.compile({ //compiles layers
      optimizer: tf.train.adam(LearningRate),
      loss: 'meanSquaredError'
    });

    var load = document.getElementById("load");//sets up HTML logging
    load.style.visibility = "visible";

    const hist = await model.fit(inputs, outputs,   //uploads datasets and fits the model
      { batchSize: batch_size, epochs: epochNum, callbacks: {
        onEpochEnd: async (epoch, log) => {
          console.log(log)
          var logStr = log.loss;
          var epochInt = parseInt(epoch)+1;
          logEpoch(epochInt,logStr); //logs data from each epoch
          if(cancelThrown){
            modelStoppedThrown = true;
            cancelThrown=false;
            model.stopTraining = true;
          }
        }
      }
    });
    load.style.visibility = "hidden";
    return model;

  } catch(error){
    console.log("error")
    return null;
  }
}

function logEpoch(epoch,log){ //logs epoch data
  var element = document.getElementById("LSTM_data");
  var para = document.createElement("p");
  para.style.cssText = 'font-size:16px';
  var node = document.createTextNode("     epoch "+ epoch +": "+log);
  para.appendChild(node);

  element.appendChild(para);
}
```

Finally, to make sure the impatient, button-spamming user doesn't cause the program to go up in flames, the search button has some fail-safes built in. First, each button press after the first one resets the arrays and destroys the chart. It also kills the model, if it is in the middle of training (you may have noticed the flags set up on the previous page). It clears the data log from the model's training, and to prevent any other issues, the button is disabled for 5 seconds after pressing it each time.

```
function search(query){
  if(xValues.length != 0){ //destroys graph when doing new query, resets HTML
    xValues = [];
    yValues = [];
    aaChart.destroy();
    const bill = document.getElementById('load');
    if(bill.style.visibility === "visible"){ //using as flag to see if model is in training process
      cancelThrown = true; //kills model training
      bill.style.visibility === "hidden"
    }
    const epochData = document.getElementById('LSTM_data');
    setTimeout(() => { epochData.innerHTML = ''; }, 3000); //set delay for clearing page
  }
  query.preventDefault();
  query.target.elements["submit"].disabled = true;
  setTimeout(() => {  query.target.elements["submit"].disabled = false; }, 5000);

  //other stuff, taken out for readability...

  setTimeout(() => {  mein(winSize, distance, epochNum, dataSplit, hidLayers, LearningRate, freq, tick); }, 5000); //set delay for axios to grab data
}
```

Finally, the code was compiled with the libraries using browserify into bundle.js, which is what index.html directly references. The rest of the project is completed on the HTML side, with background gradients, cover images, information, and styling all being added. Because all the libraries are attached, the program should work easily from the flash drive--but if not, it can always be accessed at https://nerdman42.github.io/.

Why does a larger prediction distance appear to only shift the chart to the right?

The farther the offset between the SMA dataset and the corresponding SMA value used in the pairing (dubbed the ‘prediction distance’), the less correlation between the two. When this happens, the LSTM model begins to rely on only the last number in the SMA dataset, causing the appearance of a shift, rather than an actual calculation. This is why there is a hard stop at 5 for the prediction distance parameter-- any larger offset would cause a complete lack of correlation.

Why isn’t the model predicting accurately?

As of right now, the market is more volatile than ever before--something that the model hasn’t had the chance to learn. Additionally, the model tracks the SMA, *not* the actual stock itself. This means that the SMA doesn’t capture extreme or sudden shifts in the stock market, and the model will be completely indifferent.
Finally, the predictions may just be inaccurate--the model is trying to find a pattern amidst the madness of the stock market. There isn’t a standalone equation that can model how the market moves, and there never will be. This model just shows an application of machine learning, and demonstrates the power of LSTM modeling.

Issues with the Chart

Of course, not everything about this program is perfect. The chart.js plugin has issues zooming in. In order to select the very right edge of the chart, you should start your cursor off the side of the chart, tothe right, and drag it left. It may take a few times, just be patient. Also, don’t zoom in more than once in a row, or else you won’t be be able to zoom out to the default level.

Citations

Chart.js | Chart.js. (n.d.). Retrieved July 13, 2022, from https://www.chartjs.org/docs/latest/

Mayank, M. (2020, October 17). *A practical guide to RNN and LSTM in Keras*. Medium. https://towardsdatascience.com/a-practical-guide-to-rnn-and-lstm-in-keras-980f176271bc

Sharma, P. (2020, October 20). Keras Dense Layer Explained for Beginners. *MLK - Machine Learning Knowledge*. https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/

TensorFlow. (n.d.). TensorFlow. Retrieved July 13, 2022, from https://www.tensorflow.org/

Time Series Forecasting with TensorFlow.js—Hong Jing (Jingles). (n.d.). Retrieved July 13, 2022, from https://jinglescode.github.io/time-series-forecasting-tensorflowjs

Understanding LSTM Networks—Colah's blog. (n.d.). Retrieved July 13, 2022, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Units in LSTM. (n.d.). Retrieved July 13, 2022, from https://tung2389.github.io/coding-note/unitslstm/

And many, many stack overflow forums.