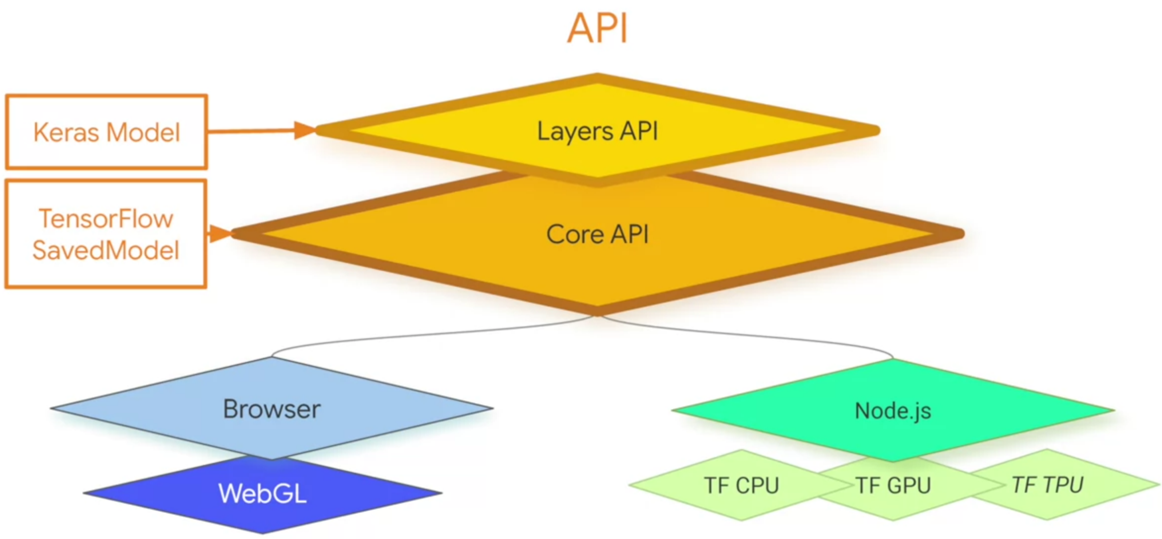
Week 1

Training TensorFlow with JS

- Using Tensorflow.js it's possible to execute it in simple web pages
- The architecture of Tensorflow.js



- It's meant to be ran in browser or Node.js servers
- First HTML code

```
<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 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]);
    const model = tf.sequential();
    model.add(tf.layers.dense({units:1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError', optimizer:'sgd'});
    model.summary();
    doTraining(model).then(() => {
        alert(model.predict(tf.tensor2d([10], [1,1])));
    });
</script>
<body>
    <h1>First HTML Page</h1>
</body>
</html>
```

Training Models with CSV Files

One Hot encoding

- Here is an example where one hot encoding is useful
- Lets say the neural network is classifying rock paper and scissors
- The result that is desired is getting a number close to one for the correct image

- One hot encoding is basically encoding the desired output to the representation of the that we want to see in the output
- basically one of the values in the array is a "hot" value
- In the dataset we are using:

setosa	1	0	0
virginica	0	1	0
versicolor	0	0	1

- Code for Iris classifier

```
<html>
    <head>Iris Classifier</head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
    <script lang="js">
        async function run(){
            const csvUrl = 'iris.csv';
            const trainingData = tf.data.csv(csvUrl, {
                columnConfigs: {
                    species: {
                        isLabel:true
                    }
                }
            });
            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)
            const numOfFeatures = (await trainingData.columnNames()).length - 1;
            const model = tf.sequential();
            model.add(tf.layers.dense({
                inputShape: [numOfFeatures],
                activation: "sigmoid", units: 5}))
            model.add(tf.layers.dense({activation: "softmax", units: 3}));

            model.compile({loss: "categoricalCrossentropy", optimizer: tf.train.adam(0.06)});

            await model.fitDataset(convertedData,
                             {epochs:100,
                              callbacks:{
                                  onEpochEnd: async(epoch, logs) =>{
                                      console.log("Epoch: " + epoch + " Loss: " + logs.loss);
                                  }
                              }});

            const testVal = tf.tensor2d([4.4, 2.9, 1.4, 0.2], [1, 4]);
            const prediction = model.predict(testVal);
            const pIndex = tf.argMax(prediction, axis=1).dataSync();
            const classNames = ["Setosa", "Virginica", "Versicolor"];
            alert(classNames[pIndex]);
        }
        run();
```

```
    </script>
    <body>
        <h1>Iris Classifier with TensorFlow JS</h1>
    </body>
</html>
```

Exercise

- Code for exercise this week

```
<html>
<head></head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
    <script lang="js">

        async function run(){
            const trainingUrl = 'wdbc-train.csv';

            // Take a look at the 'wdbc-train.csv' file and specify the column
            // that should be treated as the label in the space below.
            // HINT: Remember that you are trying to build a classifier that
            // can predict from the data whether the diagnosis is malignant or benign.
            const trainingData = tf.data.csv(trainingUrl, {
                columnConfigs: {
                    diagnosis : {
                        isLabel : true
                    }
                }
            });

            // Convert the training data into arrays in the space below.
            // Note: In this case, the labels are integers, not strings.
            // Therefore, there is no need to convert string labels into
            // a one-hot encoded array of label values like we did in the
            // Iris dataset example.
            const convertedTrainingData = trainingData.map(({xs, ys}) => {
                return {xs: Object.values(xs), ys:Object.values(ys)};
            }).batch(10)

            const testingUrl = 'wdbc-test.csv';

            // Take a look at the 'wdbc-test.csv' file and specify the column
            // that should be treated as the label in the space below..
            // HINT: Remember that you are trying to build a classifier that
            // can predict from the data whether the diagnosis is malignant or benign.
            const testingData = tf.data.csv(testingUrl, {
                columnConfigs: {
                    diagnosis : {
                        isLabel : true
                    }
                }
            });

            // Convert the testing data into arrays in the space below.
            // Note: In this case, the labels are integers, not strings.
            // Therefore, there is no need to convert string labels into
            // a one-hot encoded array of label values like we did in the
            // Iris dataset example.
            const convertedTestingData = testingData.map(({xs, ys}) => {
                return {xs: Object.values(xs), ys:Object.values(ys)};
            }).batch(10)


            // Specify the number of features in the space below.
            // HINT: You can get the number of features from the number of columns
            // and the number of labels in the training data.
            const numOfFeatures = (await trainingData.columnNames()).length - 1;


            // In the space below create a neural network that predicts 1 if the diagnosis is malignant
            // and 0 if the diagnosis is benign. Your neural network should only use dense
```

```
            // layers and the output layer should only have a single output unit with a
            // sigmoid activation function. You are free to use as many hidden layers and
            // neurons as you like.
            // HINT: Make sure your input layer has the correct input shape. We also suggest
            // using ReLu activation functions where applicable. For this dataset only a few
            // hidden layers should be enough to get a high accuracy.
            const model = tf.sequential();
            model.add(tf.layers.dense({inputShape: [numOfFeatures], activation: "relu", units: 64}))
            model.add(tf.layers.dense({activation: "relu", units: 32}))
            model.add(tf.layers.dense({activation: "sigmoid", units: 1}));


            // Compile the model using the binaryCrossentropy loss,
            // the rmsprop optimizer, and accuracy for your metrics.
            model.compile({loss: "binaryCrossentropy", optimizer: tf.train.adam(0.06), metrics: "acc"});


            await model.fitDataset(convertedTrainingData,
                             {epochs:100,
                              validationData: convertedTestingData,
                              callbacks:{
                                  onEpochEnd: async(epoch, logs) =>{
                                      console.log("Epoch: " + epoch + " Loss: " + logs.loss + " Accuracy: " + logs.acc);
                                  }
                              }});
            await model.save('downloads://my_model');
        }
        run();
    </script>
<body>
</body>
</html>
```