**1**

***Your First TypeScript Program***

**Learning Objectives**

By the end of this lesson, you'll be able to:

* Understanding how TypeScript Works
* Your first TypeScript Program

# **Introduction**

Let’s TypeScript! Since we already set up our work environment, let’s write our first program in style and trade the typical “Hello World” for an Artificial Intelligence piece of code, see how everything comes together without worrying too much about the details. The idea is to get a feel for language capabilities and what a better way to see TypeScript in all its glory than a bit of AI code!

Continue introduction here

# **Your First TypeScript Program**

As our first program we are going to move away from the traditional “Hello World” and as promised we are going to create a small Artificial Neural Network (APerceptron). Since this is our first program we are going to create our own Perceptron in TypeScript and for all intent and purposes it will be at the most basic level of what it can be skipping a lot of the explanations and theory behind AI and focus on typescript nevertheless I will provide enough basic information for you to understand what are we building.

In order to make thingsthigs we will use no external libraries and build everything from scratch, in the ai space there advanced libraries that do a lot of the heavy lifting for arrays manipulation and math s

Our most basic Perceptron’s require at least the following parts:

1. Input layer
2. Hidden layer
3. Output layer
4. Weights between the layers
5. A deliberate activation function for every hidden layer. In this we’ll employ the Sigmoid activation function.

# Our Problem

We want to create a small Perceptron that will predict the correct output value when provided a new set of data. The output should always equal to first value in the table provided.

In this case we would expect see to the same result in the output where we see the question (?) mark.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Input | |  | Output |
| **Data 1** | **0** | **0** | **1** | **0** |
| **Data 2** | **1** | **1** | **1** | **1** |
| **Data 3** | **1** | **0** | **1** | **1** |
| **Data 4** | **0** | **1** | **1** | **0** |
| **New situation** | **1** | **0** | **0** | **?** |

# A close up of a clock Description automatically generated

# **The entire exercise will run on the terminal for our first program there is no HTML or CSS involved just pure TypeScript code and results are printed to the terminal.**

# **Our project is broken down in 3 files App.ts, Perceptron.ts and NumTs.ts**

# **App.ts is the wrapper app while Perceptron.ts is our neural network class and NumTS.ts essentially is a utility class for specific functions that will do some calculations for us.**

# **First let’s take a look at the code for App.ts we will do a 40 000** thousant feet helicopter view of the code and go over the mayor pieces before we dive into actual code.

# 

# **A close up of a logo Description automatically generated**

# The first part of the code is to import two classes our Neural Network Perceptron.ts code and an utility class NumTS.ts our App.ts class is essentially like an envelope class to run the project. You will be able to reference import statements in chapter X

# **A screenshot of a computer Description automatically generated**

# Following our import statements, we declare our global variables and initialize our Perceptron as an object and our numTS class as we need it to use some functions from it.

# Then we define our learning rate learning rate, steps, input, output and testing data variables. As you can see all our variables are typed, however some I left purposely as type “any” this means they can take on any type, we will discuss that in the next chapter however for now just keep that in mind.

**A screenshot of a cell phone

Description automatically generated**

Finally to run our neural network we call on the init() to initialize the program and the call on the train() function of out Perceptron object.

The we run a for loop and prints the result in the consol. In essence is pretty simple but there is a lot to it. Let’s take a look now at out Perceptron class:

A close up of a screen

Description automatically generated

We start by importing our NumTS utils this is so our Perceptron class is aware of our utility class NumTS so we can use it’s methods within our Perceptron class to help us do some calculations

A screenshot of a cell phone

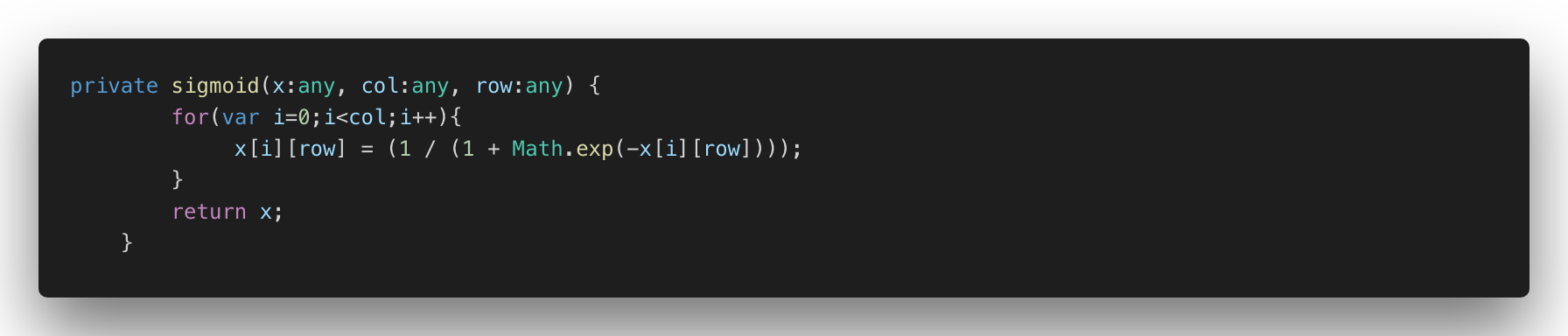
Description automatically generated

We start our Perceptron class with the keyword **export** this is to indicate TS this class can imported and expose its public methods.

The constructor normally is the first thing that executes then you instantiate a class so we take advantage to initialize our weights by creating out array and also manually initializing a it values with random numbers.



We use our init() function to print weights and display them in the console. Notice I use the console.log() to print to the console messages so we can see what’s being executed within our code.



Our sigmoid function in essence normalizes the values in our array by giving us a value between 0 and 1. This is my interpretation of the equation in typescript . Wikipedia defines sigmoid function as such: A **sigmoid function** is a [mathematical function](https://en.wikipedia.org/wiki/Function_(mathematics)) having a characteristic "S"-shaped curve or **sigmoid curve**. Often, *sigmoid function* refers to the special case of the [logistic function](https://en.wikipedia.org/wiki/Logistic_function) shown in the first figure and defined by the formula.

The sigmoid is used frequently in neural networks if you want to read more about reference this page in Wikipedia: <https://en.wikipedia.org/wiki/Sigmoid_function>



Our derivative sigmoid function completes the work by squeezing our input into a 0 or a 1.

A close up of a screen

Description automatically generated

Our train function takes 4 inputs : inputs , real outputs, iteration and learning rate. This first part only uses a method called “show” from our utils class to print nicely the values coming for these variables into the console.

A screenshot of a computer

Description automatically generated

We initialize our delta weights with an array and set it values to zero. We do manually as in other languages like python have libraries that do this automatically in our case to keep things simple, we are doing it manually.

A screenshot of a cell phone

Description automatically generated

The training happens in the next 3 loops. We do our forward pass, back propagation an adjust the weights. In this stage we train our neural network, so it can make an accurate prediction. Each input will have a weight which is positive or negative. These weights if they have a big number weather is positive or negative will influence our results.

A screenshot of a cell phone

Description automatically generated

Our

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

NumTS

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated



A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**Now let’s copy and paste this code and run it.**