Neural Networks
Deep Learning
and friends...

Agenda

- AI – Where does DL fit?
- How is it different from “Classical” Programming
- Path from Design to Production
- What is a Neuron
  - With live examples
- Neuronal Network: TensorFlow playground
- Machine Learning
  - Layers
  - See it at work on TF Playground
- Training a Model
  - Building the Network
- Car Insurance Demo
- Character Recognition
  - With OpenCV

The Tools

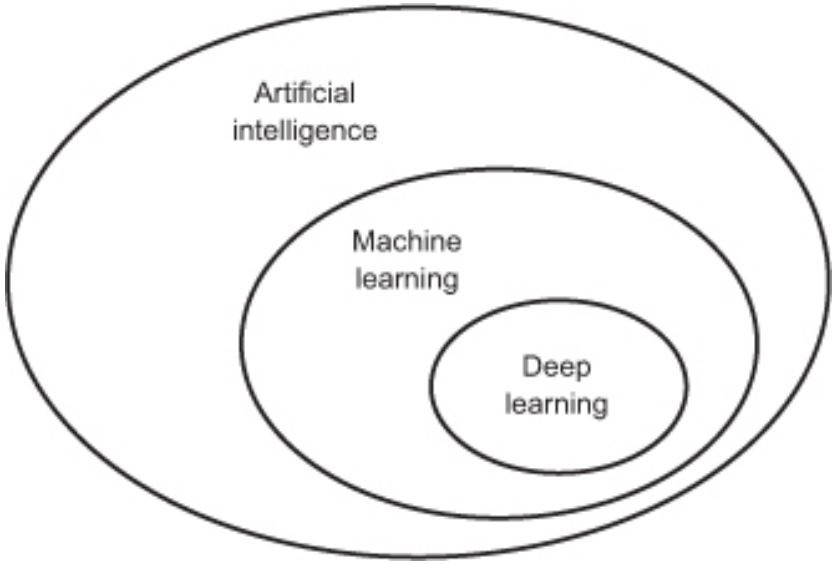
- Java
- Python
- Jupyter Notebooks
  - For Python
  - For Java
- TensorFlow
- Keras
- OpenCV

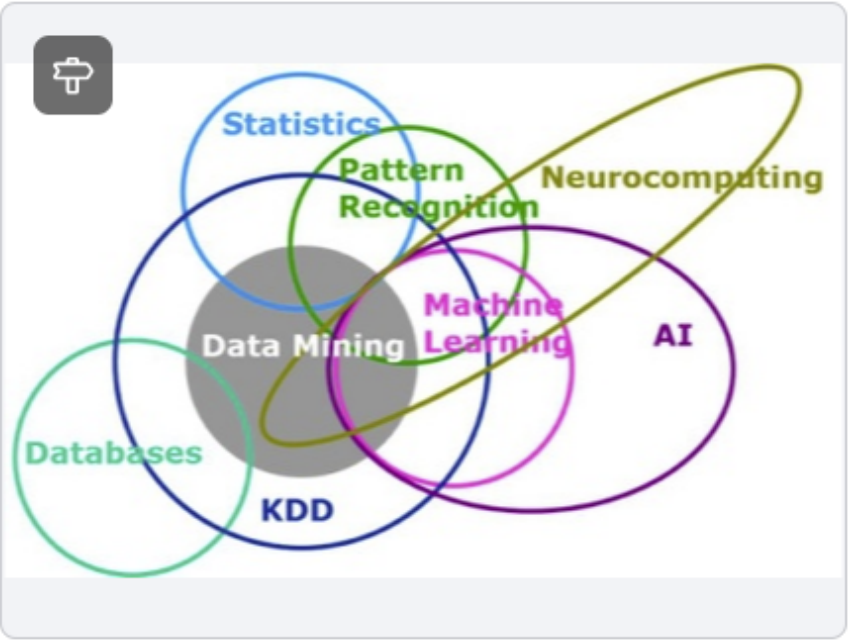
We will

- See what ML/DL is, why it is useful, and what challenges it might face
- See what a neuron is, where and how it fits
- See what a Neuron Network is and how to build one
- See how to use a Neuron Network to train a model
- Use a simple example, train a model, and use it
- Take a more complex (aka real) example, end to end

Machine Learning

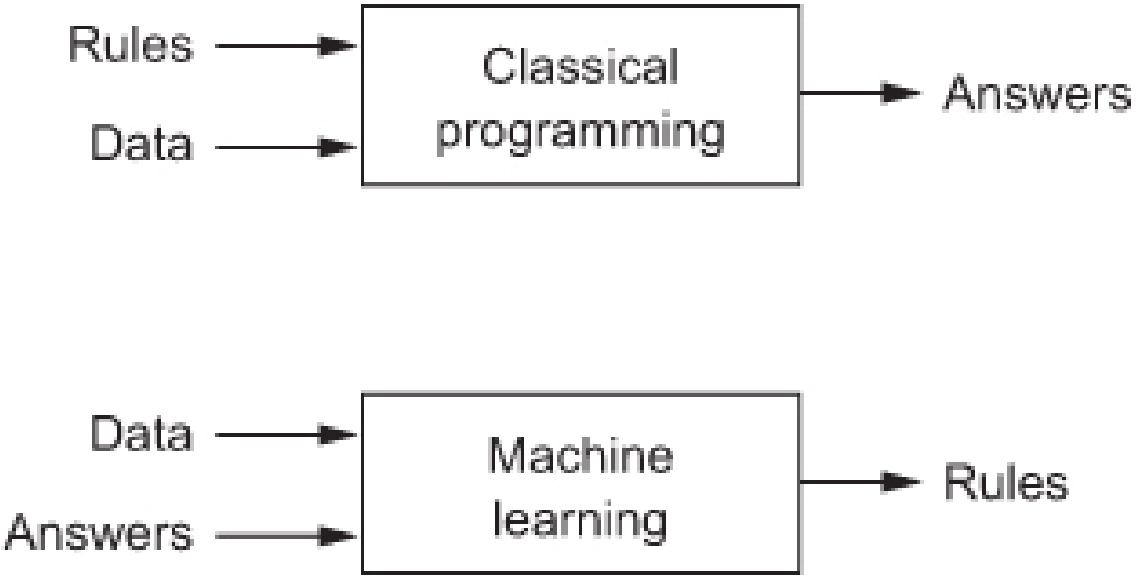
Machine Learning





(KDD: Knowledge Discovery in Databases)

How is that different?



For the learning phase – aka model training – you need a lot of data (the more the better), separated in two groups:

- Input data, parameters of the equation (like age, speed, mileage)
- Known output, result (risk level)

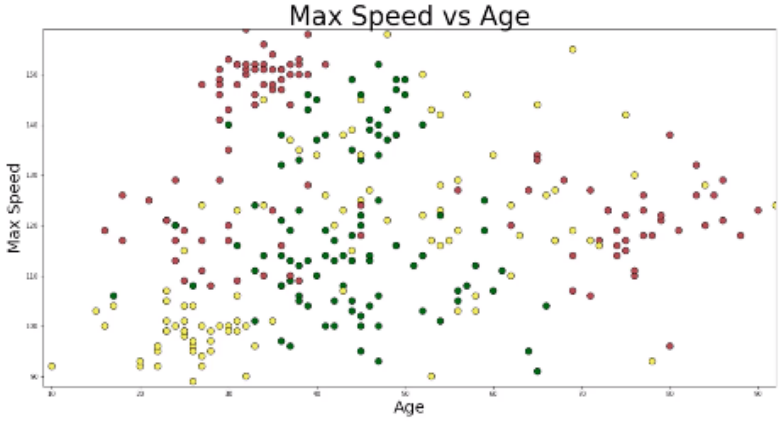
The *learning phase* is the process that tries to find the way to reach the expected result with the input parameters ***for all the data*** of the *training data set*.

Where it fits

- It is all about the **MODEL**
- The **MODEL** is the entity that will eventually apply the rules found during the *Training*
  - *Training* the **MODEL** is equivalent to *finding the Rules* to apply to *future* input data to *predict* the output
- Once *trained*, the model is deployed to *production*
  - Real world and real time data can be sent to the **MODEL** to get its *Predictions*
- A deployed **MODEL** can be re-trained and re-deployed to *production*

Our first example

- Let’s say you are an insurance company, and you want to be able to predict the risk associated with a potential customer, based on
  - His age
  - The maximum speed of his car
  - The mileage he drives per year
- We do have existing data, from some insurance company database



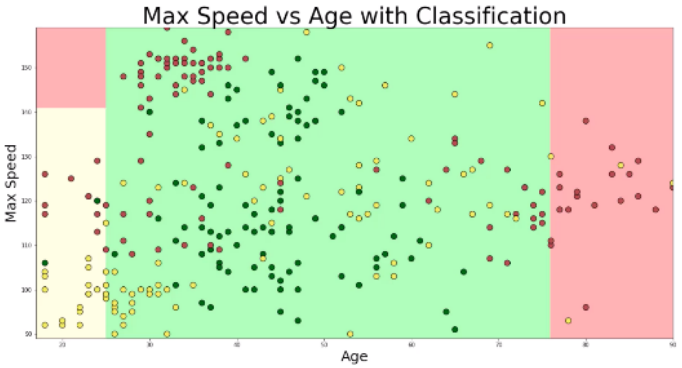
Apply a predefined rule

```
red = 0
green = 1
yellow = 2

def evaluate_risk(age, speed, miles_per_year):
  if age < 25:
    if speed > 140:
      return red # Crazy young guy, car too fast
    else:
      return yellow # Car is slow enough for medium risk

  if age > 75:
    return red # Get off the road, old man!

  if miles_per_year > 30:
    return red # You drive too much

  if miles_per_year > 20:
    return yellow

  return green # otherwise, low risk
```

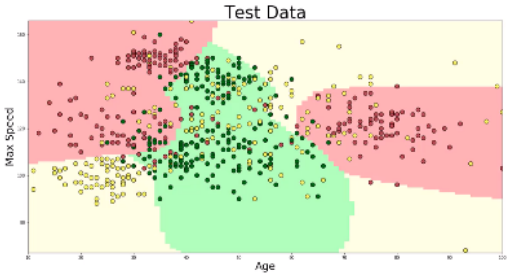
Notebook here: http://localhost:8888/notebooks/NeuralNetworks.ipynb

What we have vs what we want

- This would produce something like this:



- As opposed to what we would expect:



Not good, obviously!

- We need another way to tackle that.
- Let’s try to teach the machine ...

Neurons

(aka Artificial Neurons)

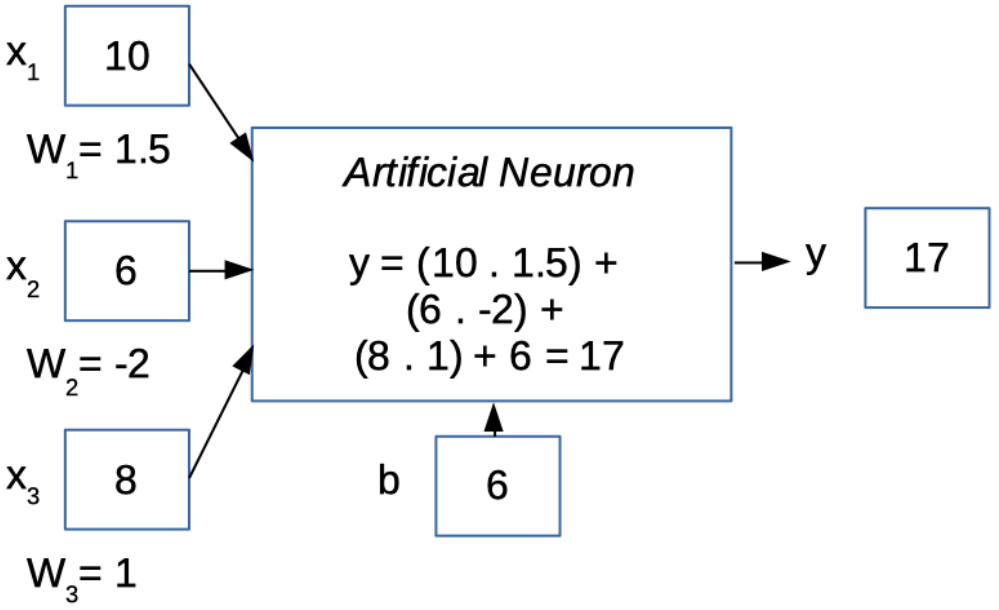
What's a Neuron

- First of all, ***this is not new!***
- It looks a lot like a mathematical function, because this is what it is
- A Neuron has a single numerical output (usually called **y**)
- It can have several inputs (also called dimensions), usually named $\mathbf{X_i}$, associated with a weight $\mathbf{W_i}$.
- It also has a constant, called the bias (noted **b**)
- Its formula is

$$y = \left(\sum_{i=1}^{n} x_i . W_i\right) + b$$

- In the next sections, they will be part of the *hidden layers*

Example



- Yes, it is very simple!
- Java Notebook
- Web page

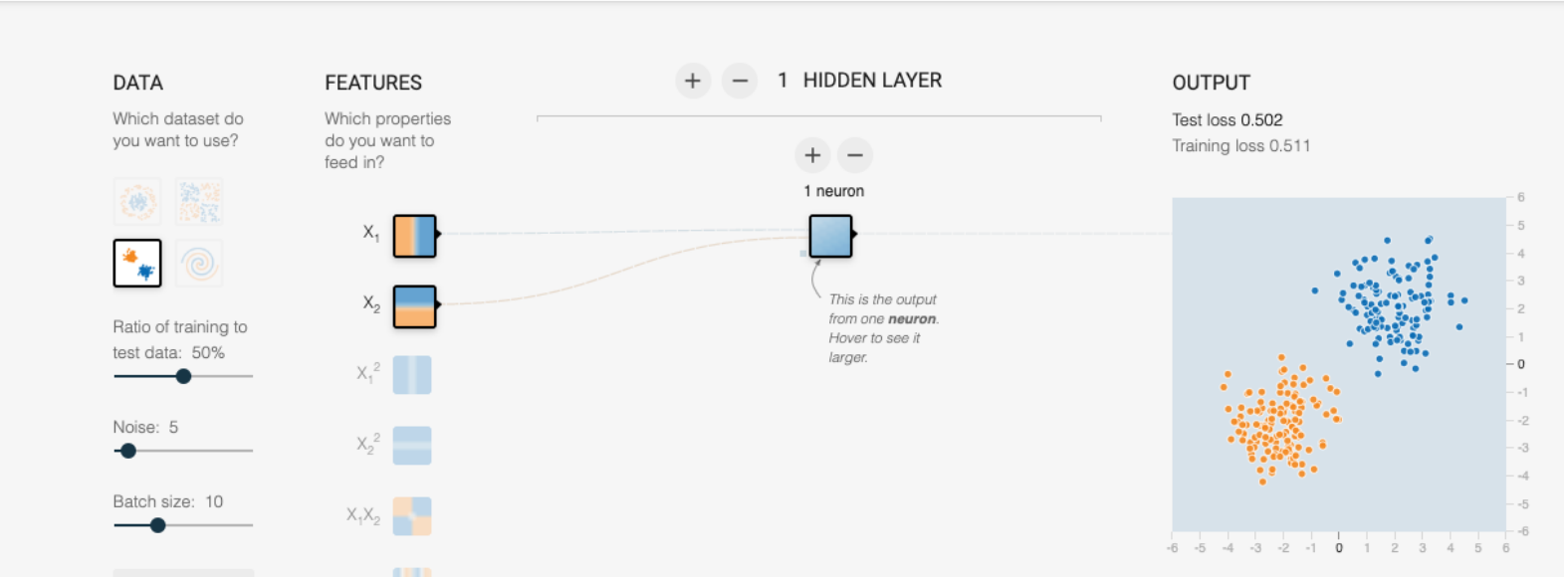
Neuron Networks

Neural Networks

- Introducing TensorFlow Playground
  - The different datasets
  - See the input nodes
  - The ***hidden layers***
    - ***Note:*** The number of hidden layers is called the ***depth*** of the network. This is where the ***Deep*** in ***Deep-Learning*** comes from.
  - The weights (lines thickness)
  - The bias (small squares at the bottom left of the neurons)
  - The run button
    - Notice the loss graph
      - Test, training losses
  - Learning rate
  - Activation
  - ...

Note: A ***Tensor*** is the vector containing the input data

TensorFlow Playground



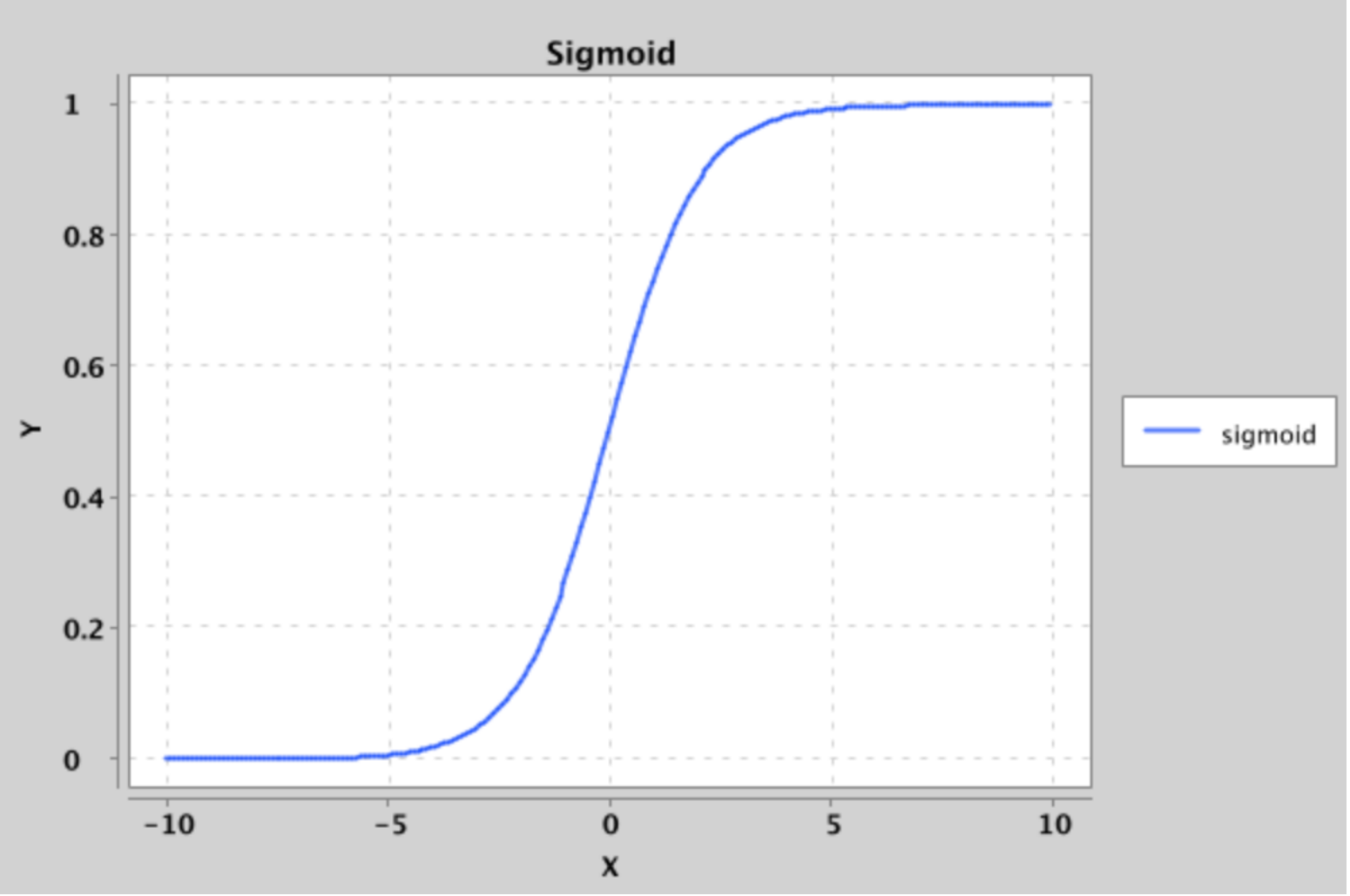
- Here we have 2 input dimensions, $\mathbf{x}_1$ (abscissa) and x_2 (ordinate).
- The ***weight*** of each dimension is represented by the thickness of the line between the dimension and the neuron(s)
- The ***bias*** of the neuron is represented by the (very) small little gray square at the bottom left of the neuron(s).
- *The output will be a single number, ranging from -1 to +1.*
- The background of the graphic on the right will reflect the prediction, it will be orange if $y < 0$, and blue if $y > 0$.
- You can change the weights associated with each box (dimensions and neuron(s)), as well as the bias of each neuron.

Activation Functions

- An activation function is triggered after weight(s) and bias are applied
- It “re-centers” the result (y) of the neuron
  - In the previous TF example, the result will be “orange” or “blue”. A value like -1 or +1, 0 or +1 is absolutely good enough
- Common activation functions are
  - Sigmoid
  - TanH
  - Step
  - ReLU
  - SoftMax
- See them live here

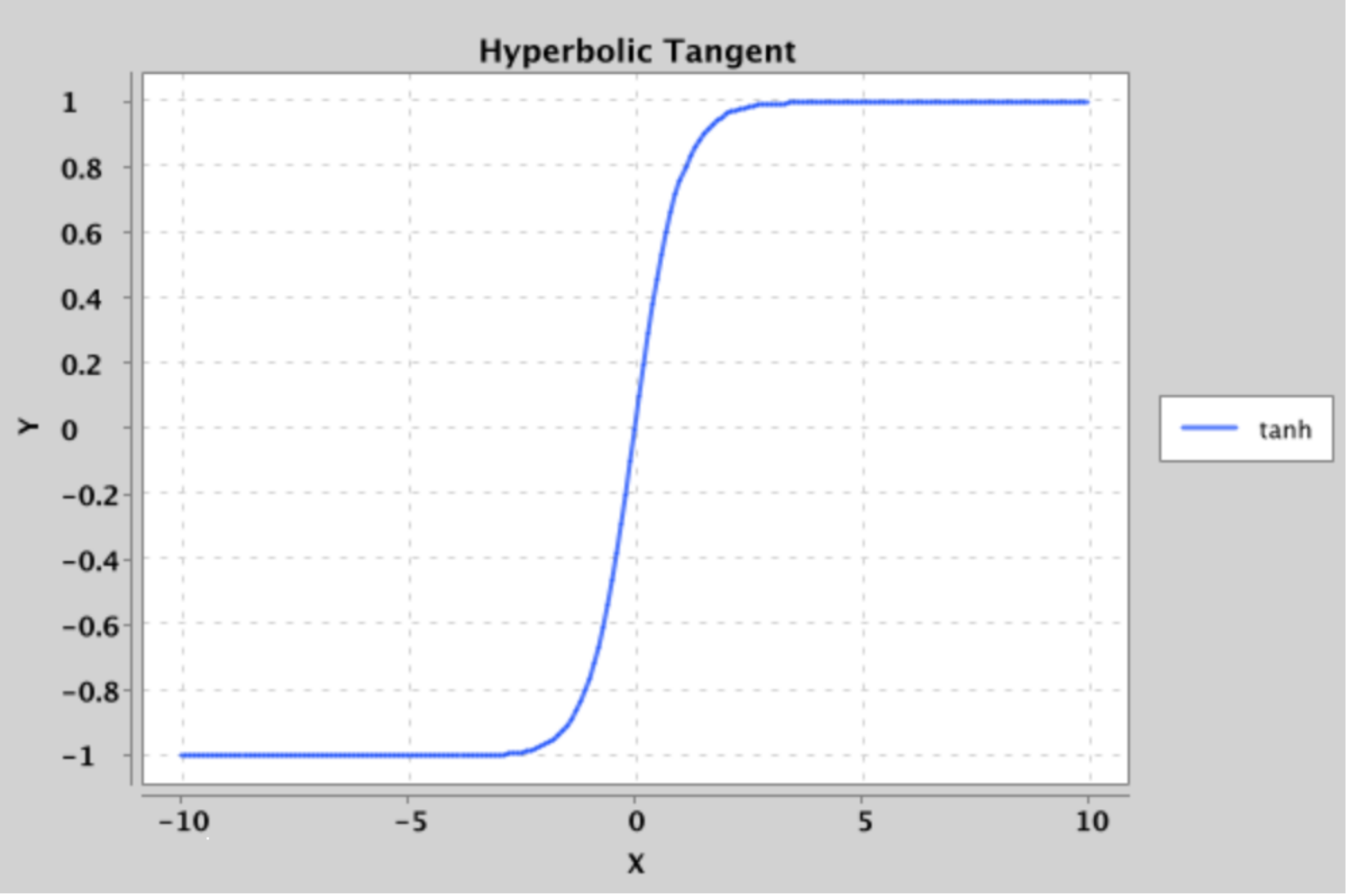
Sigmoid

$x \rightarrow (1 / (1 + e^{-x}))$



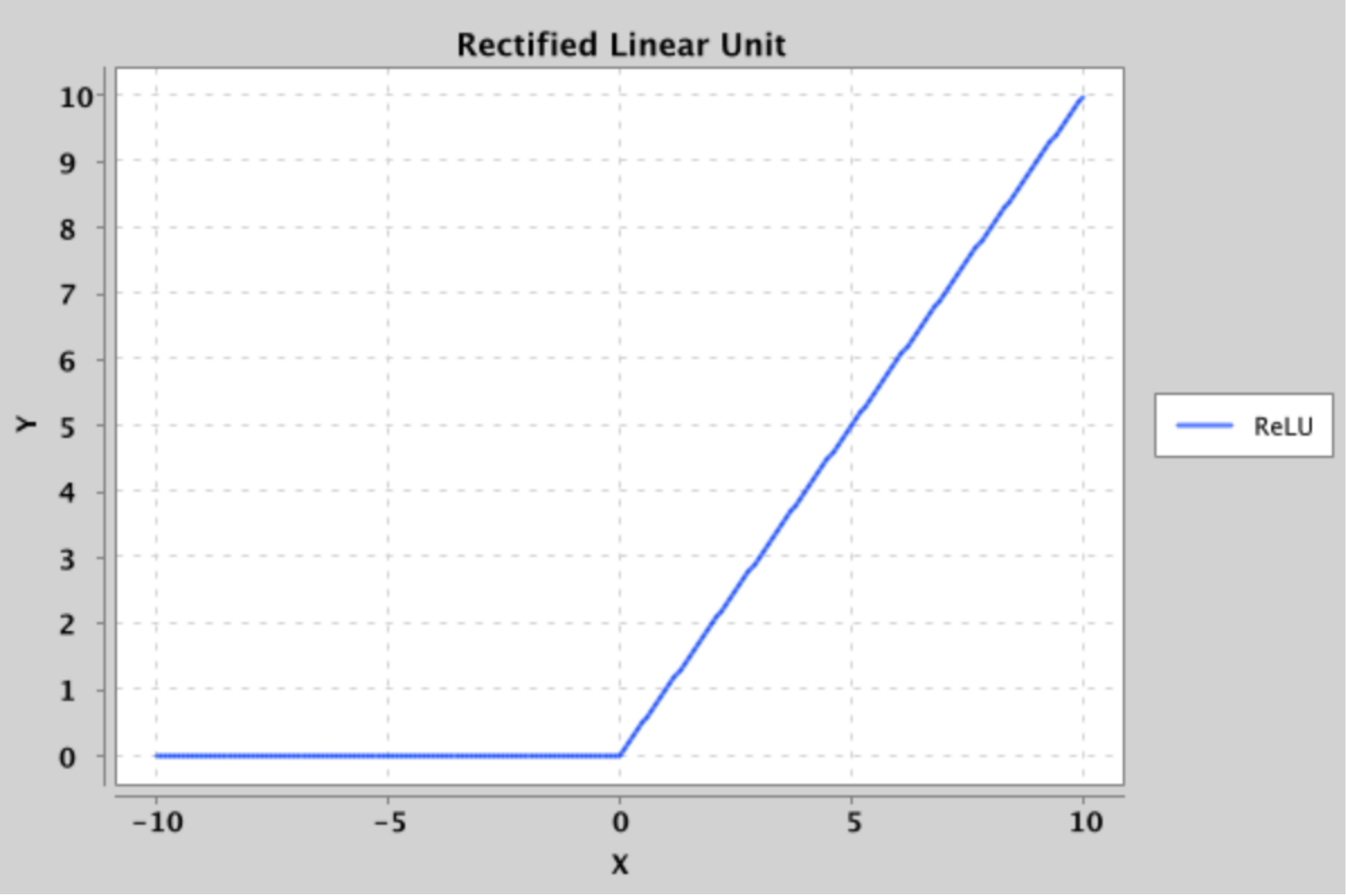
Hyperbolic Tangent

$$x \rightarrow (e^{x} - e^{-x}) / (e^{x} + e^{-x})$$



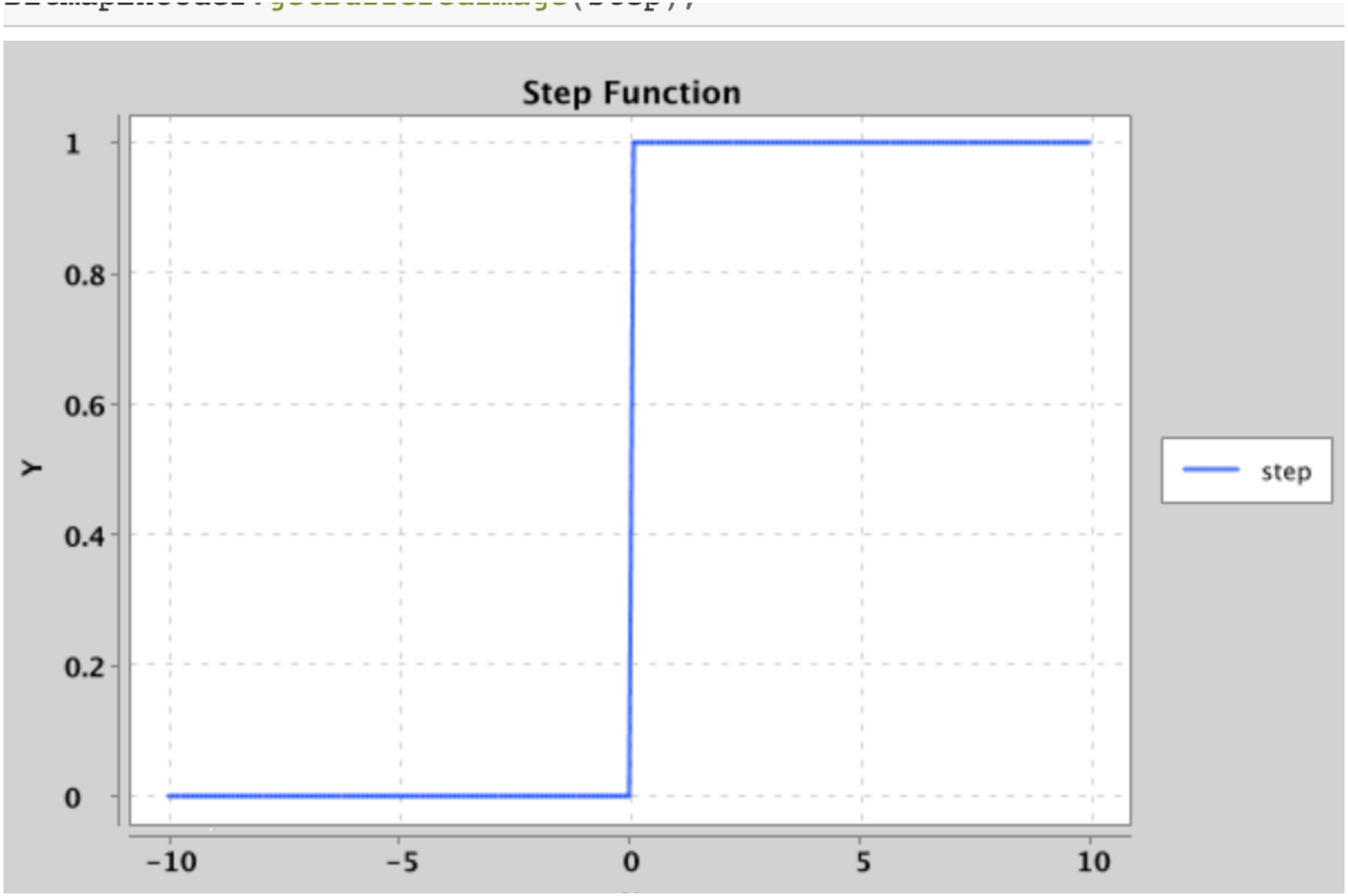
Rectified Linear Unit (ReLU)

```
x → x < 0 ? 0 : x
```



Step

```
x → x < 0 ? 0 : 1
```



SoftMax

- Depends on the number of expected output values, suited for Classification problems
- $\sum_{i=1}^{n} value[i]=1$

For example, in the car insurance case
We say:

- `Red = 0`
- `Yellow = 1`
- `Green = 2`

A result like **`[0.01, 0.95, 0.04]`**
is **`red=1%, yellow=95%, green=4%`**

Loss

- That is ***THE*** thing leading to an accurate model
  - The loss is the difference between the reality and what the model comes up with
  - Finding the minimal loss ***is the goal*** of the training
  - The smallest the loss, the best the model
    - Obviously!
- Loss can be set for
  - Training data
  - Test data

Demo
TensorFlow Playground

How does the Network learn?

Minimizing the Loss

- For each tuning position of the network (weights, bias, for *all* neurons in *all* layers)
- Compare the calculated result with the expected one
- Use Mean Square Error (MSE) to magnify the loss
- Plot it against epoch
- Find the spot where loss is the smallest
- See here

Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Training a TensorFlow model

Let’s see some code

Car Insurance Demo:

- Jupyter notebook, training a model
  - Setting up the network requires some… intuition
- Convert model to TensorFlowJS
  - `tensorflowjs_converter --input_format keras ./insurance.h5 ./tfjs`
- Using the model in “Production” (Web UI)

Closer to the real world

Hand-written character recognition

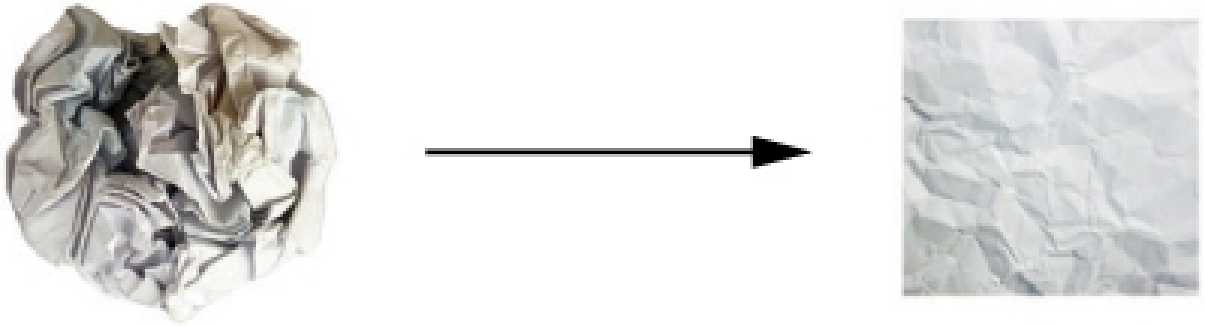
- Now we've seen the bits and pieces, let's put them at work
- We are going to ***train*** a model to recognize hand-written digits, in [0..9]
- We will use a webcam to "see" a notepad
- Take a snapshot, ***prepare*** the data returned by the camera
- Send them to the ***model*** to ***predict*** what digit it is
- Let's go

Hand-written character recognition

- We need to “see” the user writing on the notepad
- Introducing OpenCV

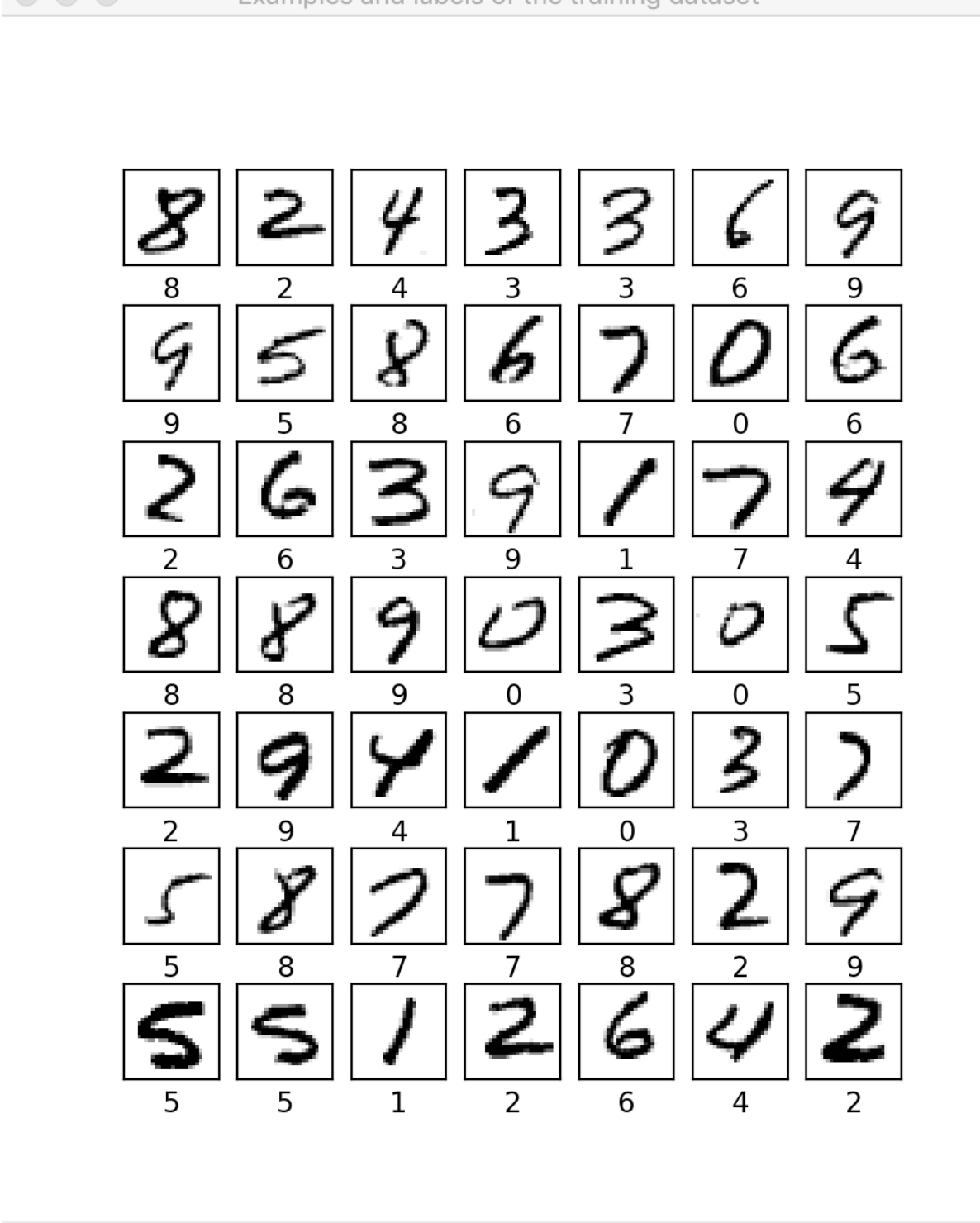
Hand-written character recognition

- A word about data preparation
- It is like unraveling the data



Hand-written character recognition

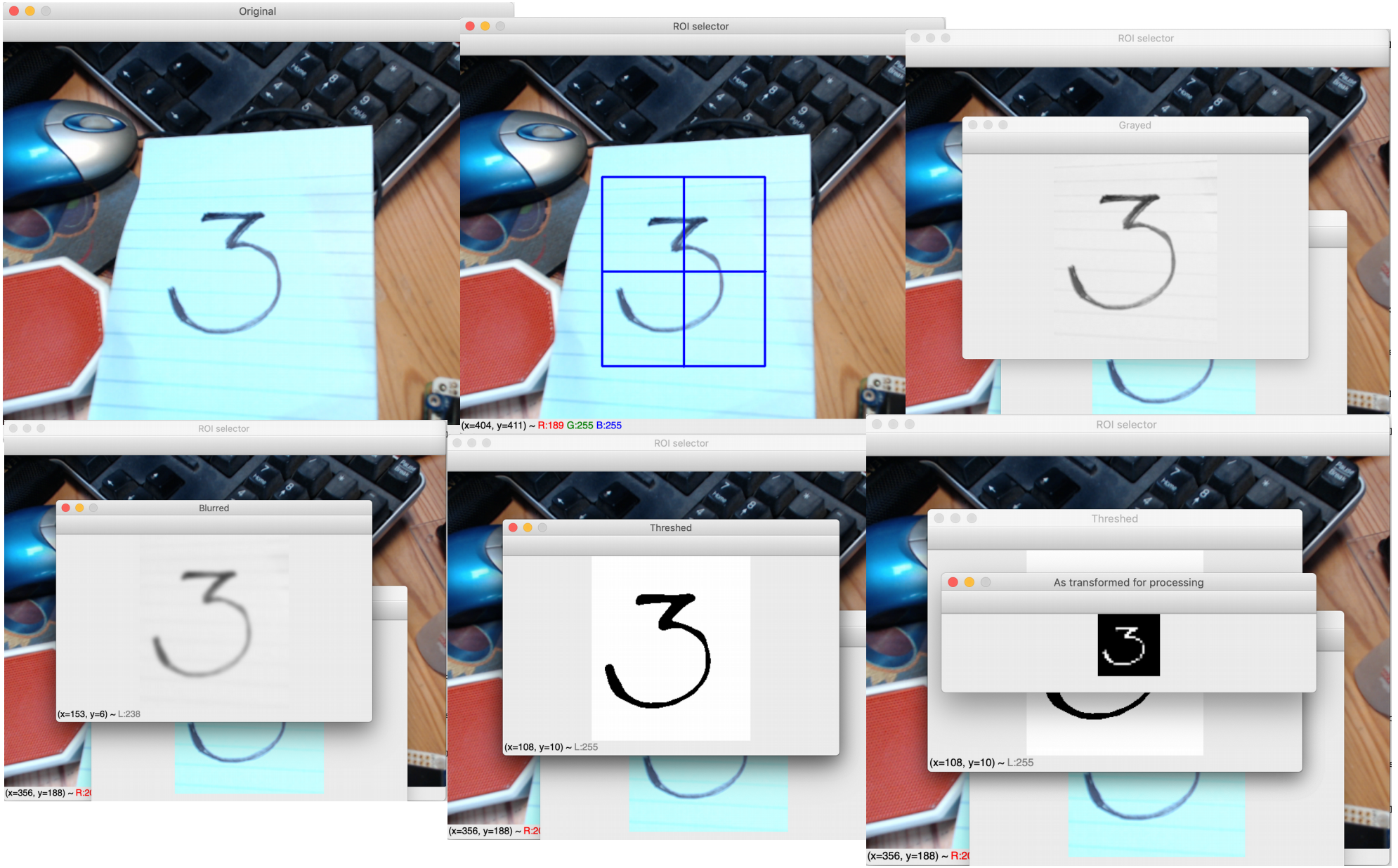
- The model will be trained with B&W images, 28x28 pixels
- This is the **`mnist`** data set, that comes with Keras


Examples and labels of the training dataset
8 2 4 3 3 6 9
9 5 8 6 7 0 6
2 6 3 9 1 7 4
8 8 9 0 3 0 5
2 9 4 1 0 3 7
5 8 7 7 8 2 9
5 5 1 2 6 4 2

Hand-written character recognition

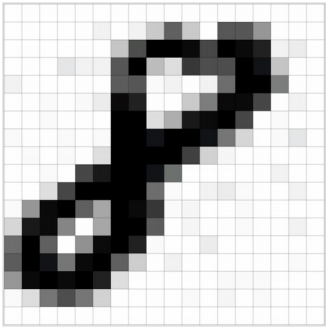
- Using OpenCV, we will
  - Take a snapshot
  - Select a Region Of Interest (ROI)
  - Gray it (no color)
  - Blur it to get rid of potential background
  - Thresh it (minimize number of colors)
  - Resize it as expected by the model

Hand-written character recognition



Hand-written character recognition

- Data preparation



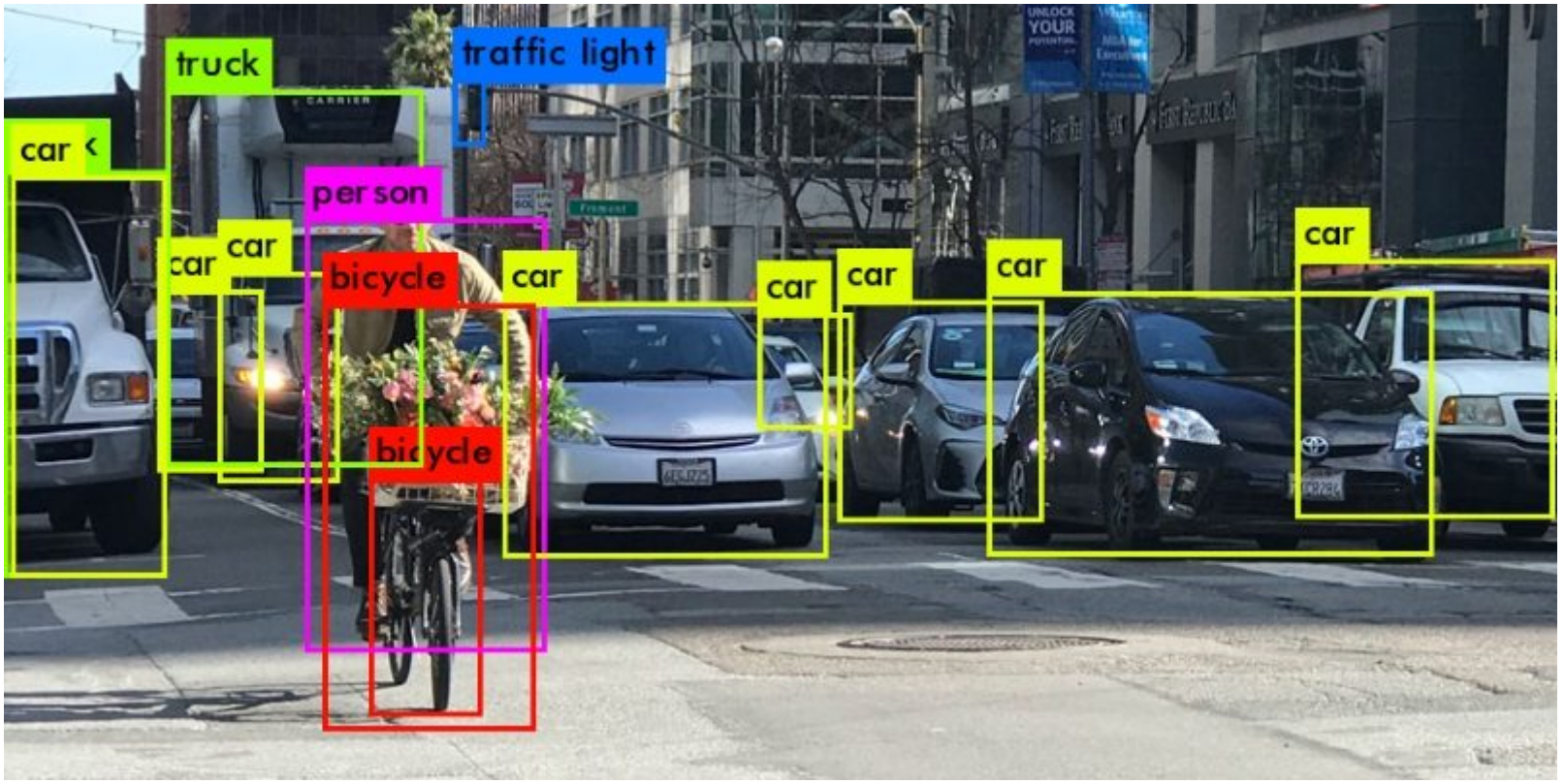
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

=

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
 0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Hand-Written Digits Demo

Even closer to the real world: Self driving cars



- Detect objects around
  - cars, bikes, pedestrians, traffic lights, curbs, etc
- Classify them, and make the right decision
  - brake, stop, slow down, turn right, etc

Final Bonus.

59 seconds

Recommended reading

- Manning's Deep Learning Crash Course
  - by Oliver Zeigermann
- Manning's Grokking Deep Learning
  - by Andrew Trask
- Manning's Deep Learning with Python
  - by François Chollet

- All the material on github.