First steps to deep-learning

From the perceptron to the deep generative models

Nicolas Desassis

nicolas.desassis@minesparis.psl.eu

January 2022

Sources

- Deep Learning course for MINES ParisTech
  (E. Decenciere, T. Walter and S. Velasco-Forero)
- Stanford MOOC on machine learning (Andrew Ng)
- Leçon inaugurale de Y. Le Cun au Collège de France

Sources

- Deep Learning course for MINES ParisTech
  (E. Decenciere, T. Walter and S. Velasco-Forero)
- Stanford MOOC on machine learning (Andrew Ng)
- Leçon inaugurale de Y. Le Cun au Collège de France
- Shalev-Shwartz and Ben-David (2014)

Introduction

- Deep-learning is the part of machine-learning dealing with neural-networks

Introduction

- Deep-learning is the part of machine-learning dealing with neural-networks
- A neural-network (NN) is a functions $f_{\mathbf{W}}$ from $\mathbb{R}^n$ to (a subset of) $\mathbb{R}^p$ parameterized with a large number of parameters (the weights, denoted $\mathbf{W}$).

Introduction

- Deep-learning is the part of machine-learning dealing with neural-networks
- A neural-network (NN) is a functions $f_{\mathbf{W}}$ from $\mathbb{R}^n$ to (a subset of) $\mathbb{R}^p$ parameterized with a large number of parameters (the weights, denoted $\mathbf{W}$).
- In genereral, NN have a high-capacity or a great expressive power given that the number of weights is important (see later)

Introduction

- Deep-learning is the part of machine-learning dealing with neural-networks
- A neural-network (NN) is a functions $f_{\mathbf{W}}$ from $\mathbb{R}^n$ to (a subset of) $\mathbb{R}^p$ parameterized with a large number of parameters (the weights, denoted $\mathbf{W}$).
- In genereral, NN have a high-capacity or a great expressive power given that the number of weights is important (see later)
- Thank to generic algorithms (gradient based), NN can help to solve problem expressed as optimization of a functional:

$$\underset{f \in \mathcal{E}}{\operatorname{argmin}}\ T(f) \simeq \underset{\mathbf{W} \in \mathbb{R}^m}{\operatorname{argmin}}\ T(f_{\mathbf{W}})$$

where T is a functional defined over a space $\mathcal{E}$ and $f_{\mathbf{W}}$ is a neural-network parameterized with m weights stored in $\mathbf{W}$

Examples

- Classification

Examples

- Classification
- Compression (or dimension reduction)
  Find an encoder f_e which reduces the dimension of the input (e.g image) and the associated decoder f_d to transform back. The pair (f_e, f_d) has to minimize the reconstruction error.

Examples

- Classification
- Compression (or dimension reduction)
  Find an encoder f_e which reduces the dimension of the input (e.g image) and the associated decoder f_d to transform back. The pair (f_e, f_d) has to minimize the reconstruction error.
- Spatio-temporal prediction
  Example: weather forecasting from satellite images

Examples

- Classification
- Compression (or dimension reduction)
  Find an encoder f_e which reduces the dimension of the input (e.g image) and the associated decoder f_d to transform back. The pair (f_e, f_d) has to minimize the reconstruction error.
- Spatio-temporal prediction
  Example: weather forecasting from satellite images
- Partial Differential Equation resolution

Examples

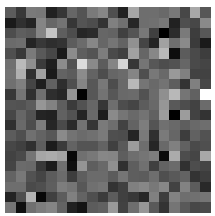
- Classification
- Compression (or dimension reduction)
  Find an encoder f_e which reduces the dimension of the input (e.g image) and the associated decoder f_d to transform back. The pair (f_e, f_d) has to minimize the reconstruction error.
- Spatio-temporal prediction
  Example: weather forecasting from satellite images
- Partial Differential Equation resolution
- Find a random process to generate images with a distribution "close" the one of the training images

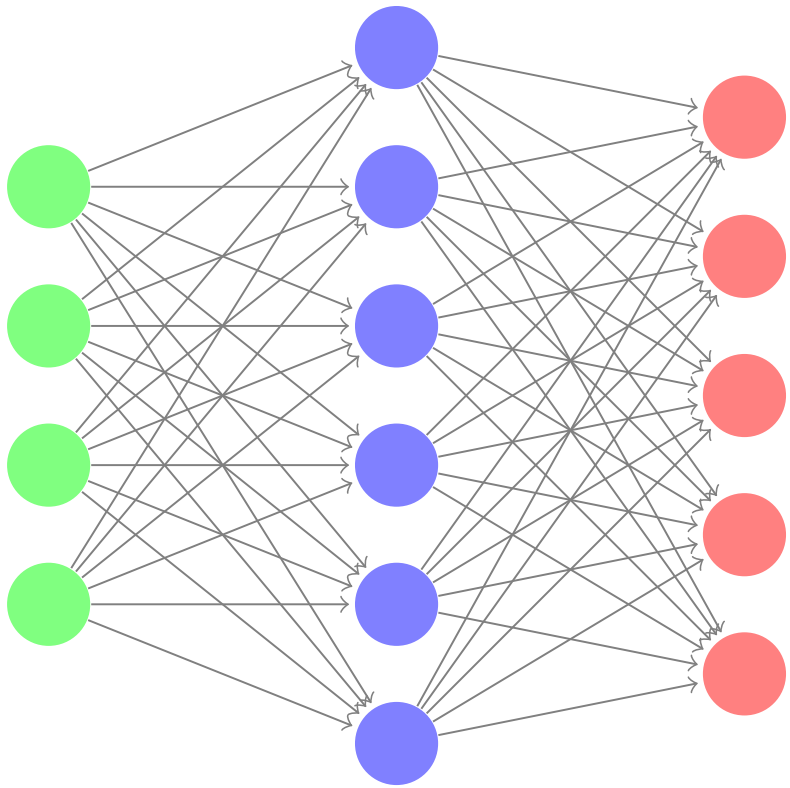
Latent space

Hidden layer

Output layer

NOISE





FAKE

Some applications

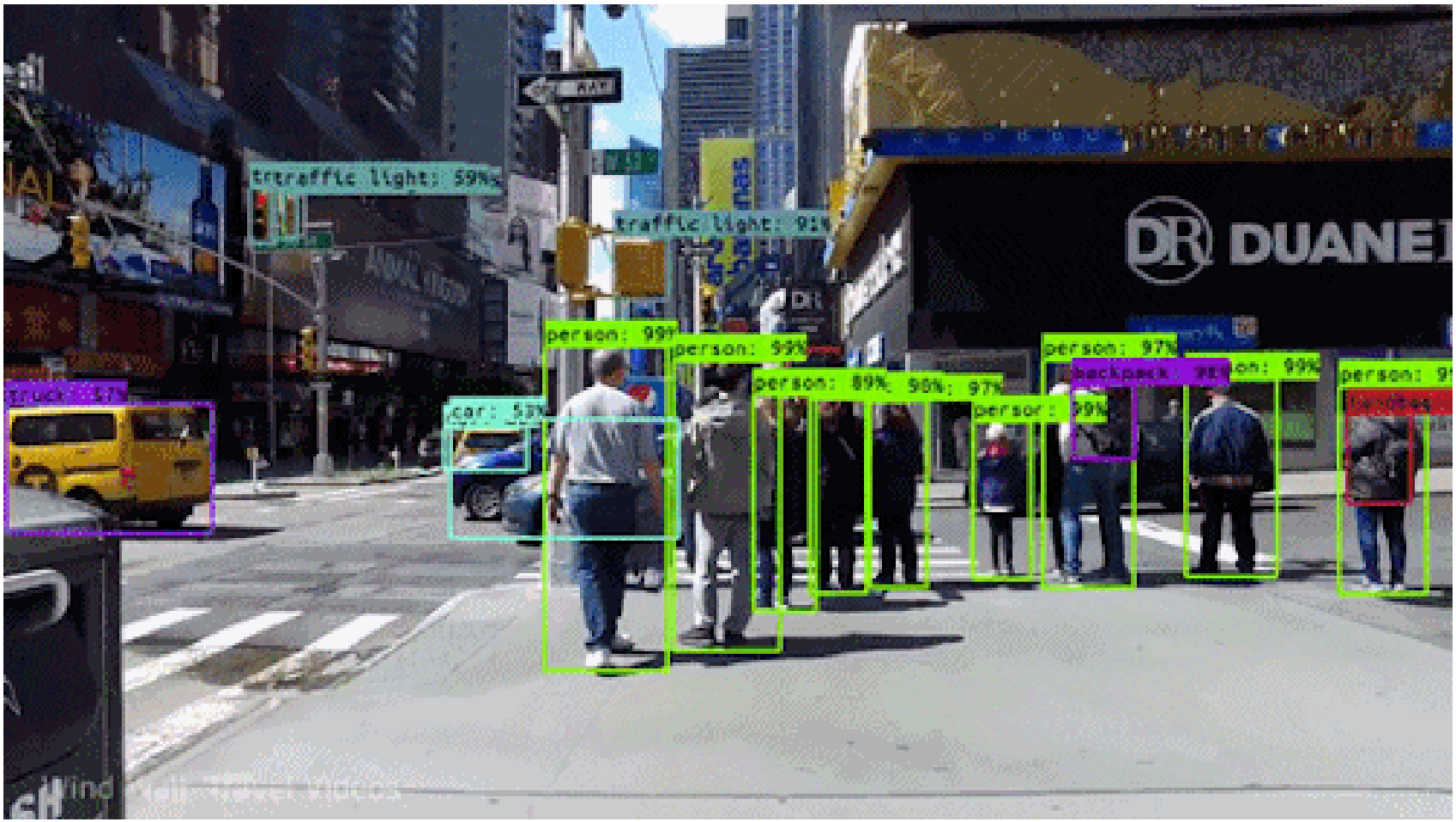
- Computer vision
  - Medical diagnoses
  - Facial recognition
  - Handwritting recognition
  - ...
- Natural langage processing (NLP)
  - Chatbots
  - Automatic translation
  - Spam detection
  - ...
- Data-sciences
  - Predictive maintenance
  - Click prediction
  - Default payement prediction
  - ...
- Automatons
  - Chess and Go
  - Generation of art (music, pictures, text,...)
  - ...
- ...

Some reasons for such a success

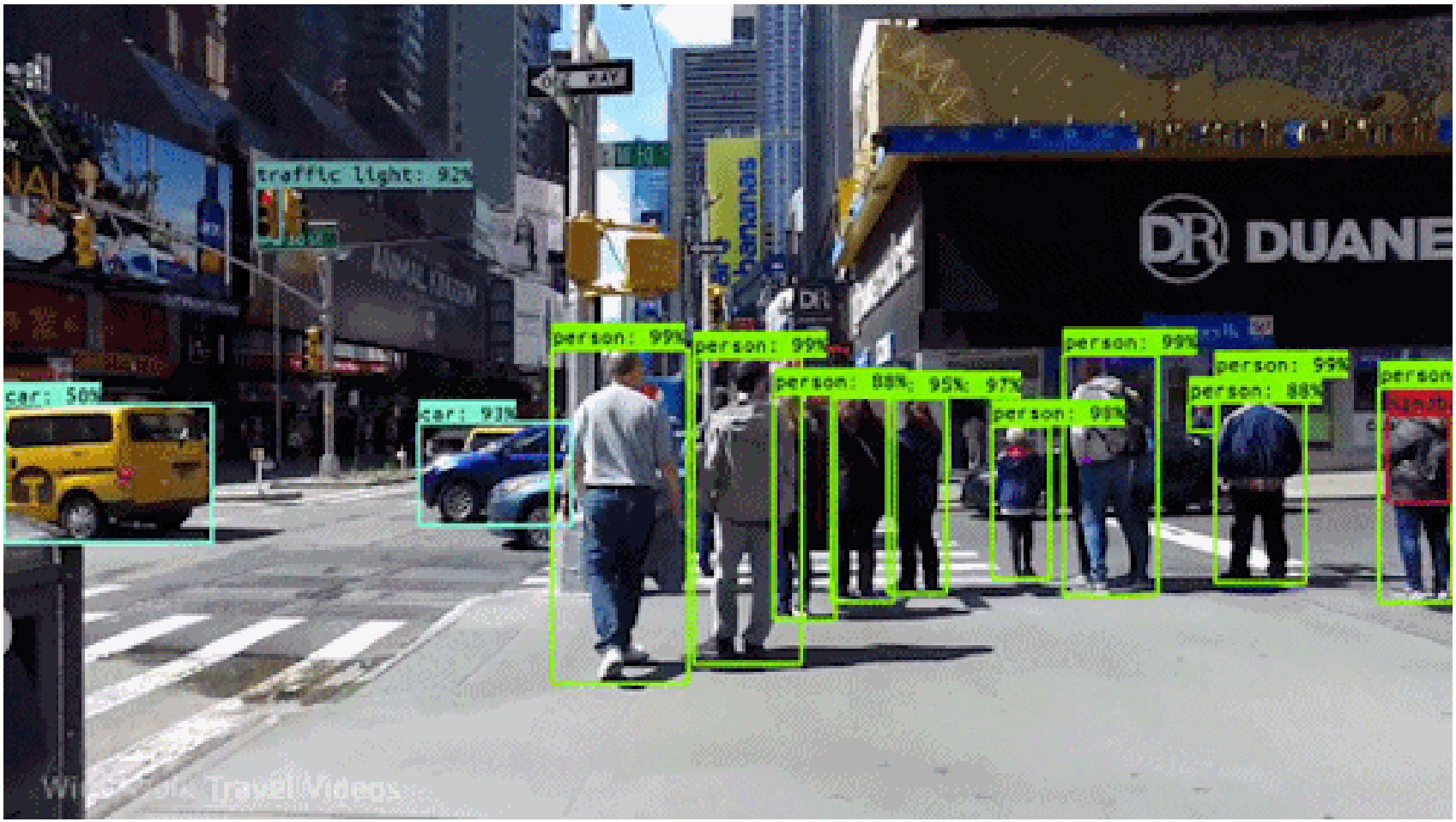
- Big Data
- New algorithms
- Python
- Google & Facebook
  – > TensorFlow, Pytorch
- New computational facilities
  – > cloud, GPU, TPU,...
- Open source codes
- Blogs
- MOOCs
- Impressive successes

Go and Chess player of Deep-Mind

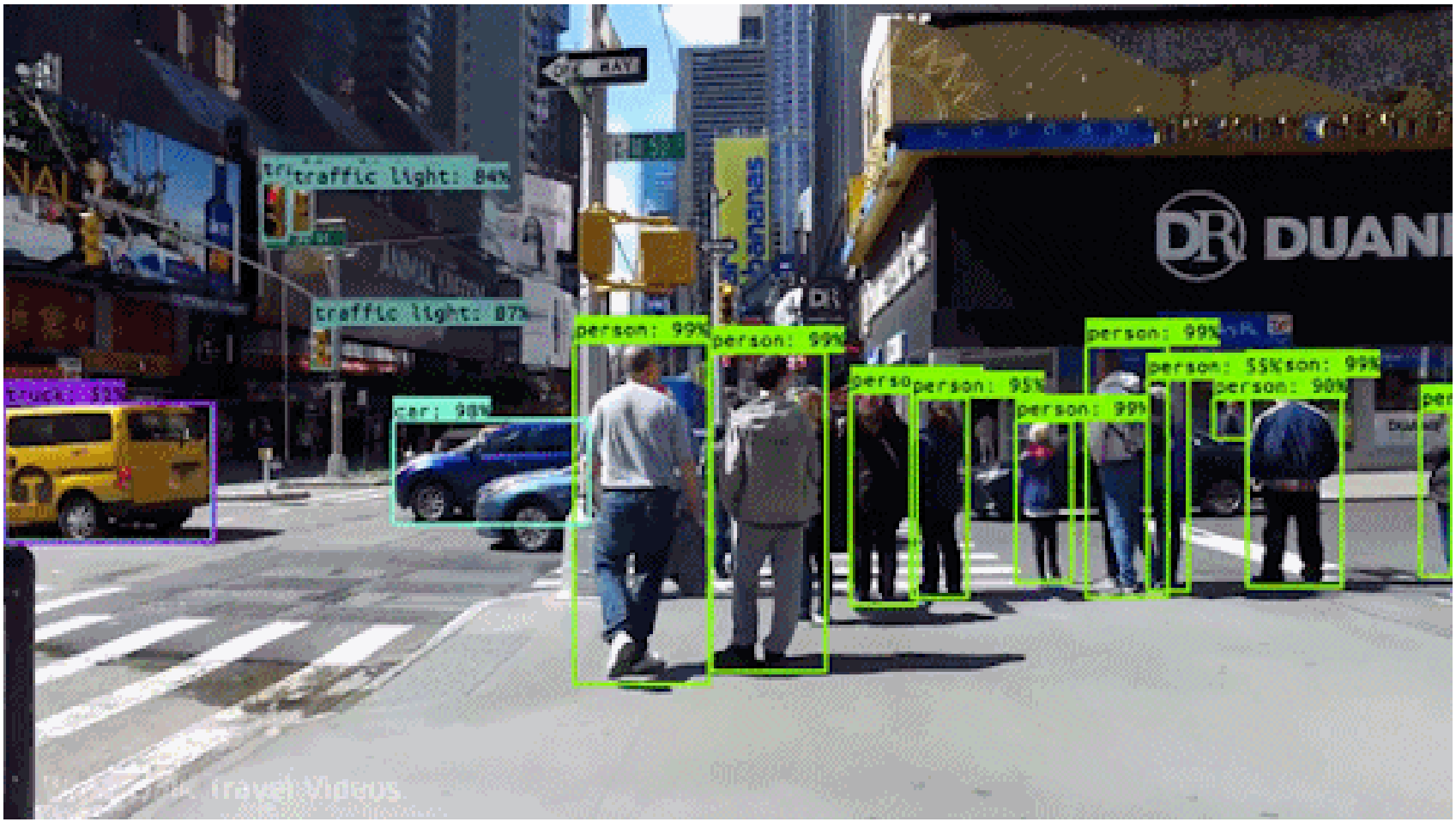
Self-driving car



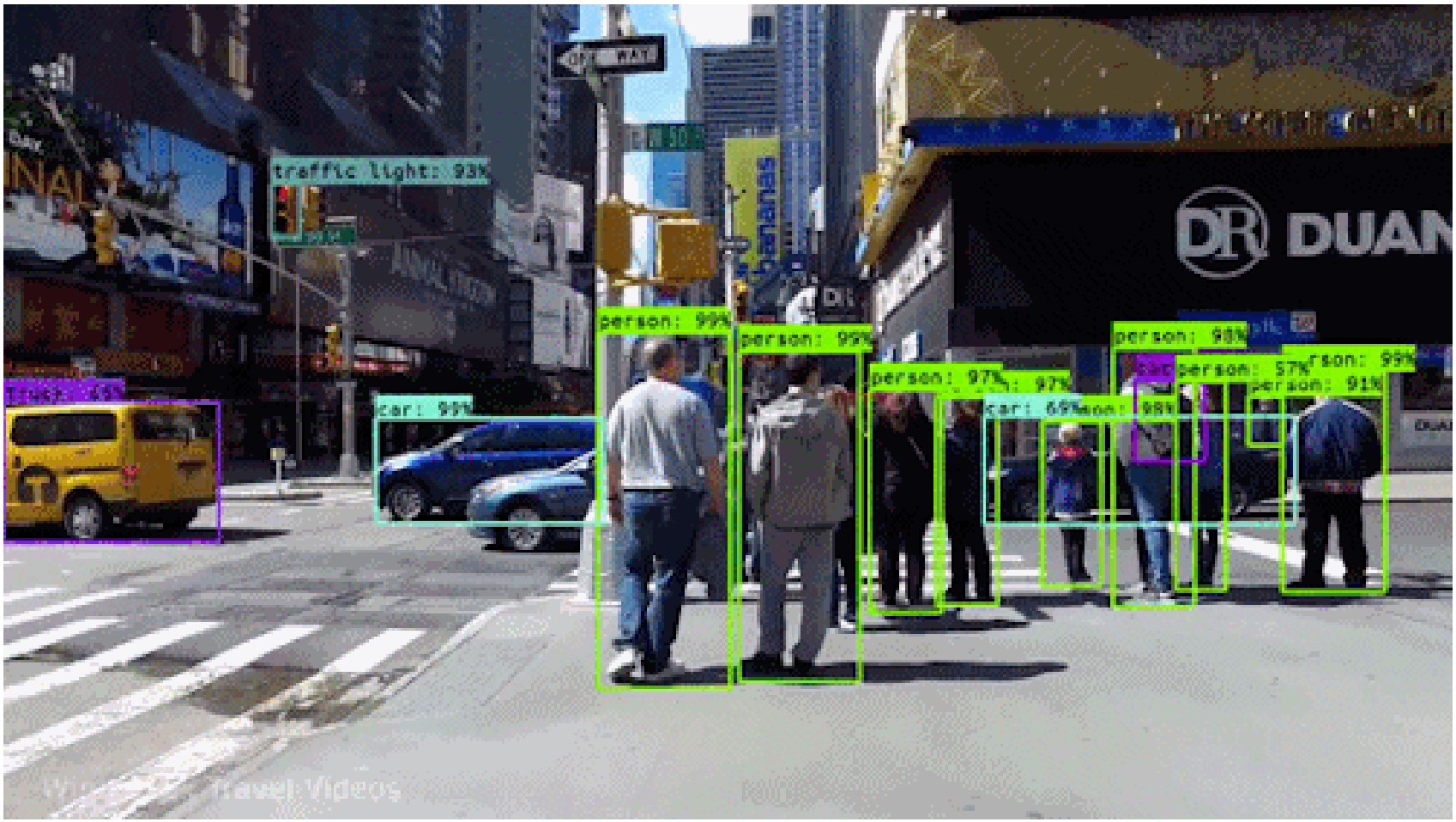
Self-driving car



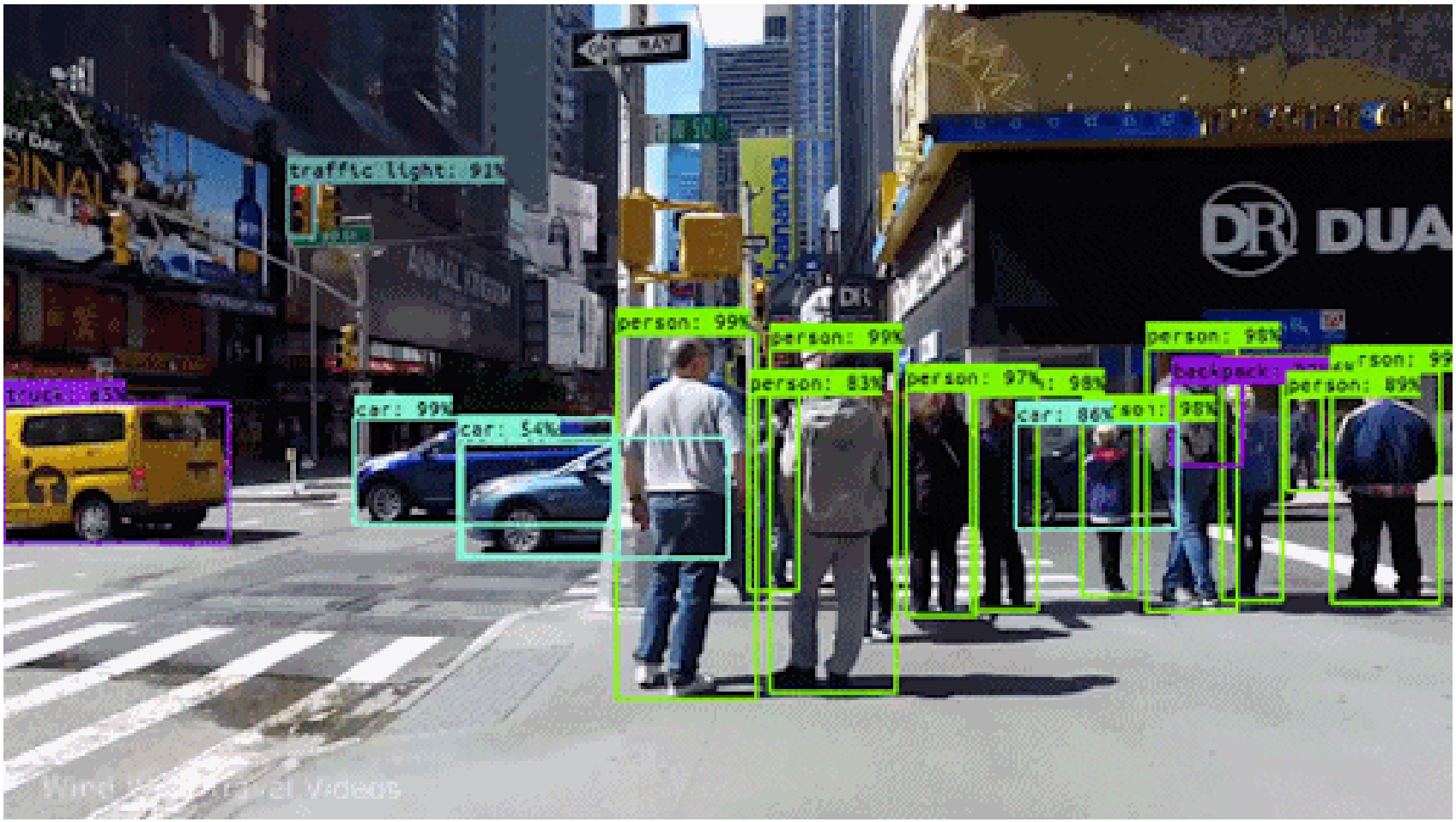
Self-driving car



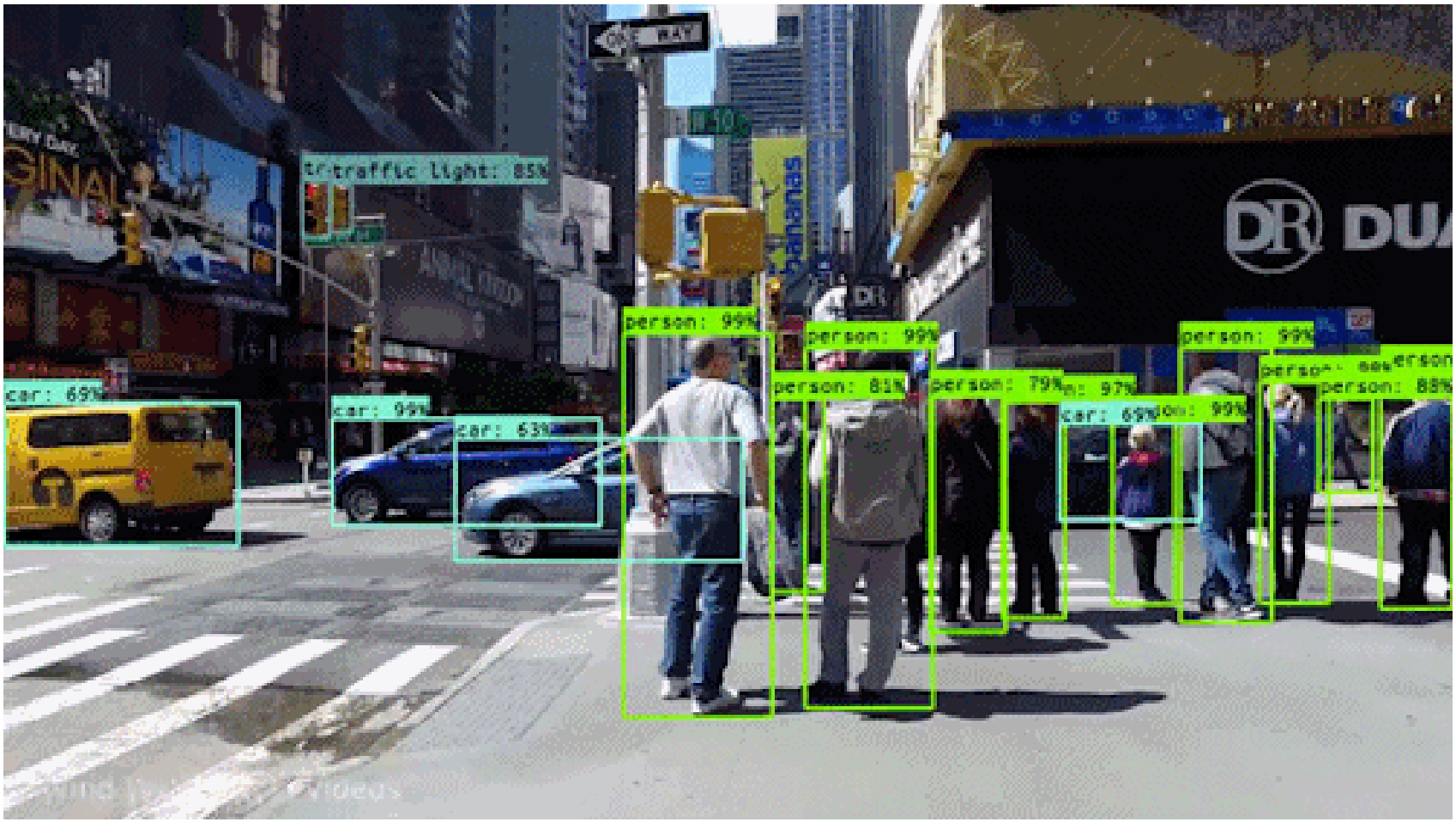
Self-driving car



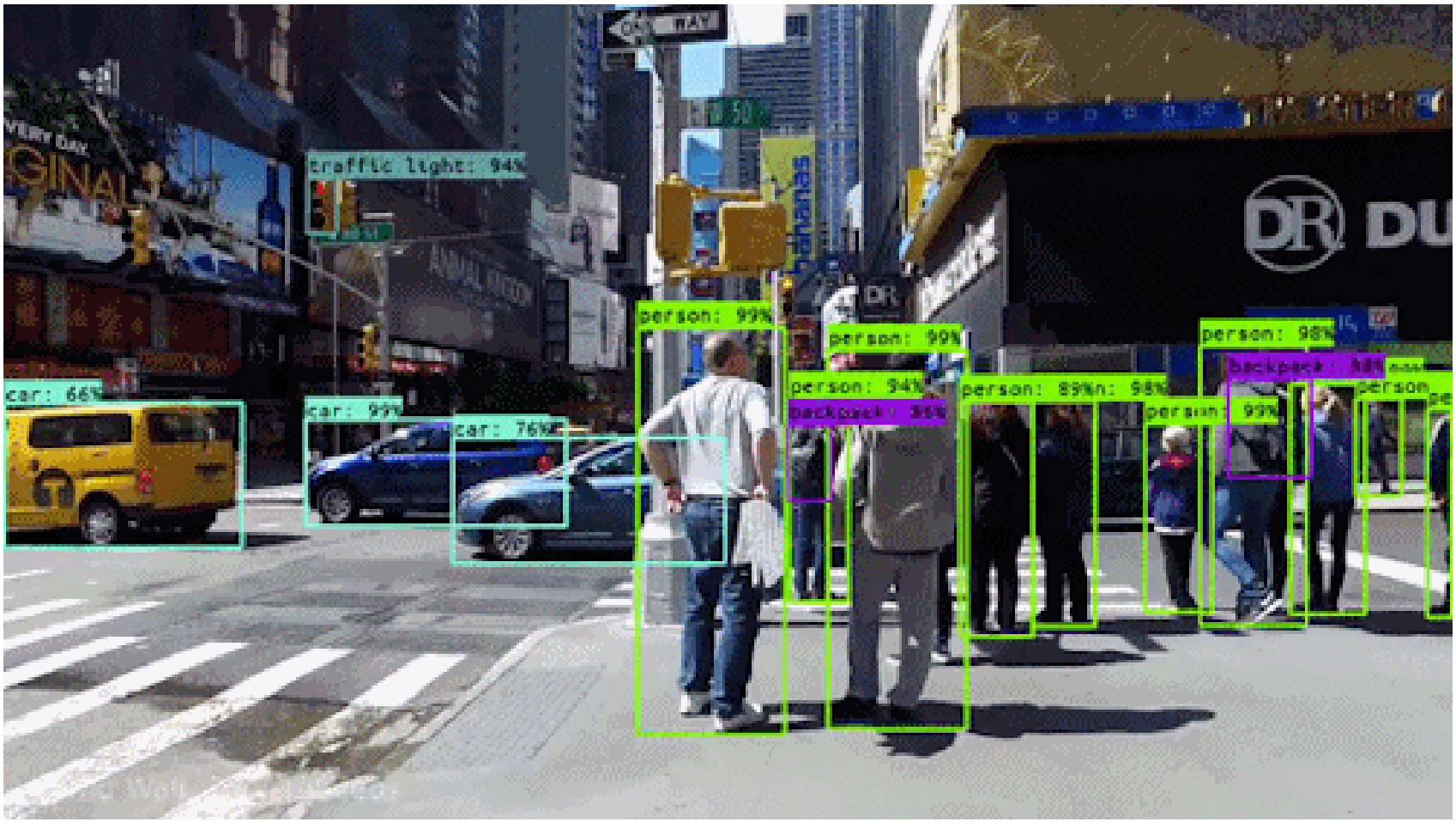
Self-driving car



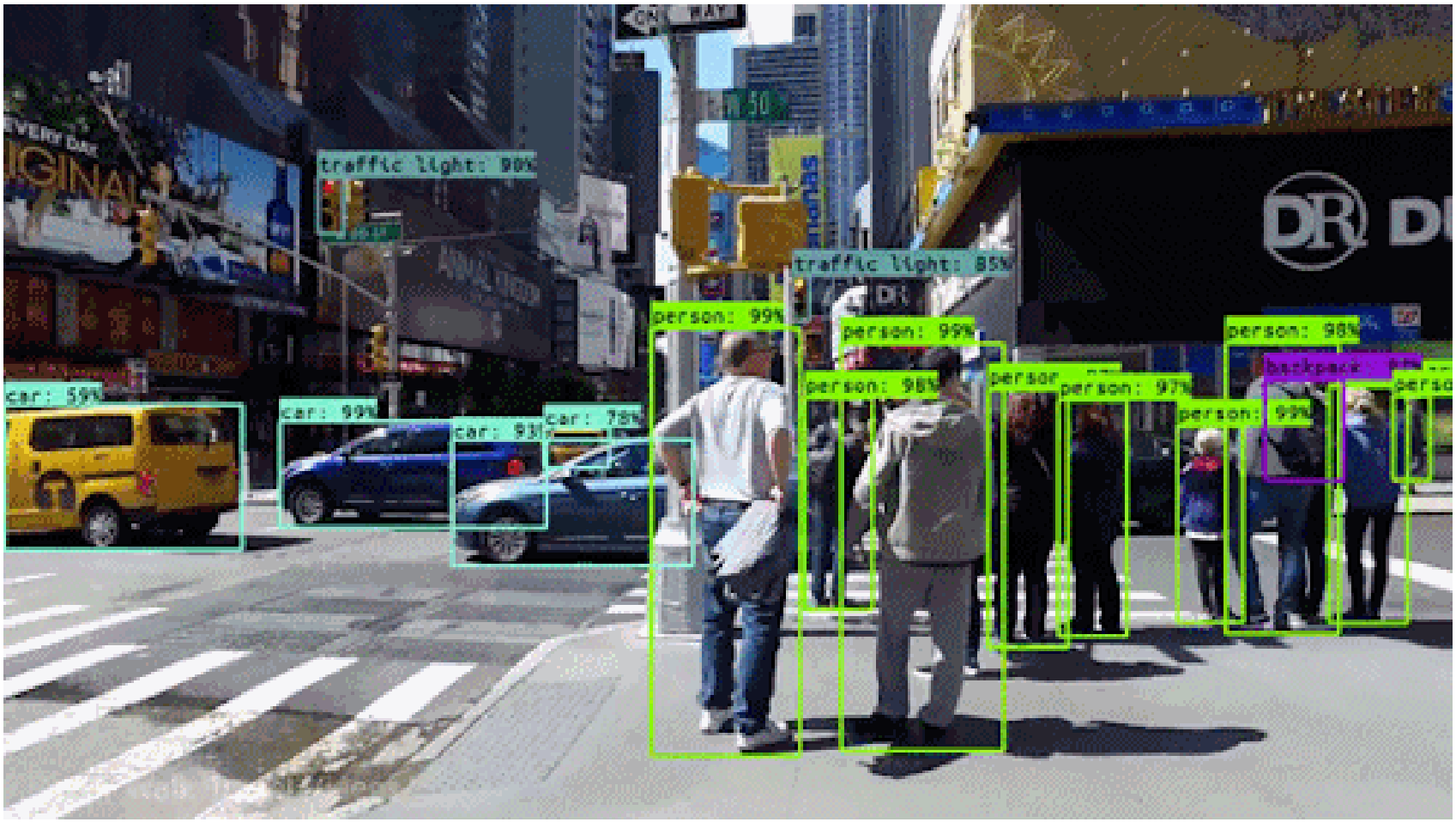
Self-driving car



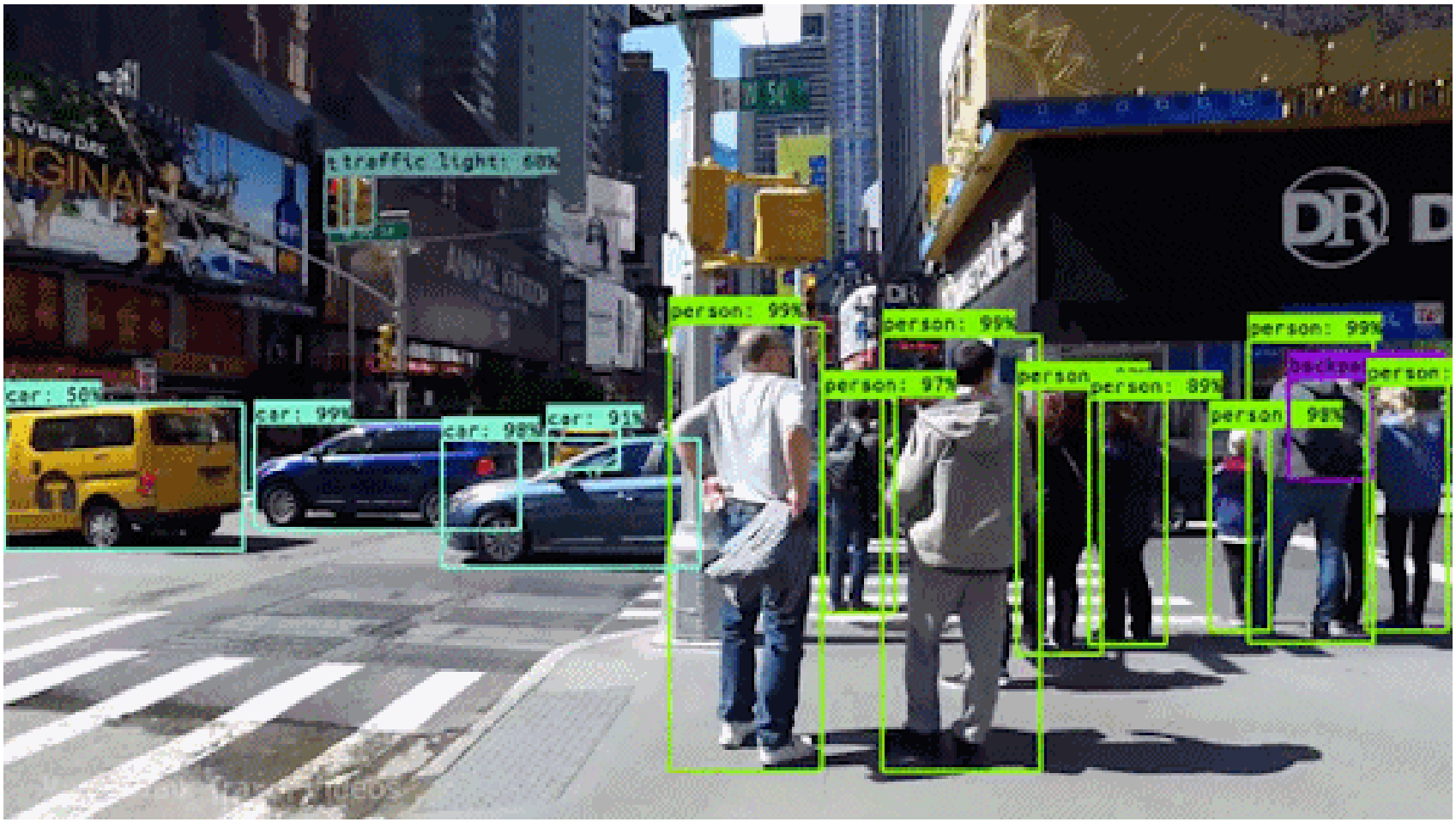
Self-driving car



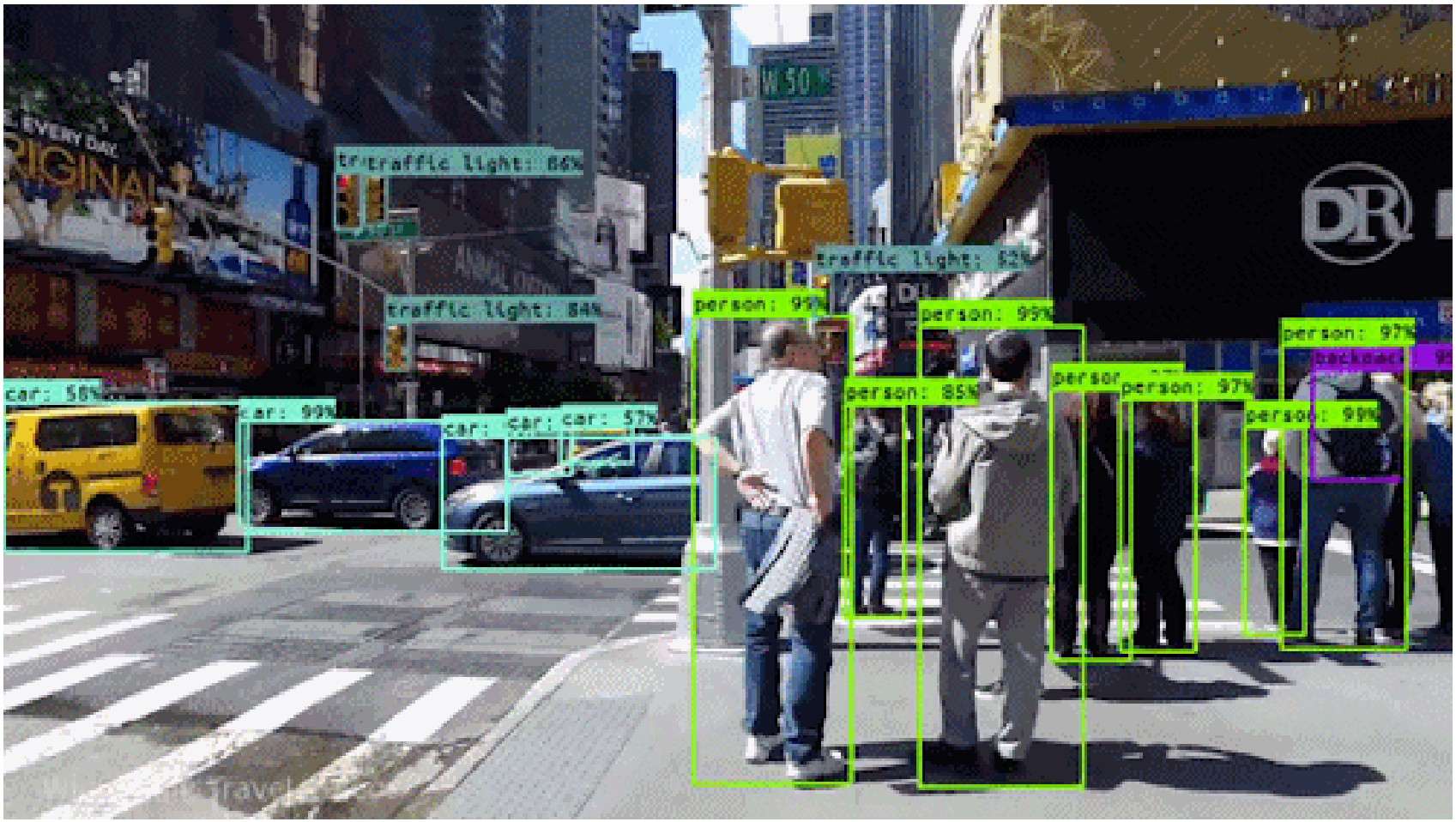
Self-driving car



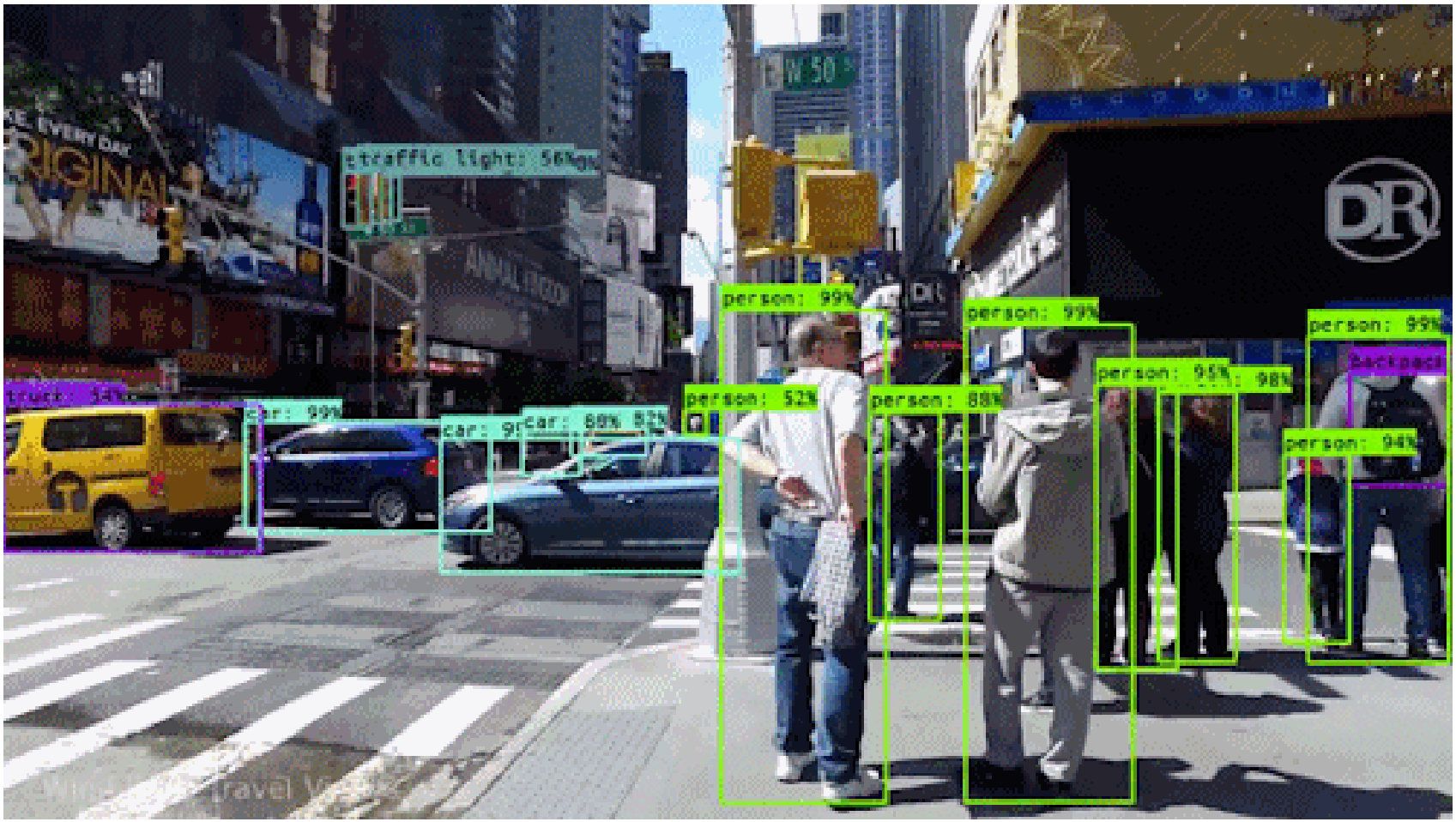
Self-driving car



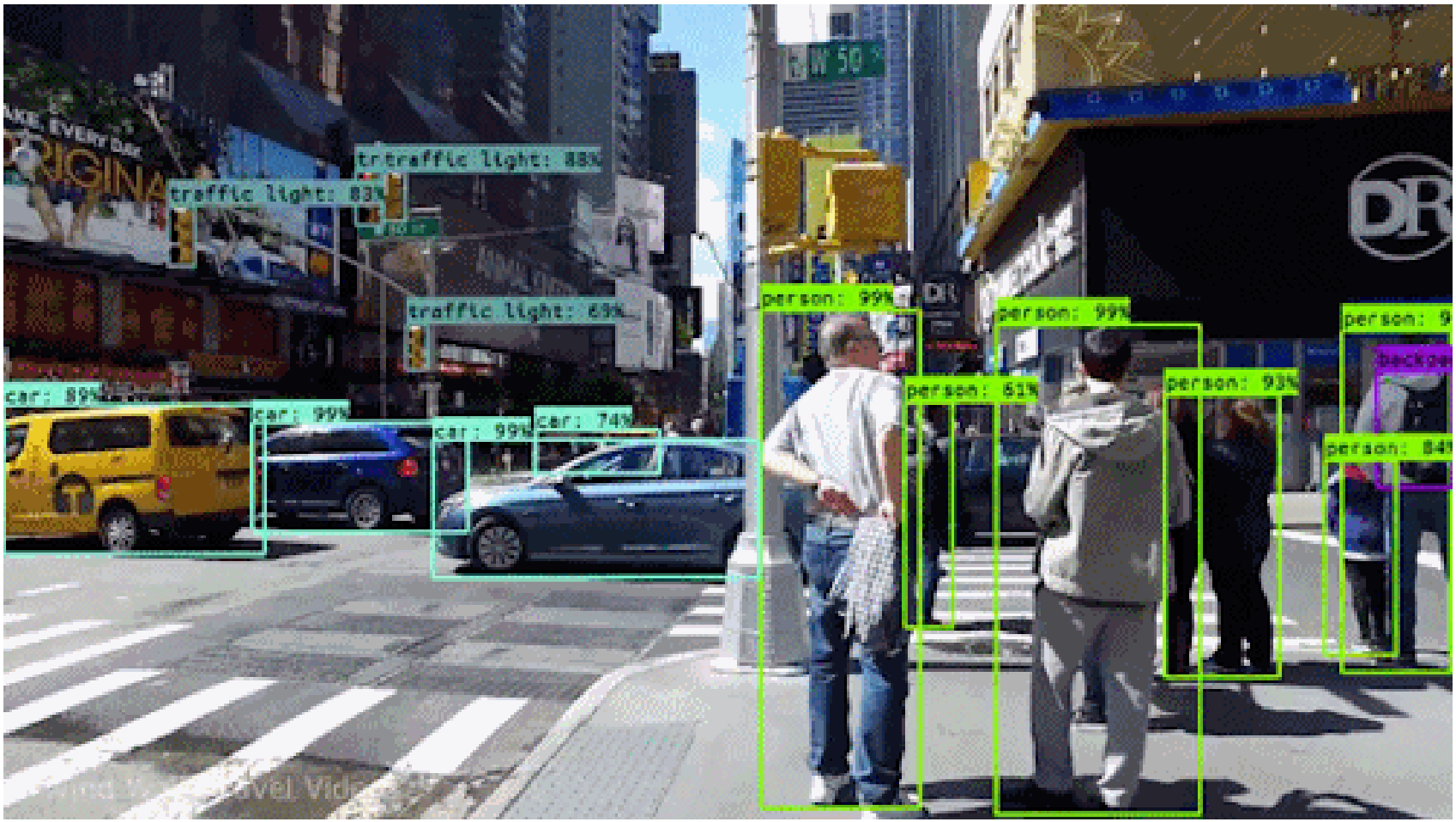
Self-driving car



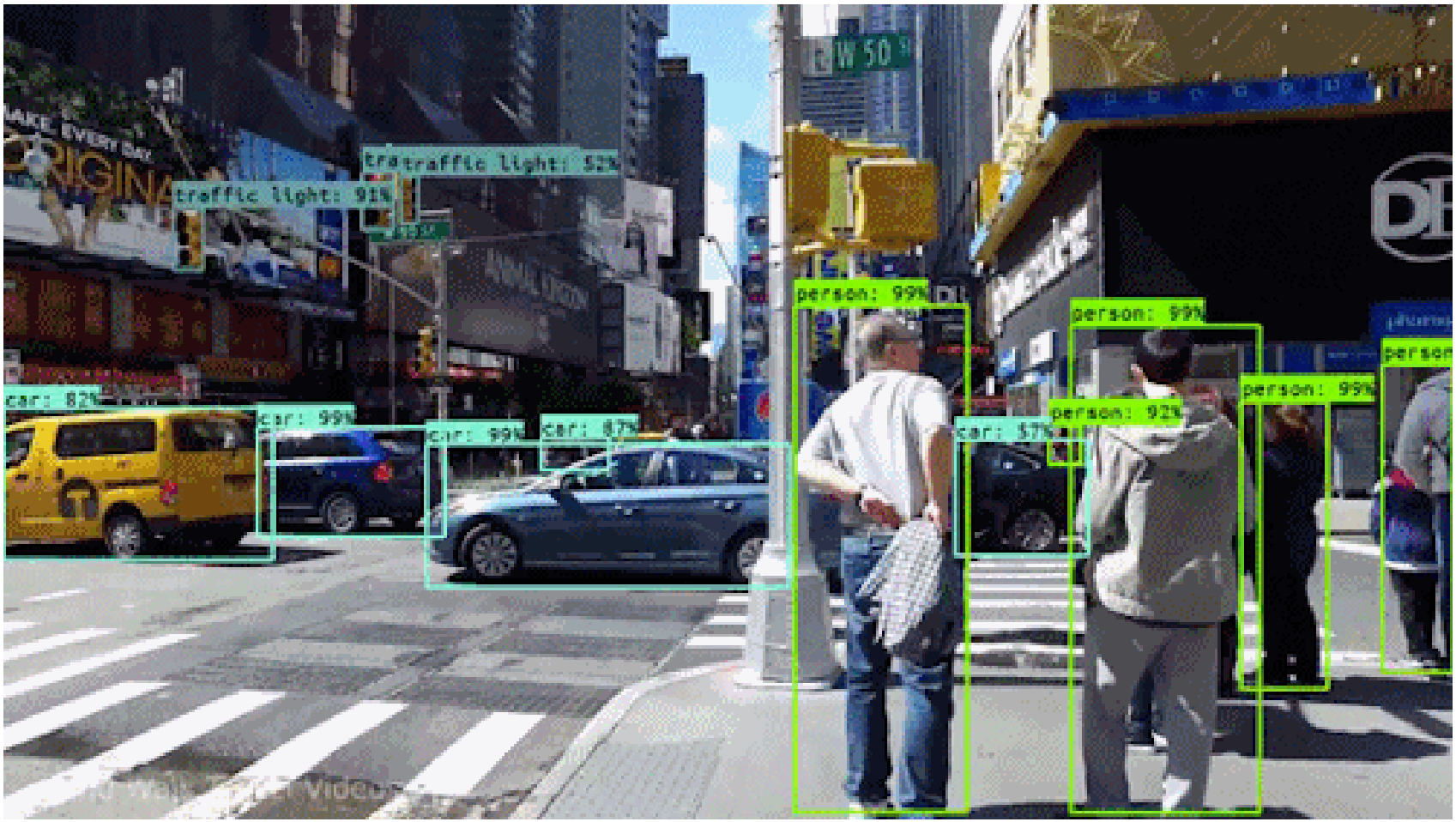
Self-driving car



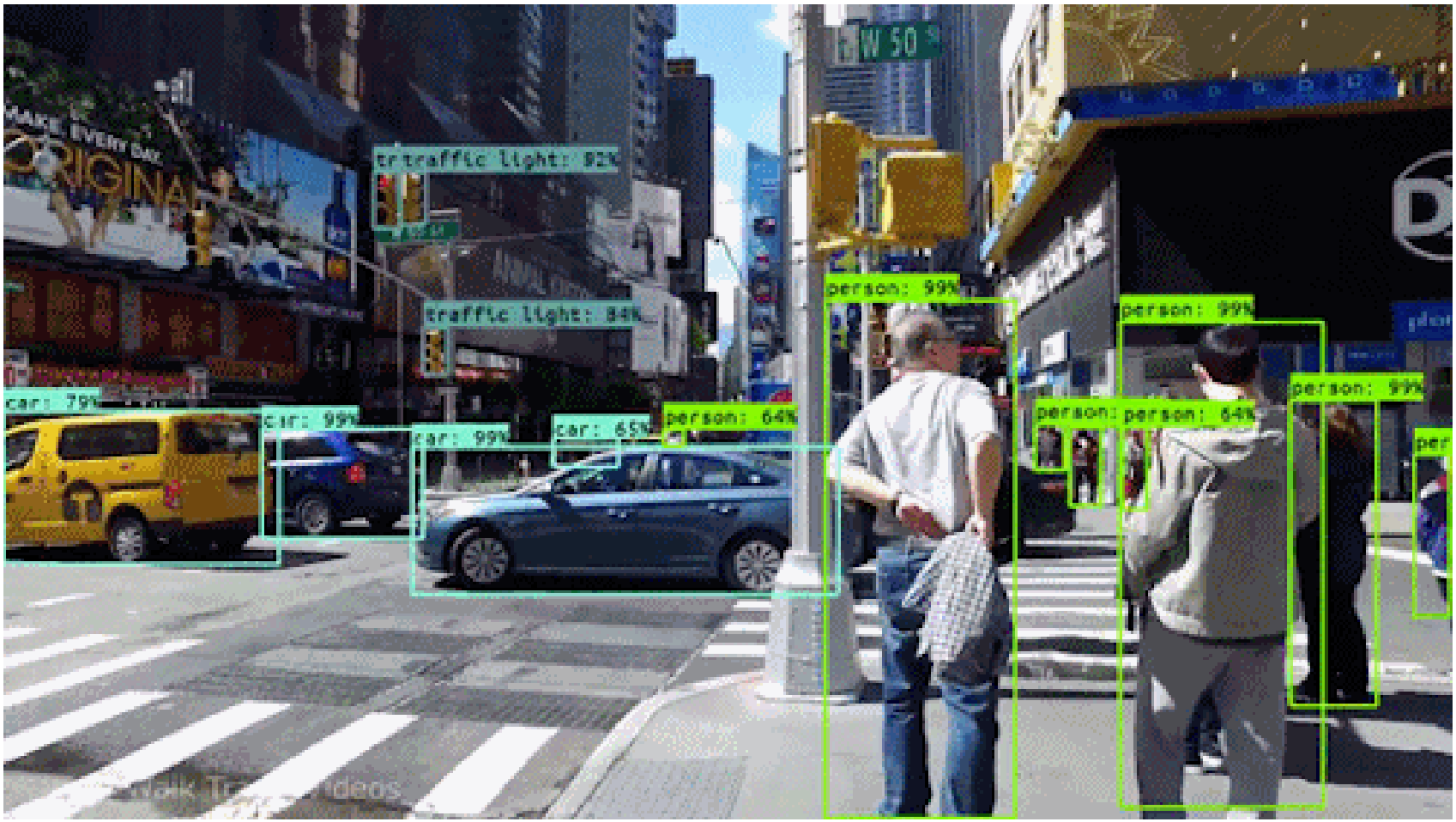
Self-driving car



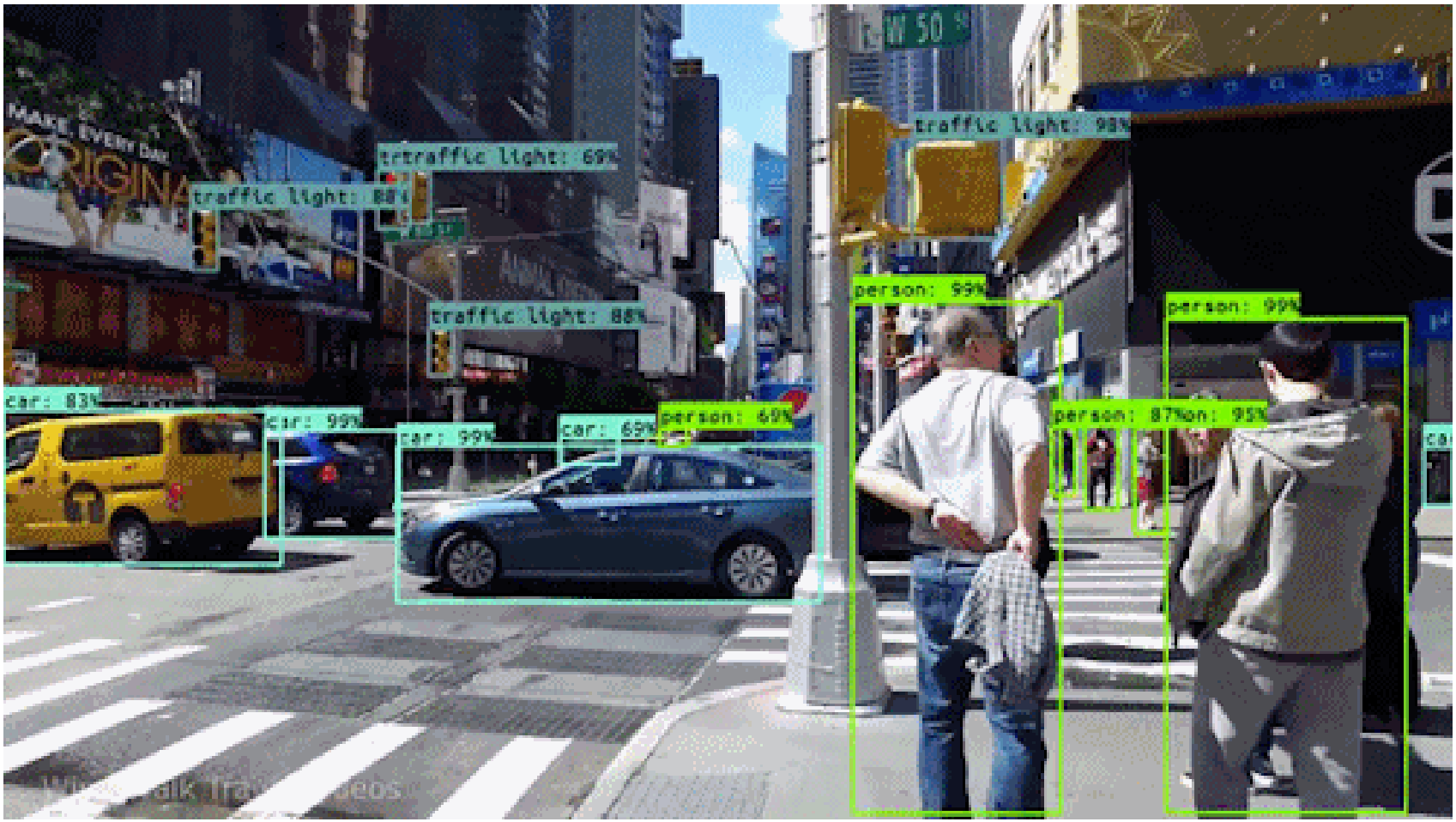
Self-driving car



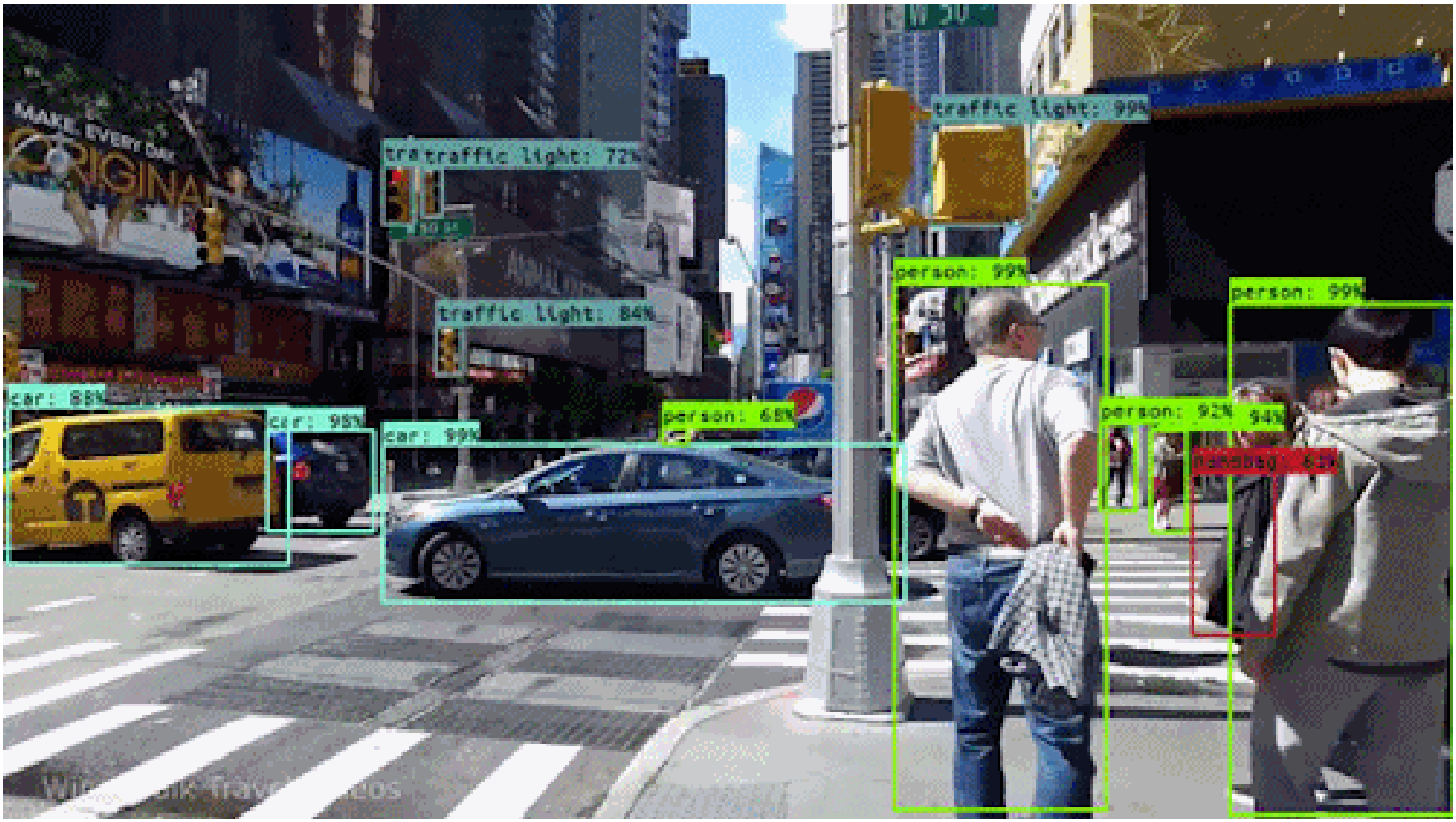
Self-driving car



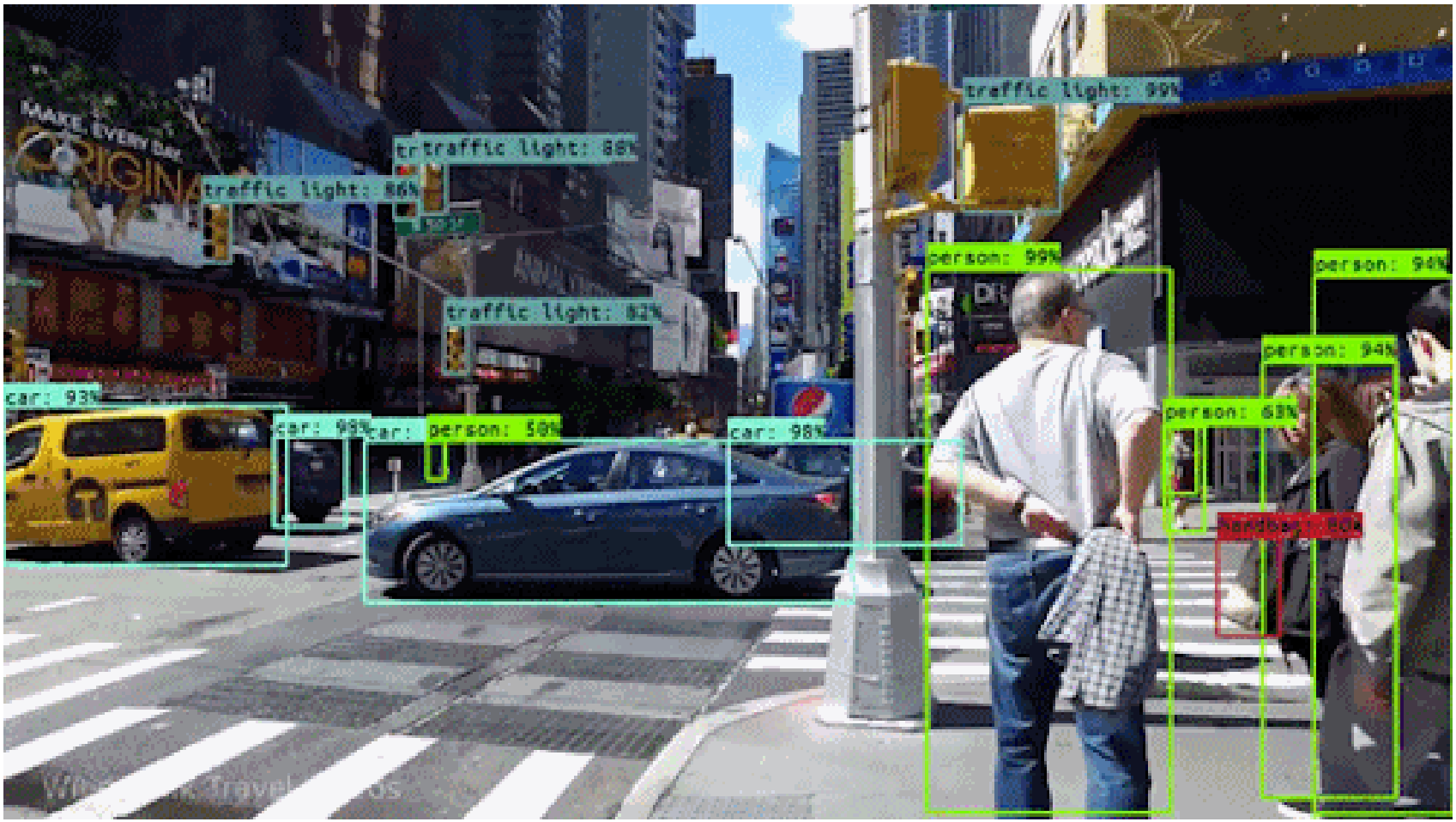
Self-driving car



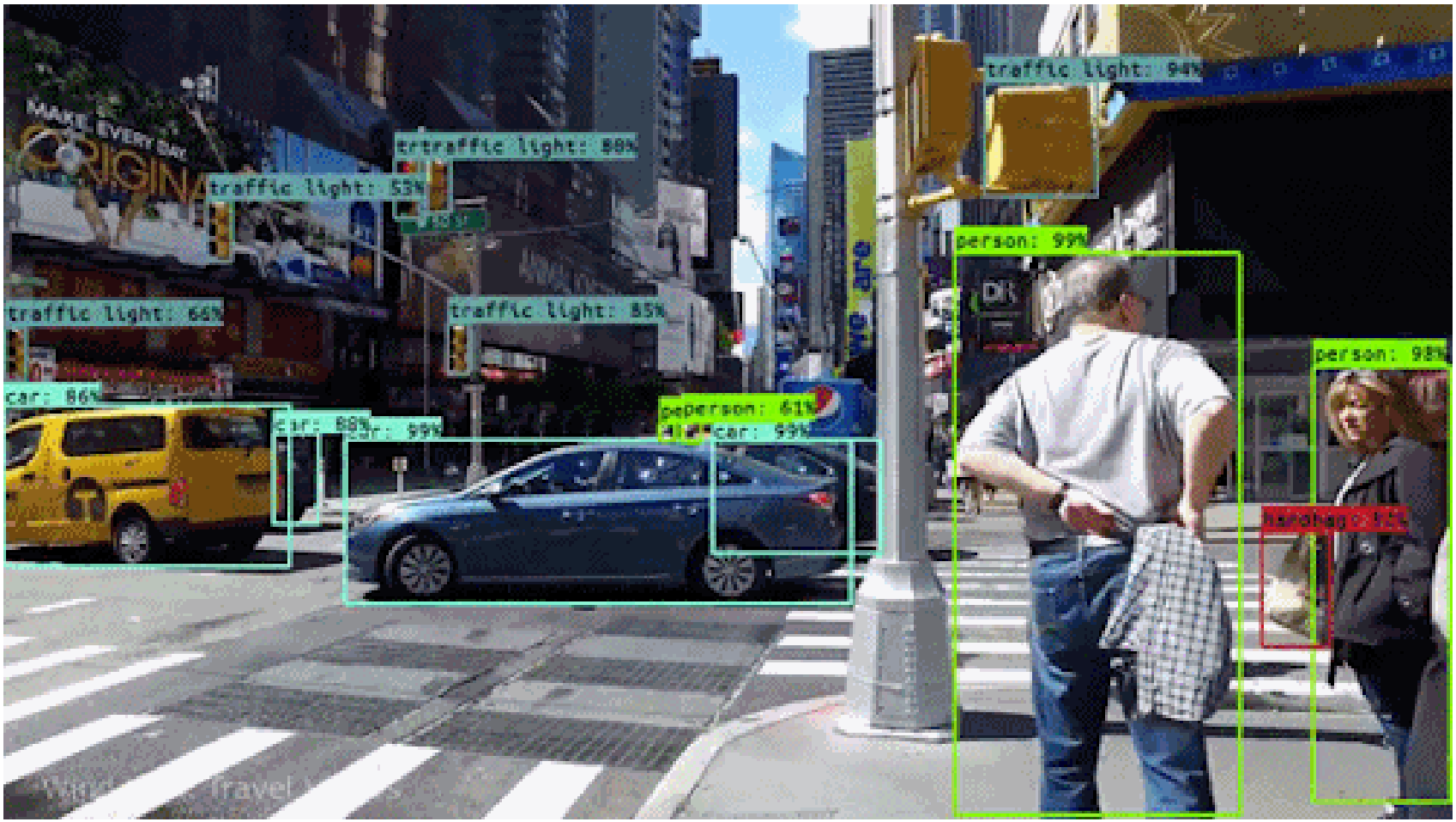
Self-driving car



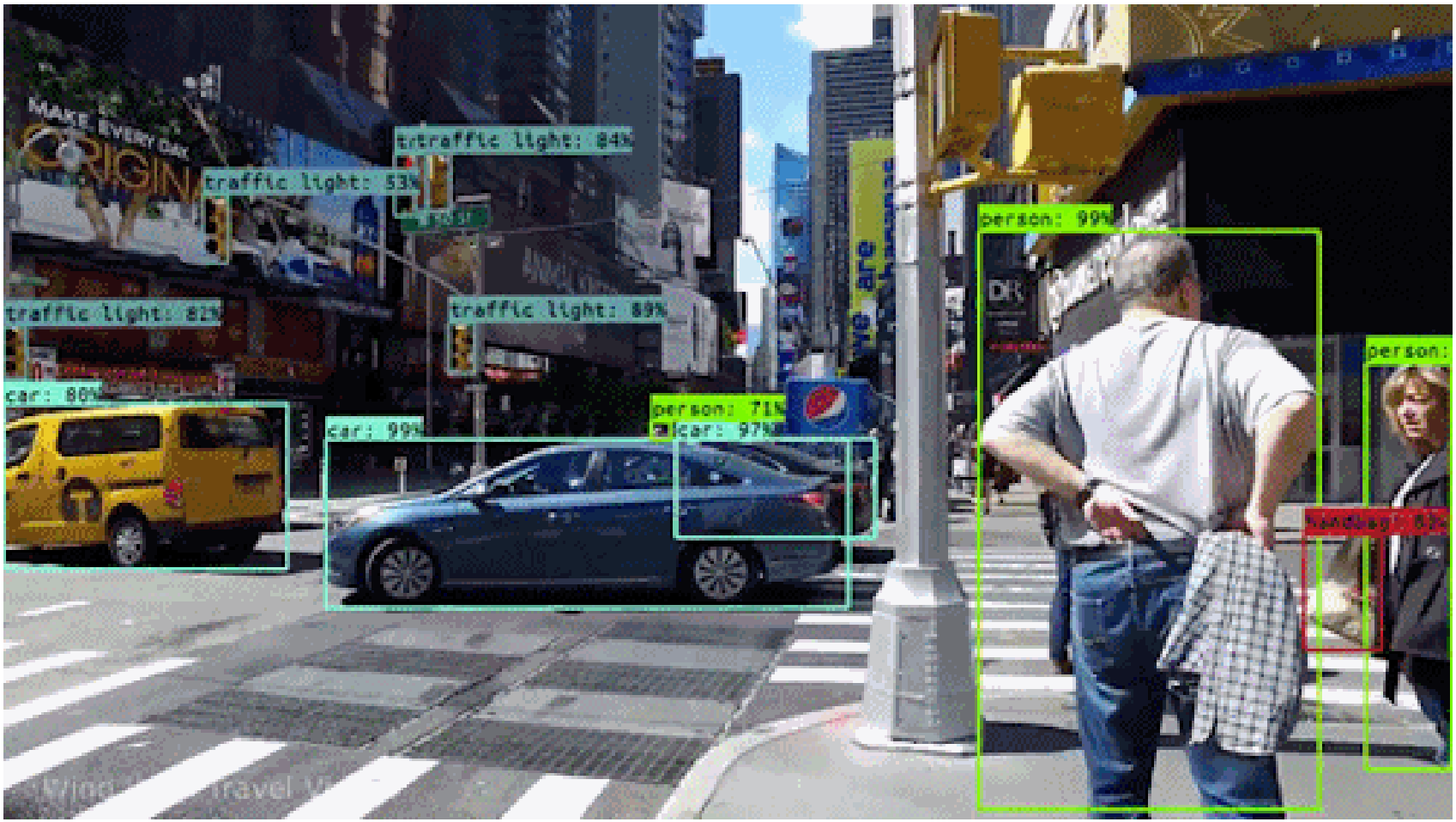
Self-driving car



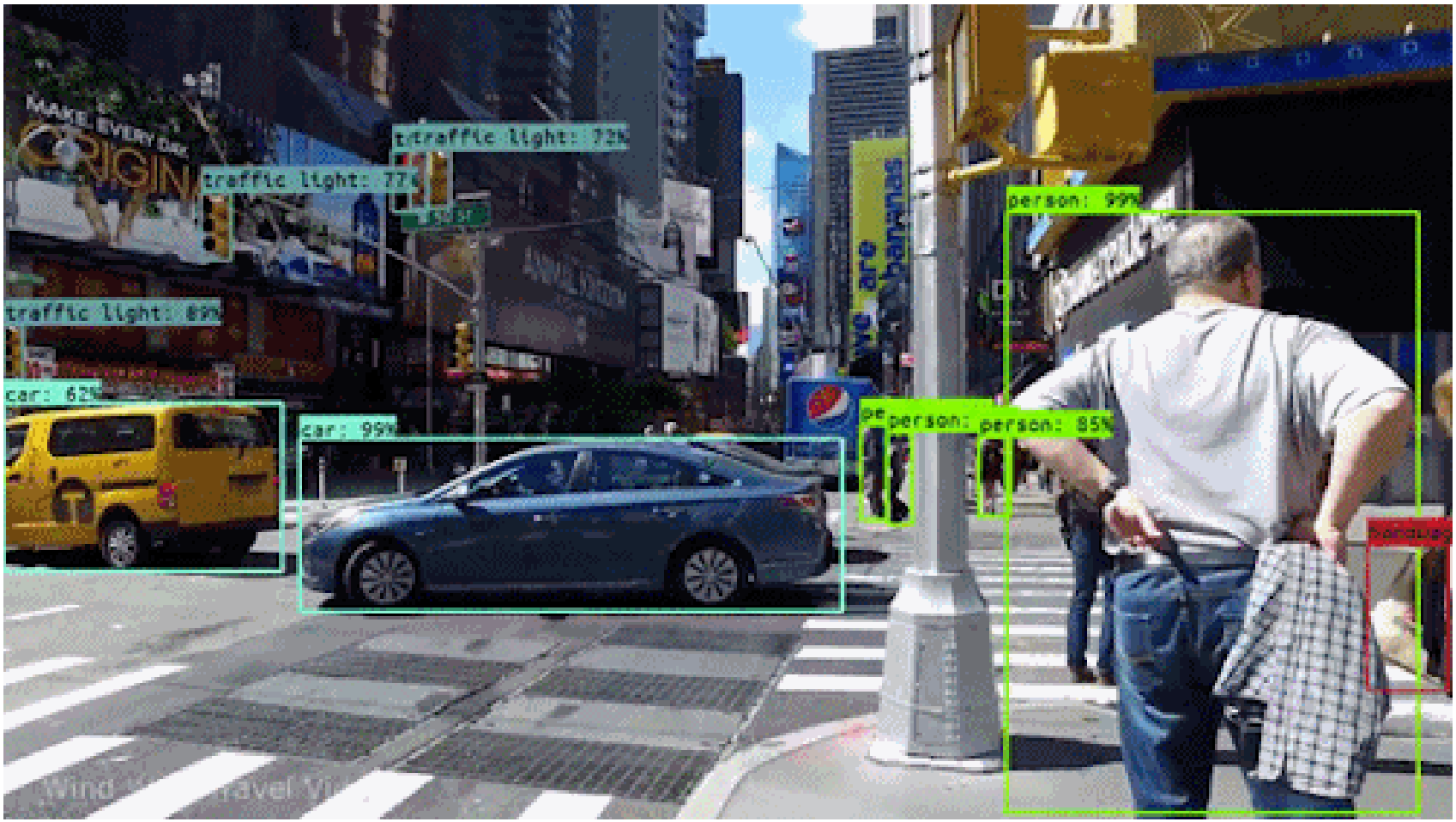
Self-driving car



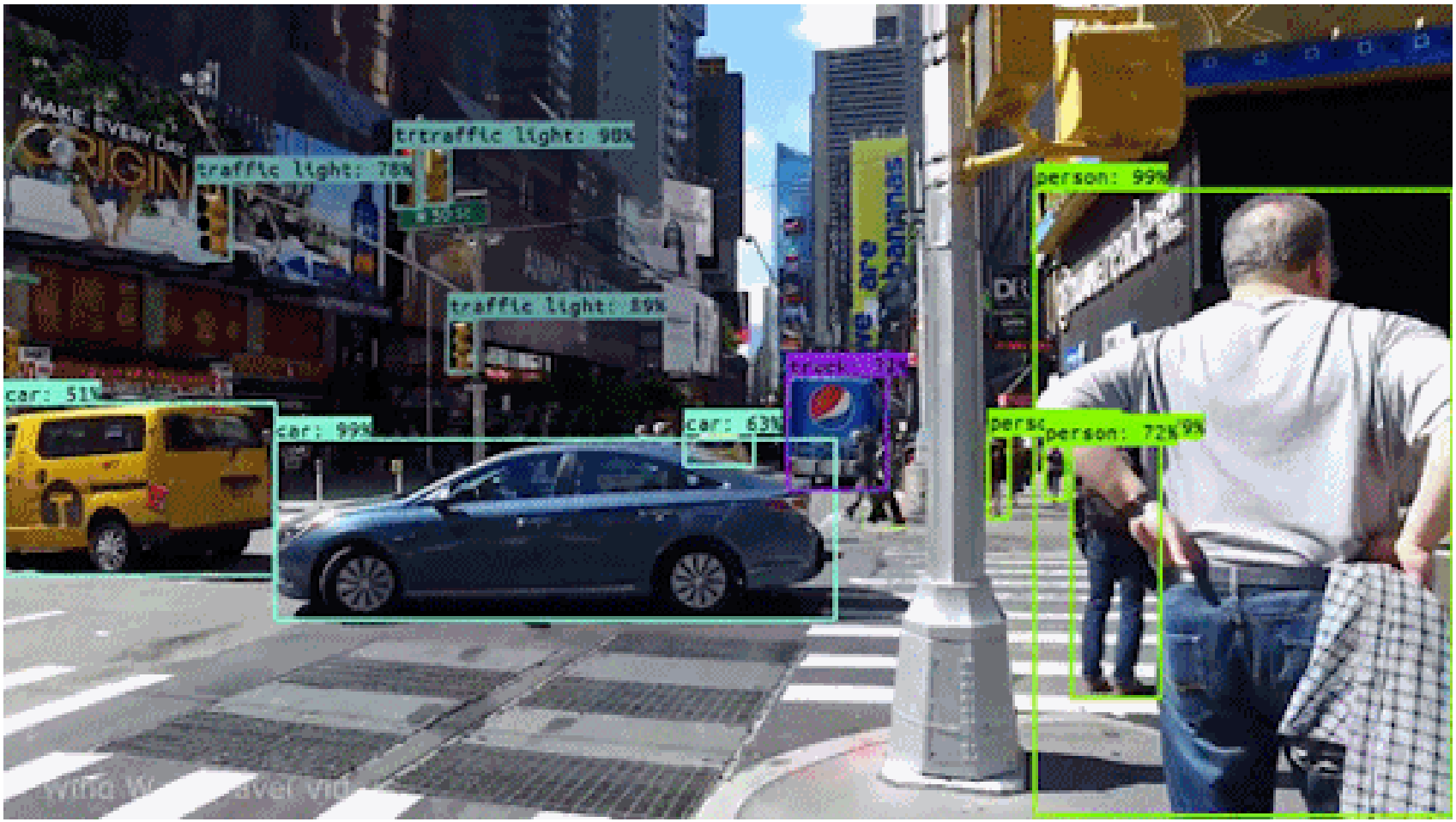
Self-driving car



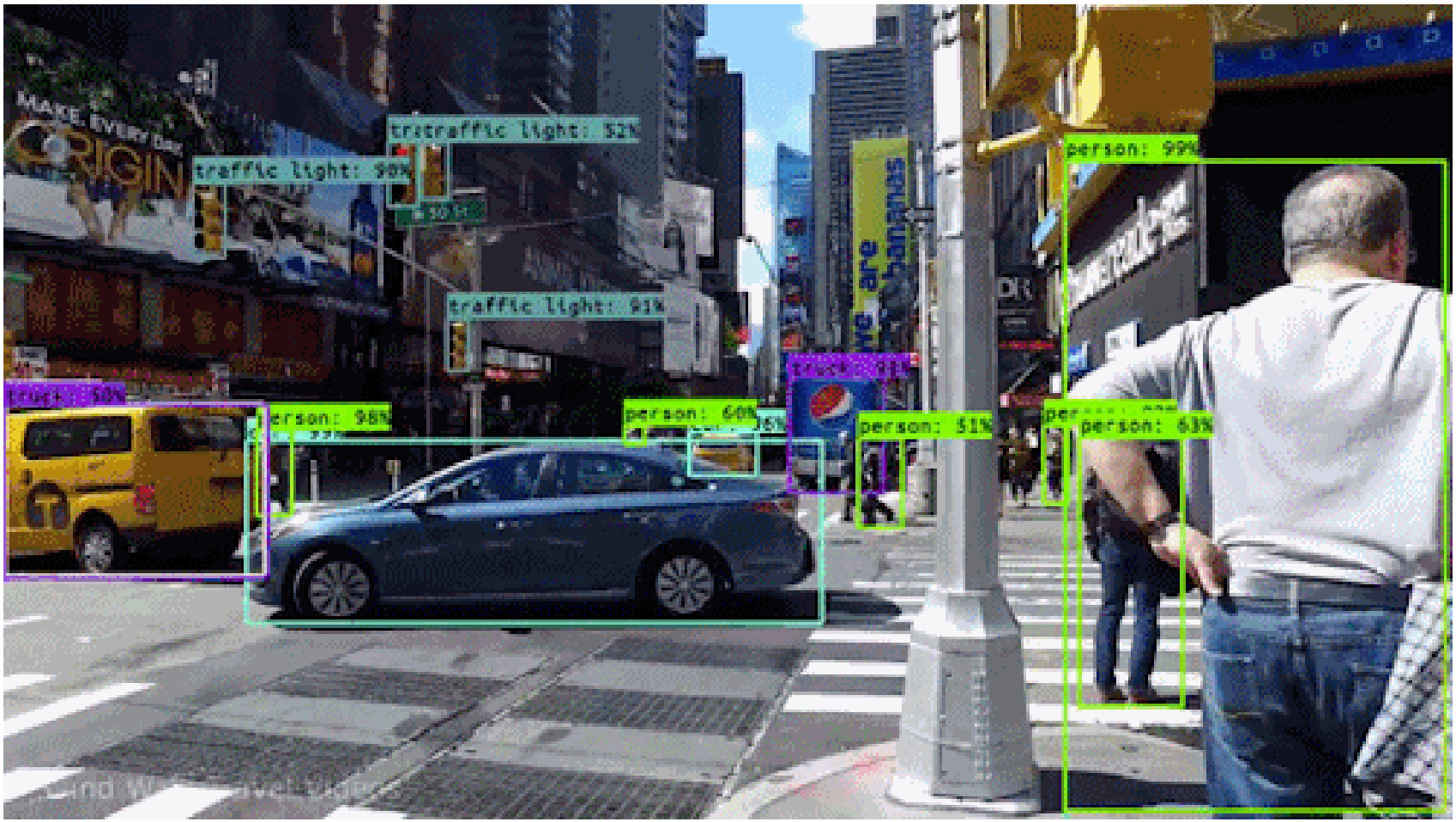
Self-driving car



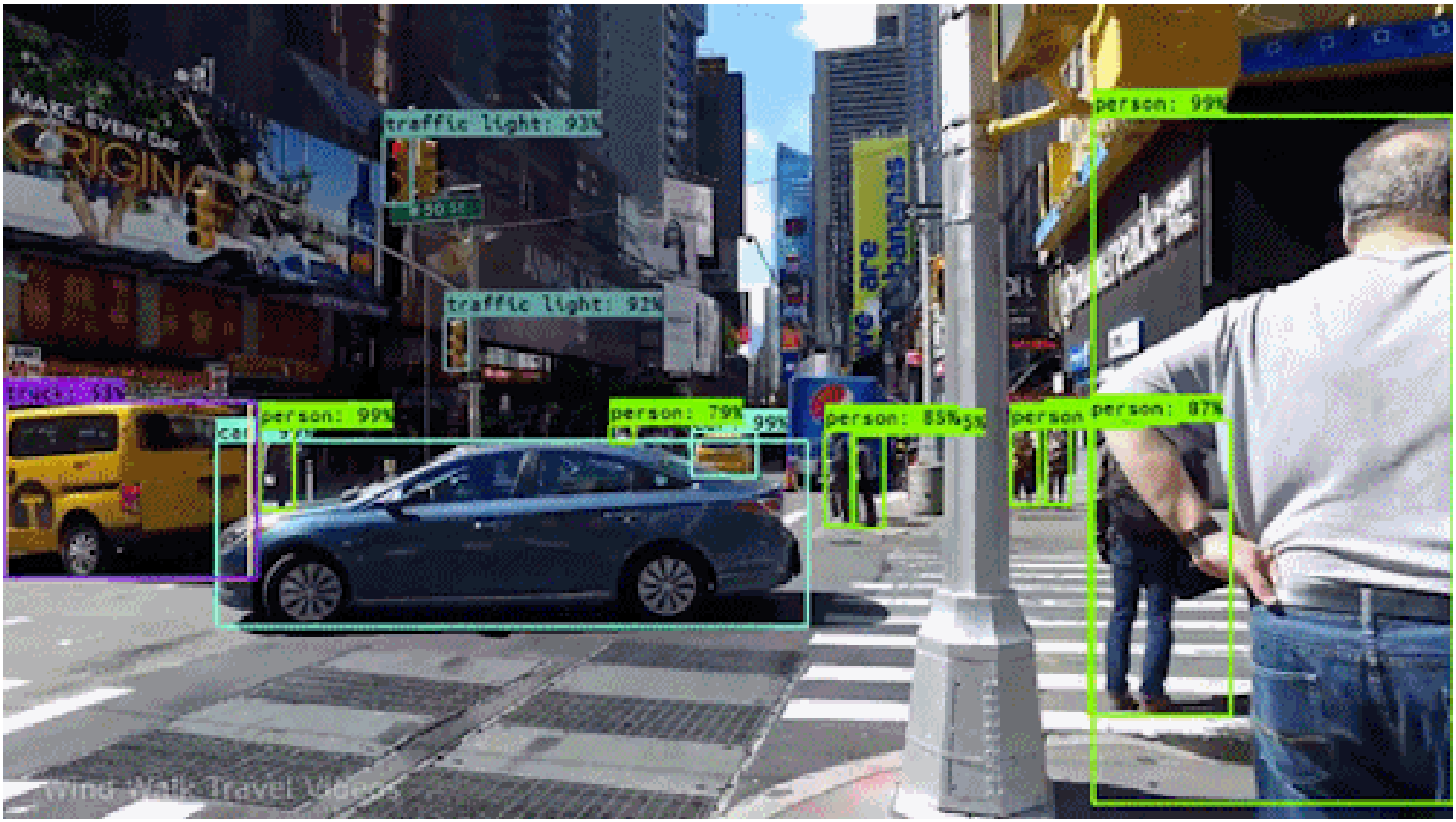
Self-driving car



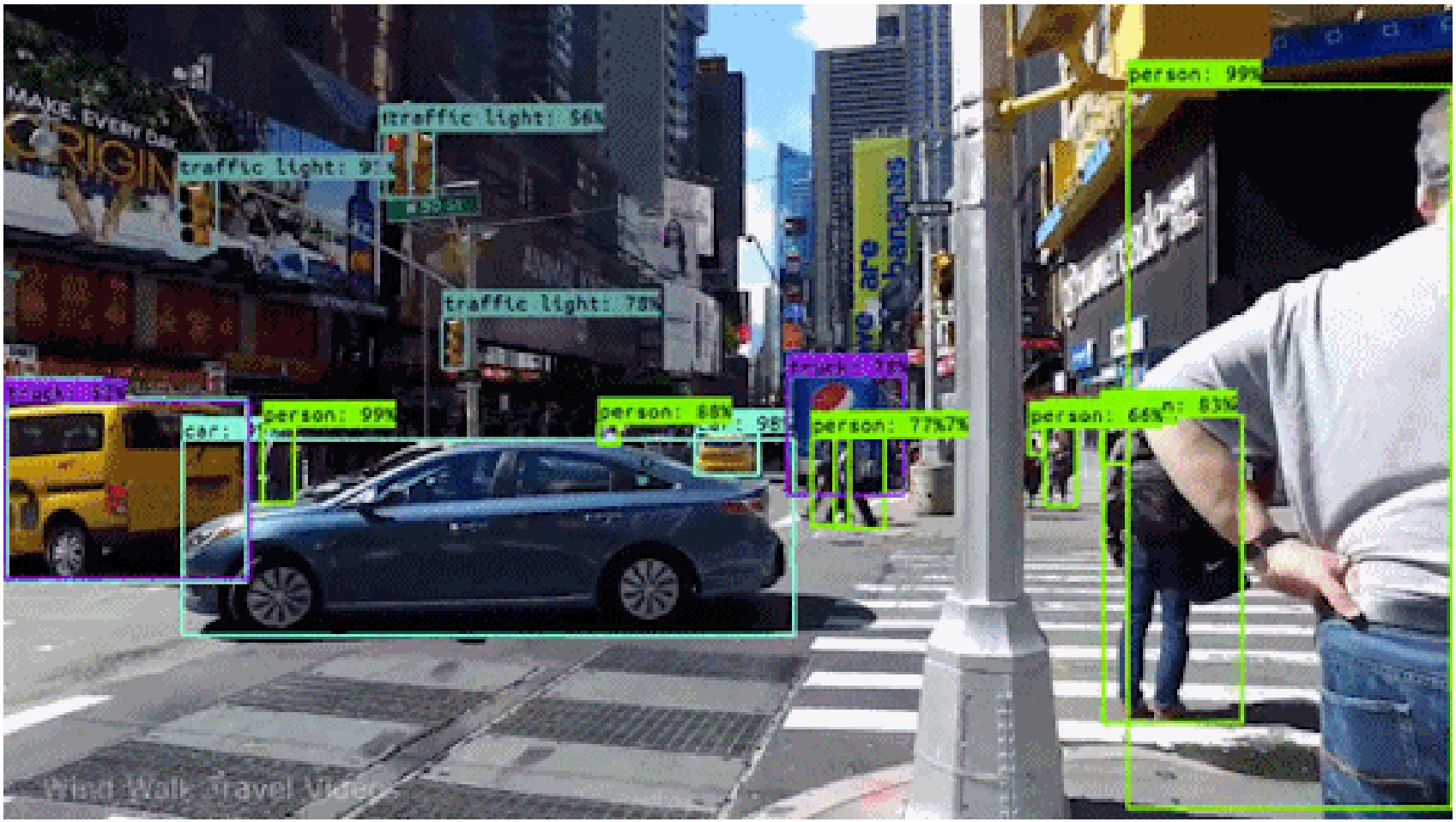
Self-driving car



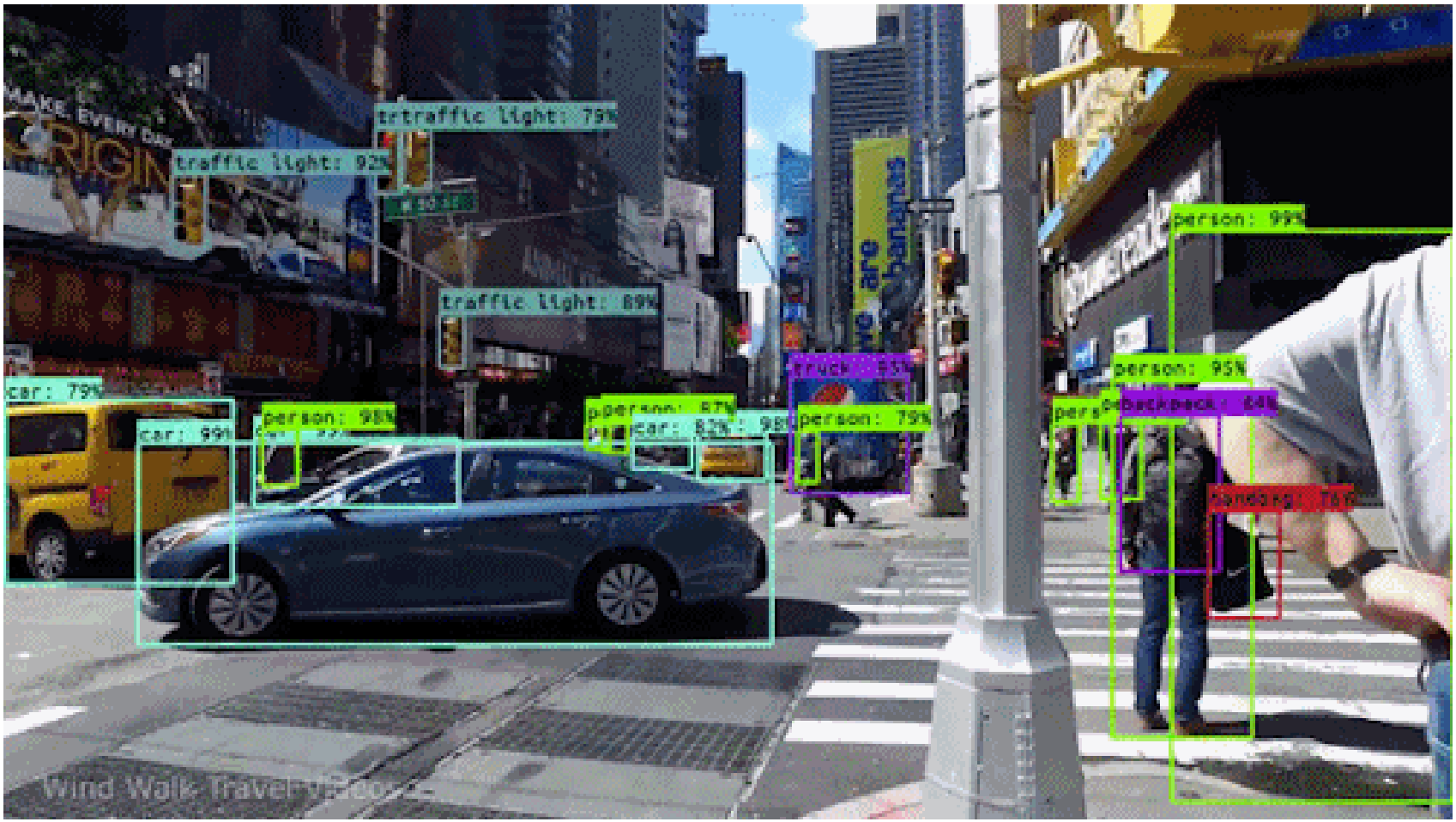
Self-driving car



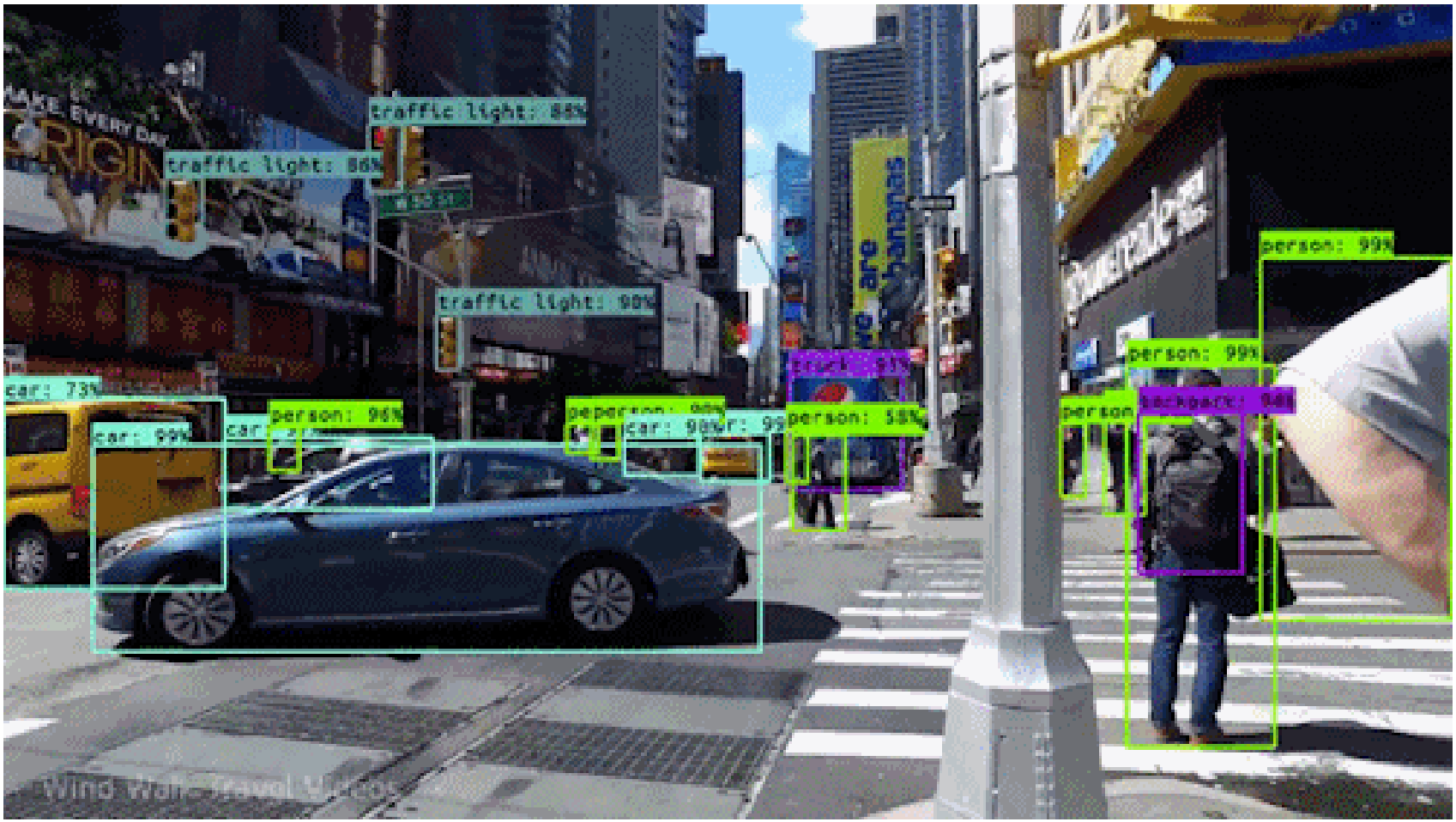
Self-driving car



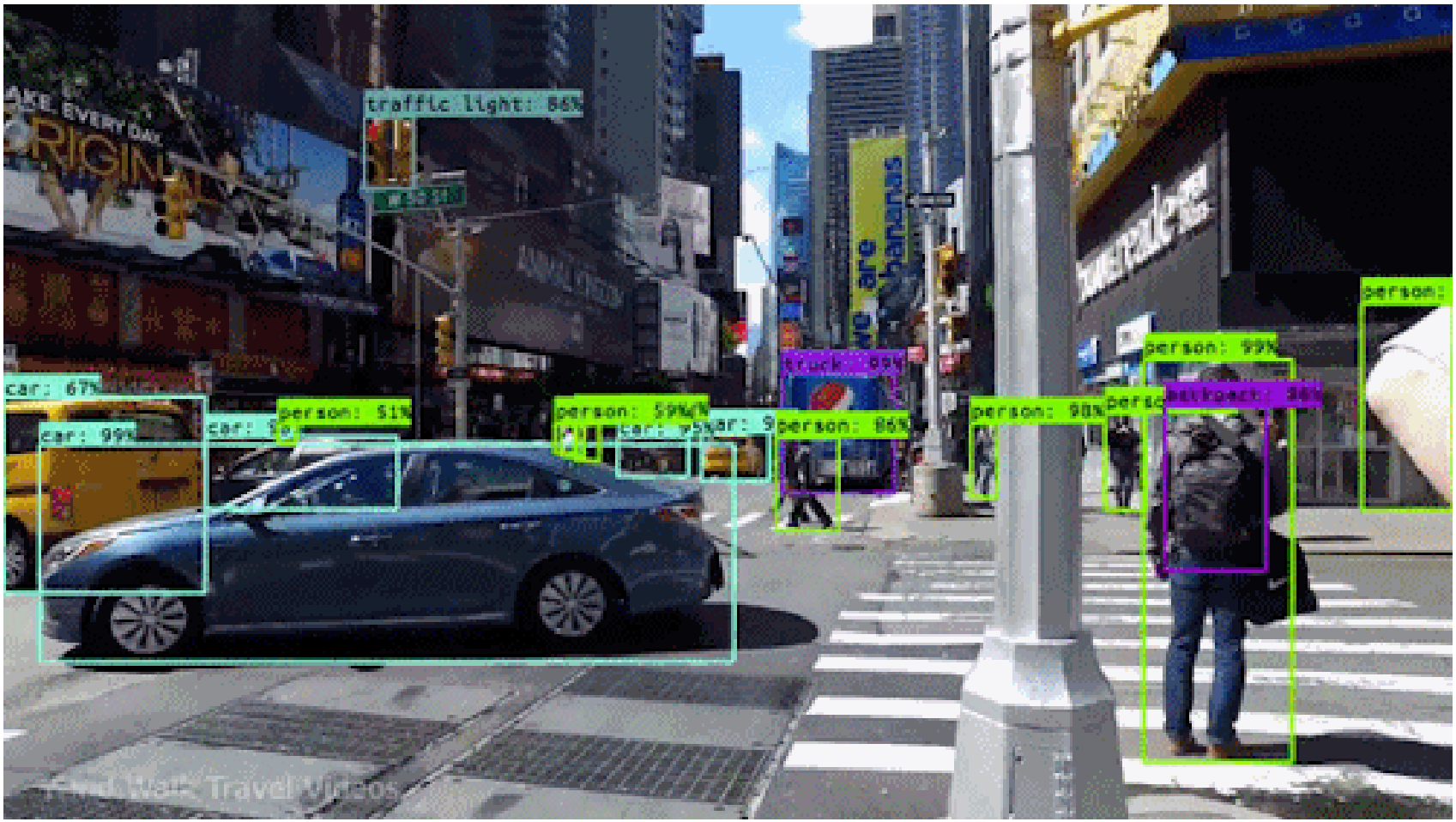
Self-driving car



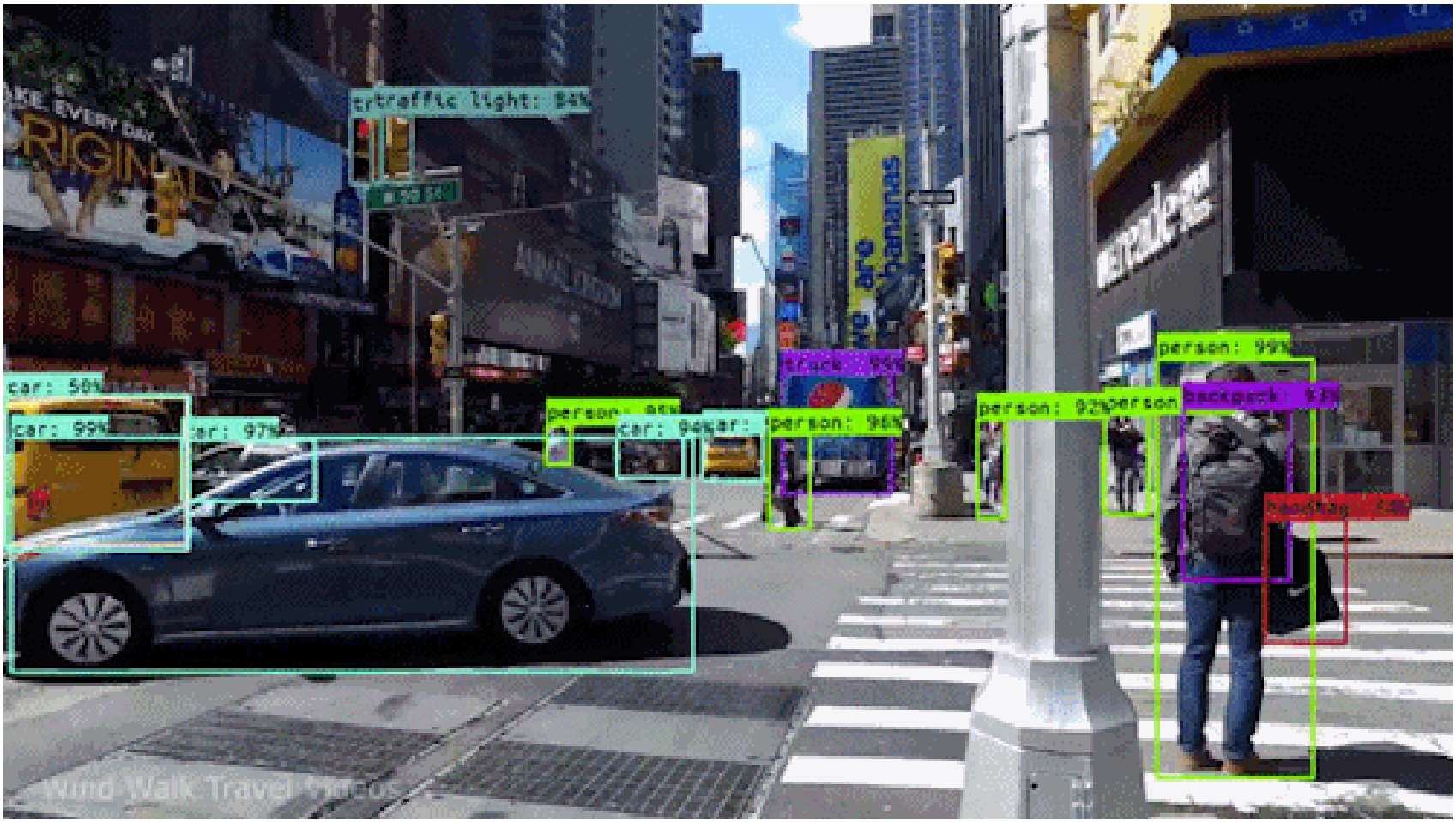
Self-driving car



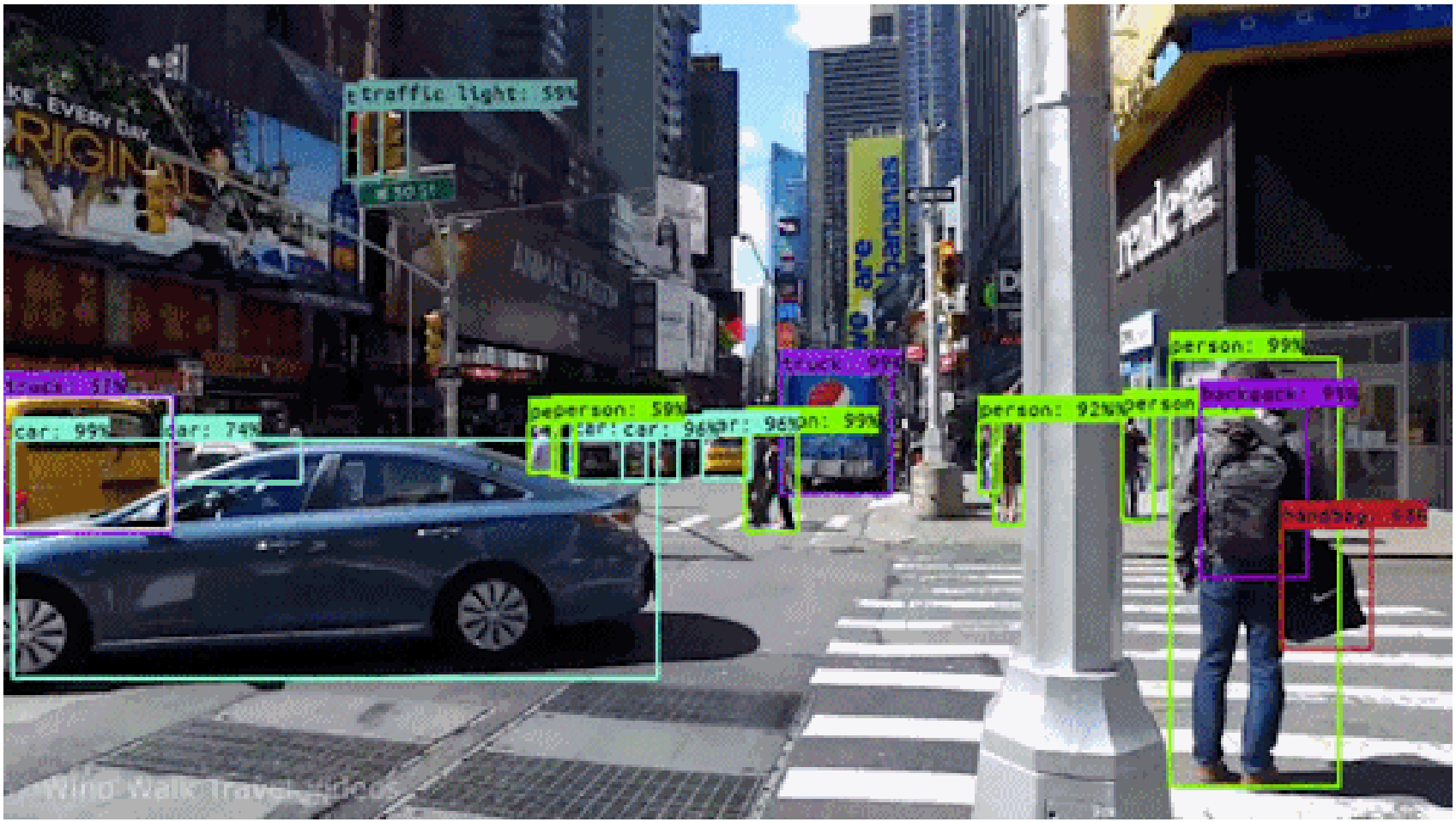
Self-driving car



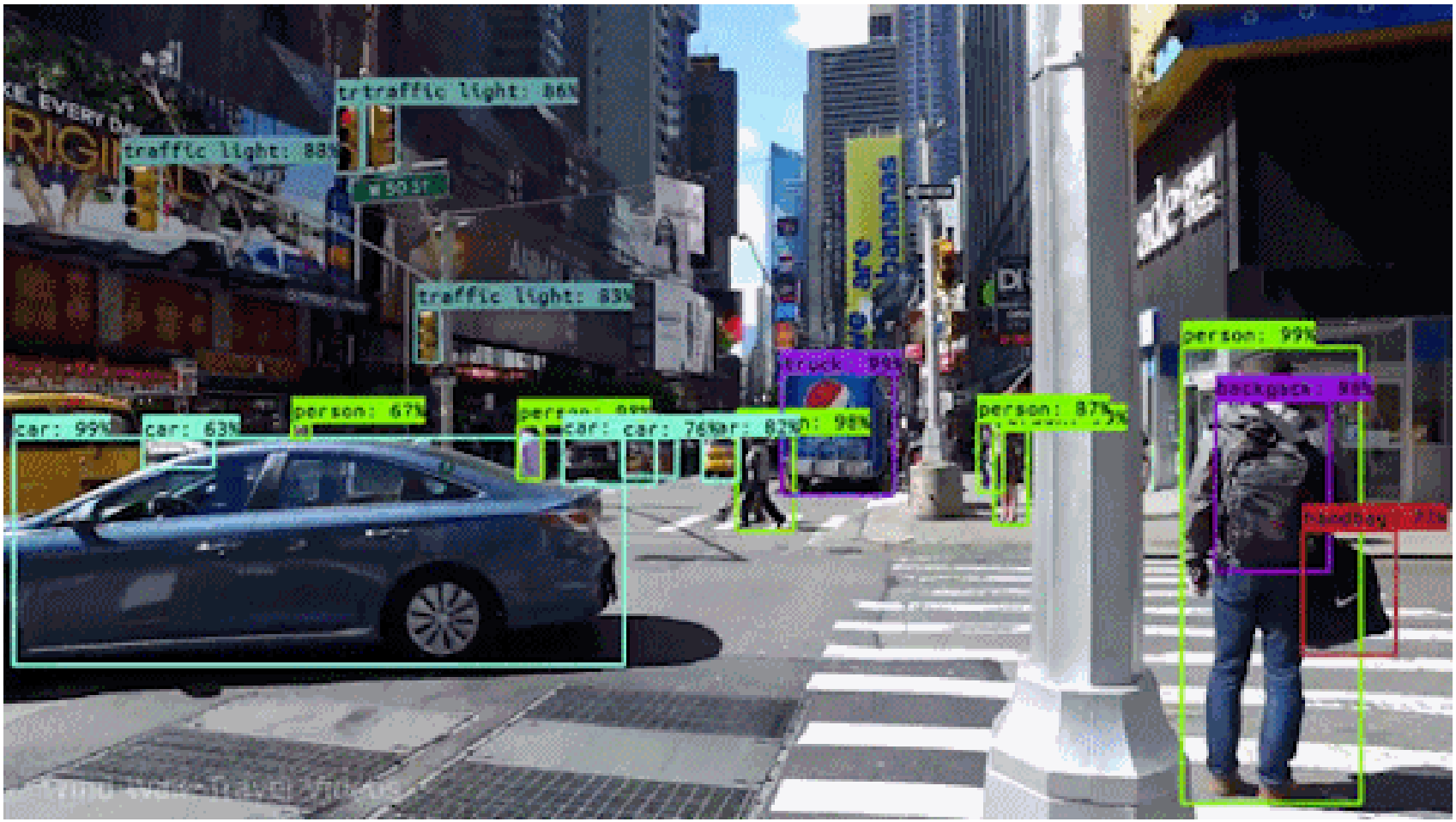
Self-driving car



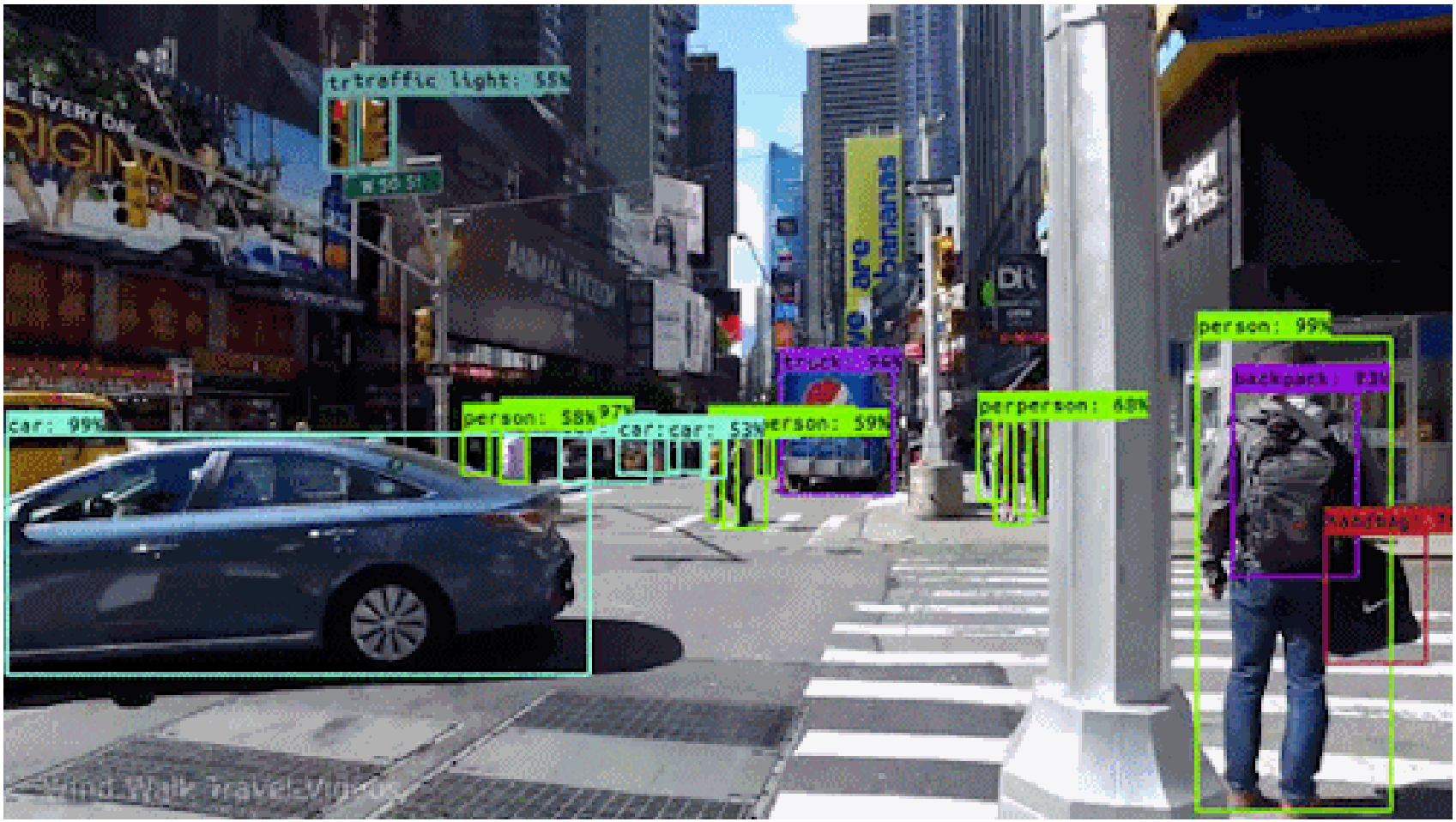
Self-driving car



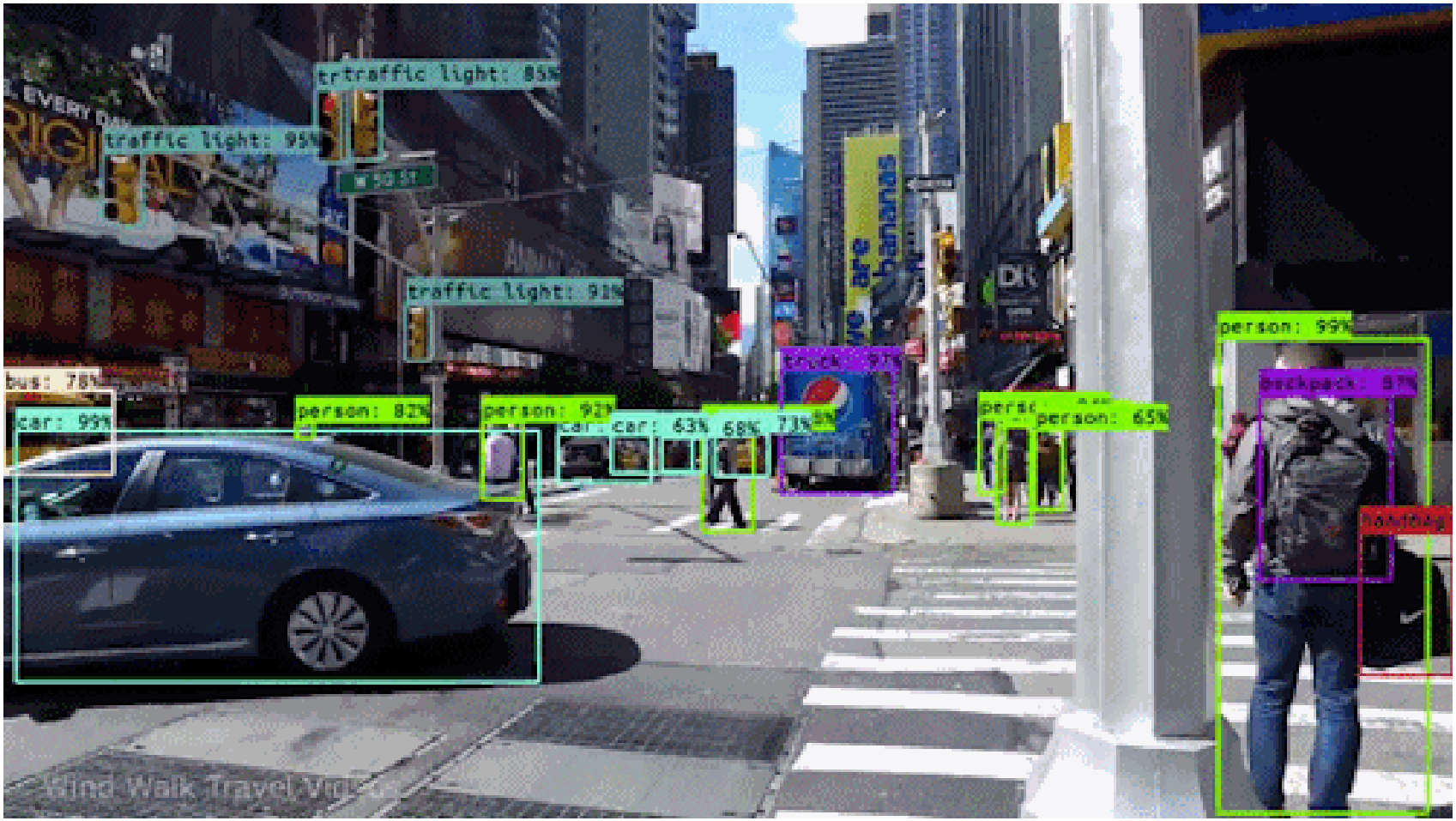
Self-driving car



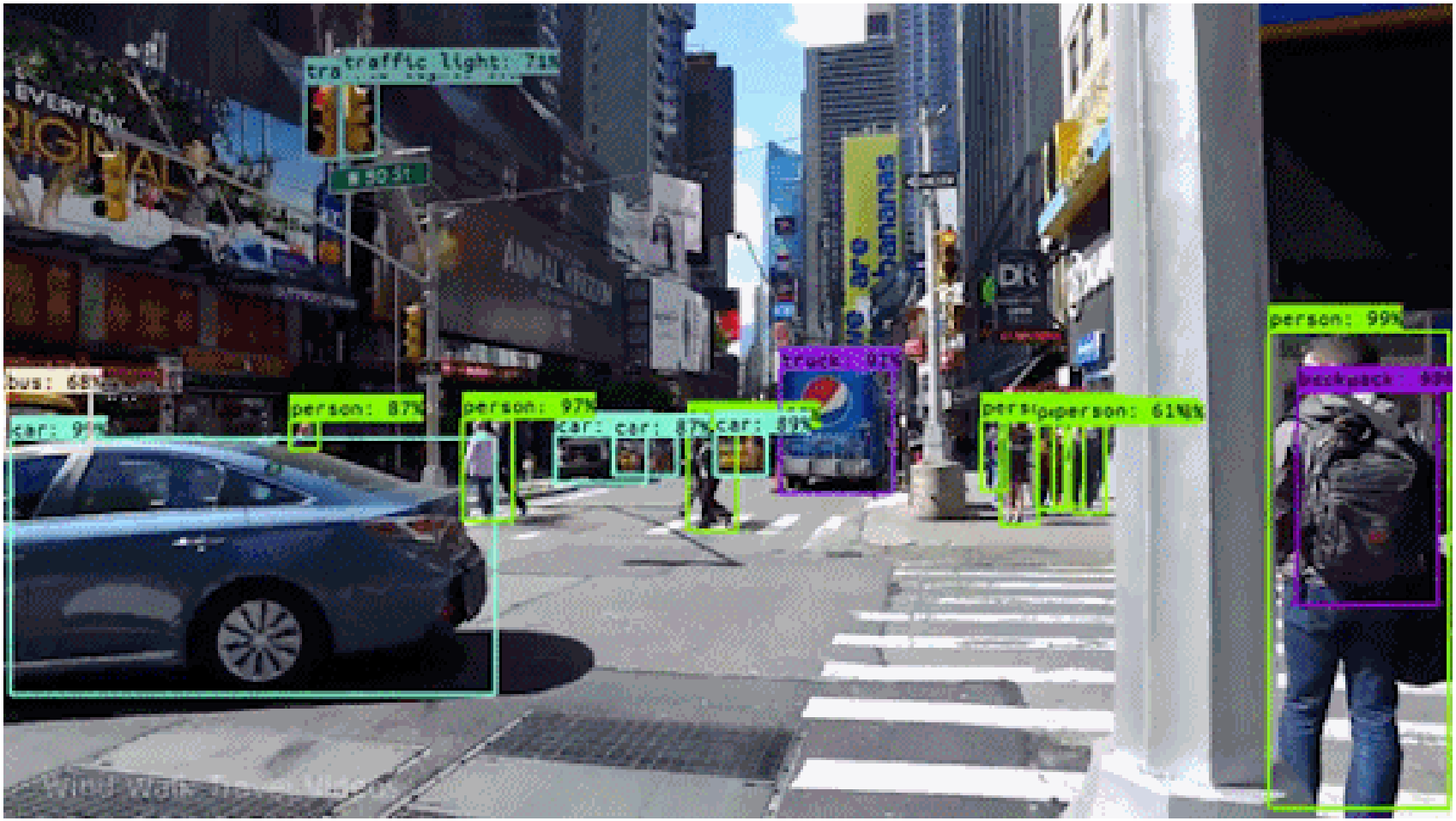
Self-driving car



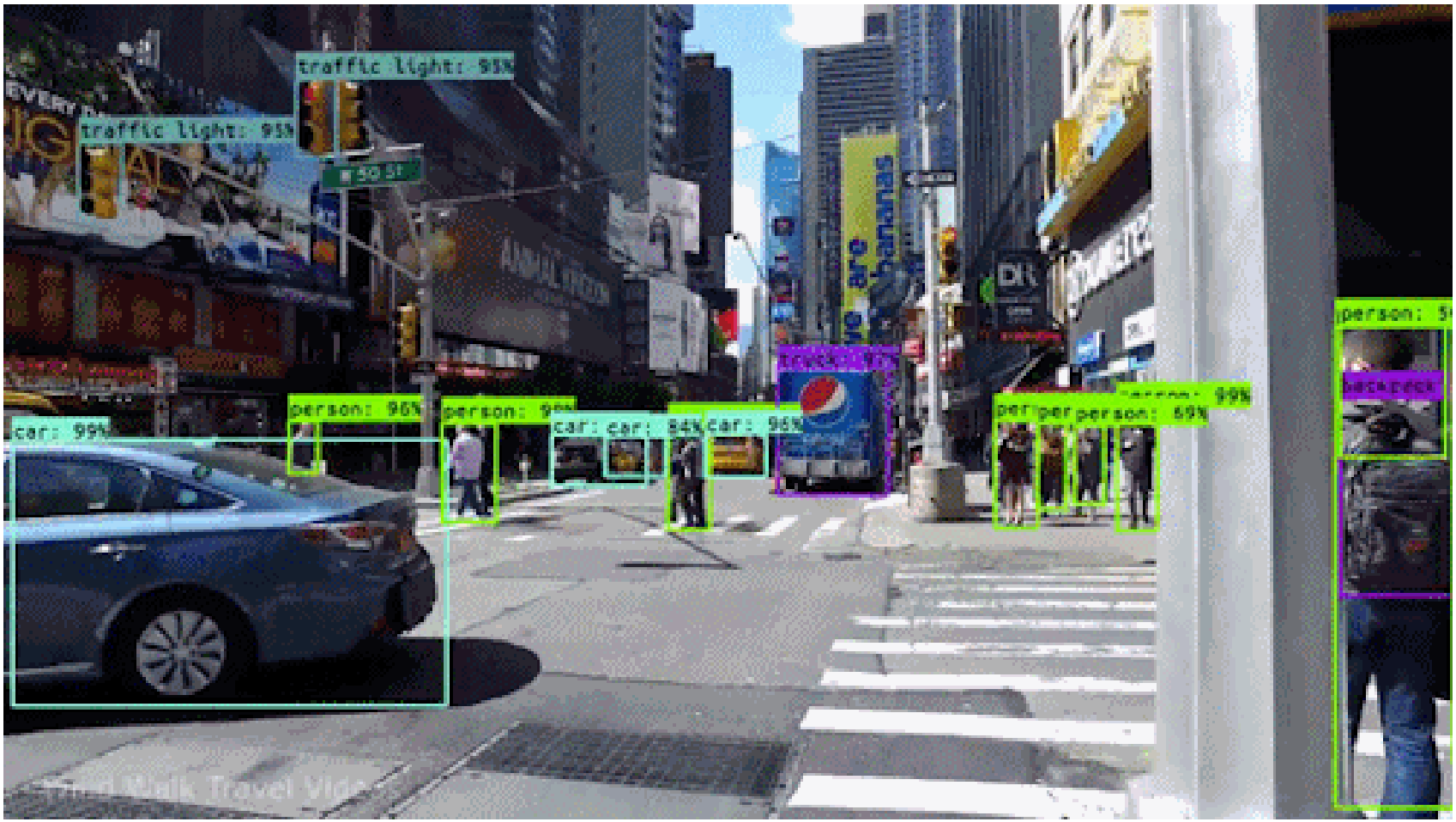
Self-driving car



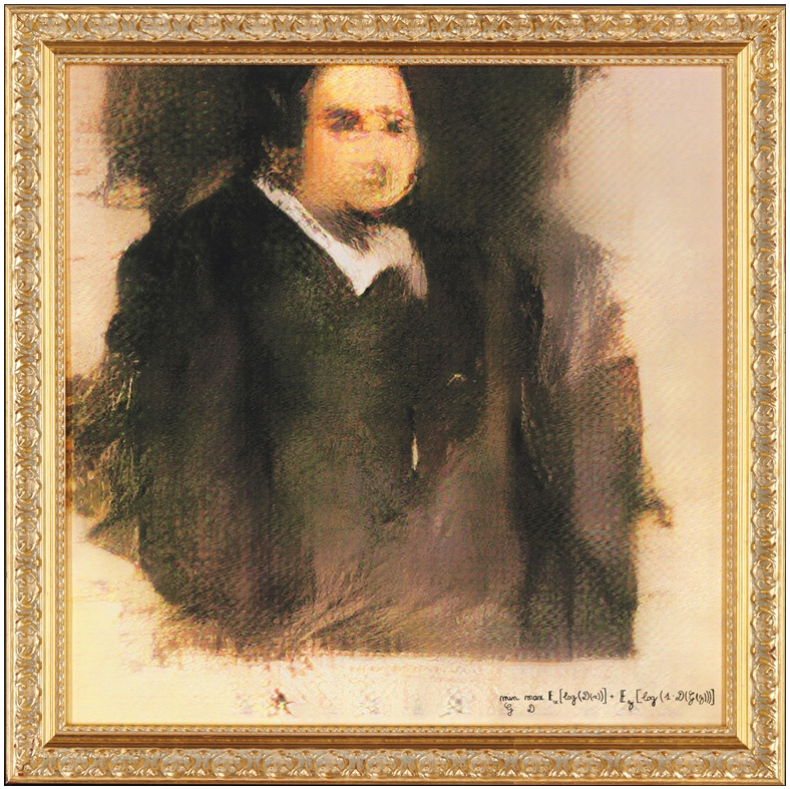
Self-driving car



Self-driving car



Generative model



- Picture generation
- Deep-fakes

Outline

- First session:
  "From the perceptron to the multi-layers perceptron (MLP)"
  - What is a neural network?
  - How to train a neural network?
  - Python code from scratch
- Second session:
  "Custom architectures"
  - Convolutional Neural Networks (CNN)
  - Intoduction to Keras
- Third session:
  "Generative models "
  - Auto-Encoders (AE)
  - Generative Adversarial Networks (GAN)
  - Keras

General framework of the day

- N individuals (for instance N images)
- For each individual i, we know
  - a set of features

$$\mathbf{x}_i = \begin{bmatrix} x_{1,i} \\ \vdots \\ x_{n,i} \end{bmatrix}$$

  - an interest variable $\mathbf{y}_i \in \mathbb{R}^m$
- If $\mathbf{y}_i$ is a categorical variable, $\mathbf{y}_i$ is named label

Aim

Predict the value of $\mathbf{y}$ for a new individual, knowing its features $\mathbf{x}$

General framework of the day

- N individuals (for instance N images)
- For each individual i, we know
  - a set of features

$$\mathbf{x}_i = \begin{bmatrix} x_{1,i} \\ \vdots \\ x_{n,i} \end{bmatrix}$$

  - an interest variable $\mathbf{y}_i \in \mathbb{R}^m$
- If $\mathbf{y}_i$ is a categorical variable, $\mathbf{y}_i$ is named label

Aim

Predict the value of $\mathbf{y}$ for a new individual, knowing its features $\mathbf{x}$

- Regression: the components of $\mathbf{y}$ are continuous
- Classification: $\mathbf{y}$ is categorical

General framework of the day

- N individuals (for instance N images)
- For each individual i, we know
  - a set of features

$$\mathbf{x}_i = \begin{bmatrix} x_{1,i} \\ \vdots \\ x_{n,i} \end{bmatrix}$$

  - an interest variable $\mathbf{y}_i \in \mathbb{R}^m$
- If $\mathbf{y}_i$ is a categorical variable, $\mathbf{y}_i$ is named label

Aim

Predict the value of $\mathbf{y}$ for a new individual, knowing its features $\mathbf{x}$

- Regression: the components of $\mathbf{y}$ are continuous
- Classification: $\mathbf{y}$ is categorical
  - Binary classification: $m = 1$ and $\mathbf{y} \in \{0, 1\}$
  - Multiclass classification: $\mathbf{y} \in \{1, \ldots, K\}$

Matricial representation

- Matrix of the features for the N individuals

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,N} \end{bmatrix}$$

Matricial representation

- Matrix of the features for the N individuals

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,N} \end{bmatrix}$$

- Matrix of labels

$$Y = \begin{bmatrix} y_{1,1} & \cdots & y_{1,N} \\ \vdots & \ddots & \vdots \\ y_{m,1} & \cdots & y_{m,N} \end{bmatrix}$$

Matricial representation

- Matrix of the features for the N individuals

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,N} \end{bmatrix}$$

- Matrix of labels

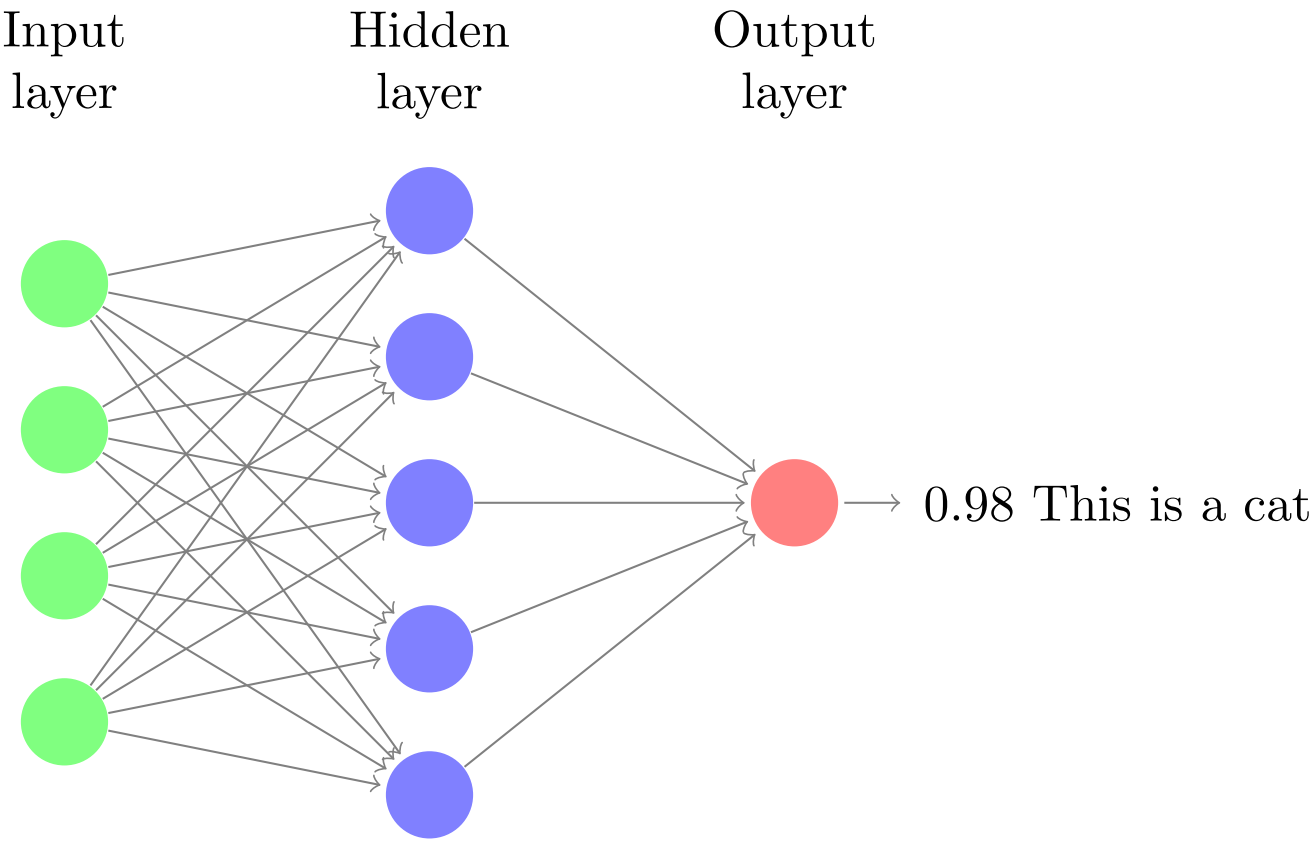
$$Y = \begin{bmatrix} y_{1,1} & \cdots & y_{1,N} \\ \vdots & \ddots & \vdots \\ y_{m,1} & \cdots & y_{m,N} \end{bmatrix}$$

- When $\mathbf{y}$ is categorical with K categories (with $K > 2$), we work with the indicators of each category (one-hot encoding)

$$y_{k,i} = 1_{y_i = k}$$

and $m = K$

Example: binary classification

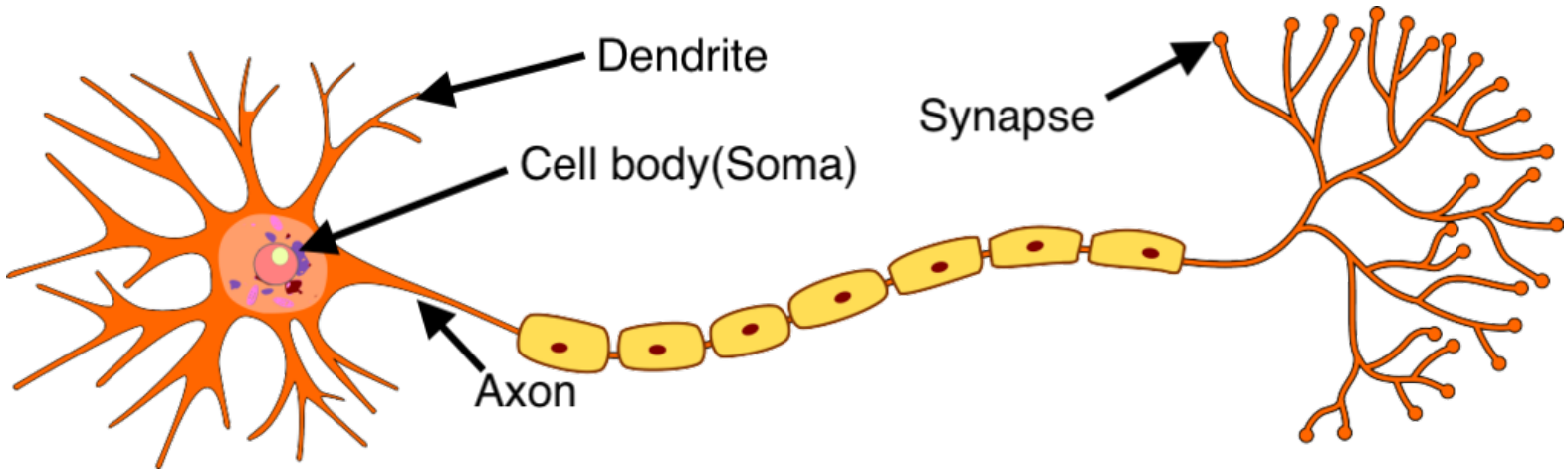


Find $\mathbf{W}$ such as all the $f_{\mathbf{W}}(\mathbf{x}_i)$'s of a training set are "close" to the associated labels $\mathbf{y}_i$'s where

- $\mathbf{x}_i$ is the vector of features of the i^{th} image (pixel values)
- $\mathbf{y}_i$ is the label (cat or not cat)

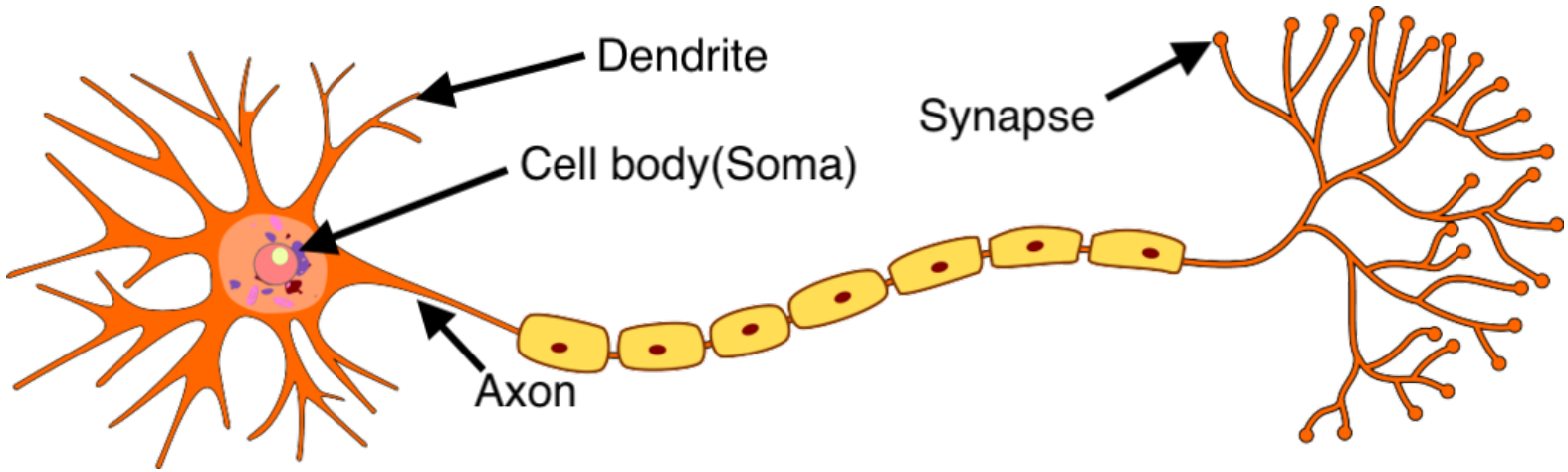
Artificial neural

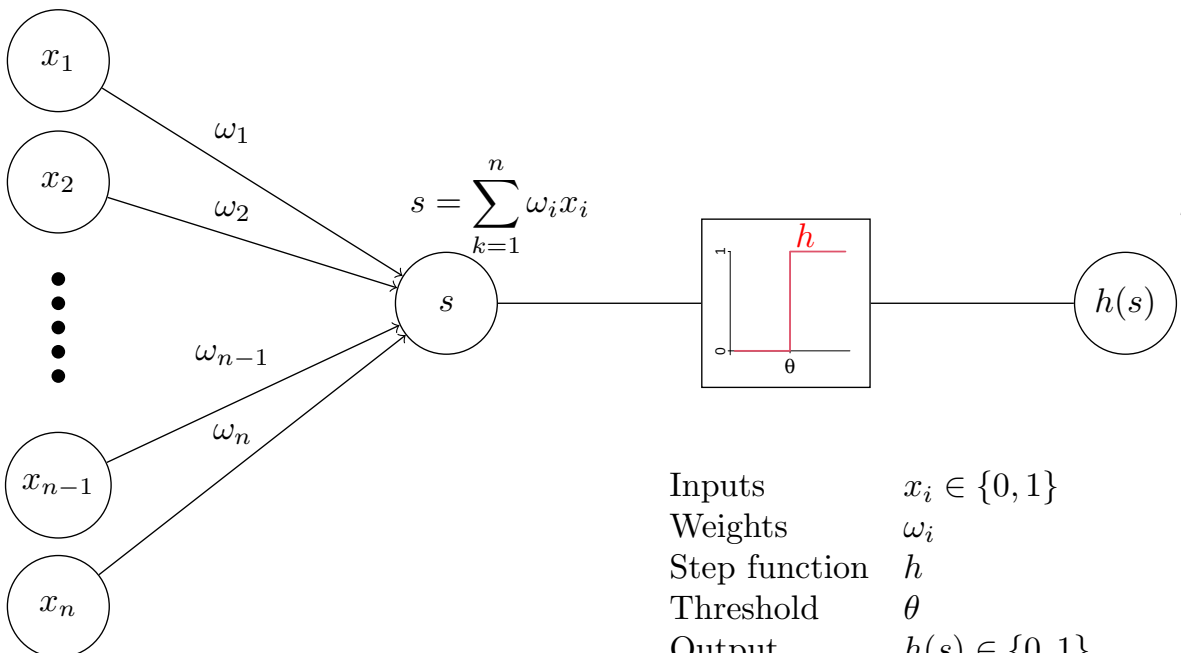
A first model: Mc Culloch and Pitts (1943)



Artificial neural

A first model: Mc Culloch and Pitts (1943)

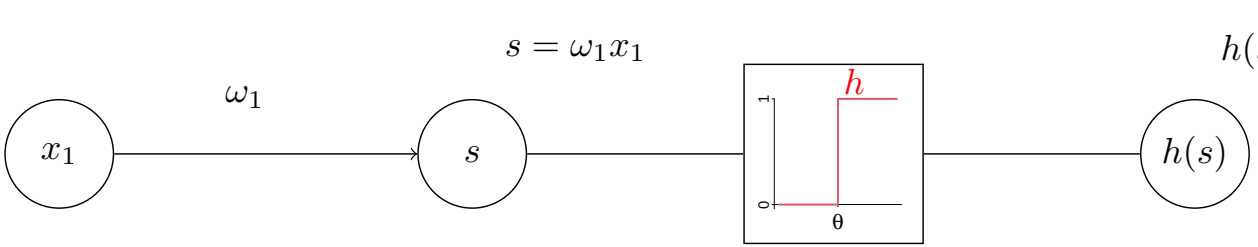




Inputs	$x_i \in \{0,1\}$
Weights	ω_i
Step function	h
Threshold	θ
Output	$h(s) \in \{0,1\}$

Implementing logical gates

NOT



$$h(s) = \begin{cases} 1 & \text{if} \quad s \geq \theta \\ 0 & \text{if} \quad s < \theta \end{cases}$$

x_1	$h(s)$
0	1
1	0

Implementing logical gates

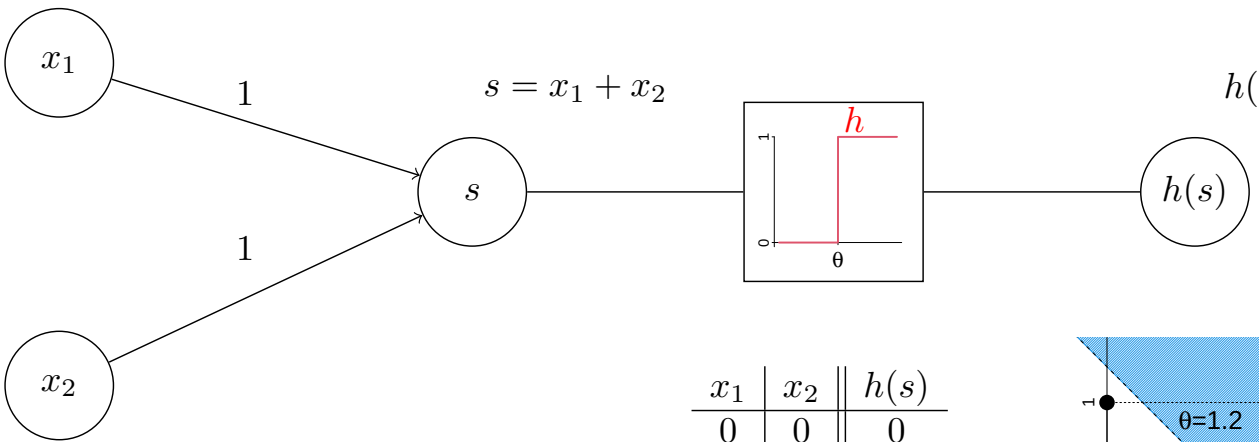
NOT

$$h(s) = \begin{cases} 1 & \text{if} \quad s \geq -0.5 \\ 0 & \text{if} \quad s < -0.5 \end{cases}$$

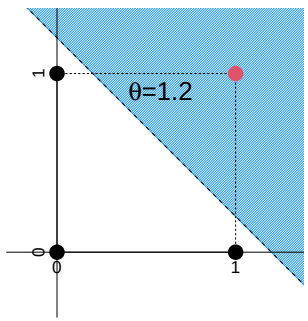
x_1	$h(s)$
0	1
1	0

Implementing logical gates

AND

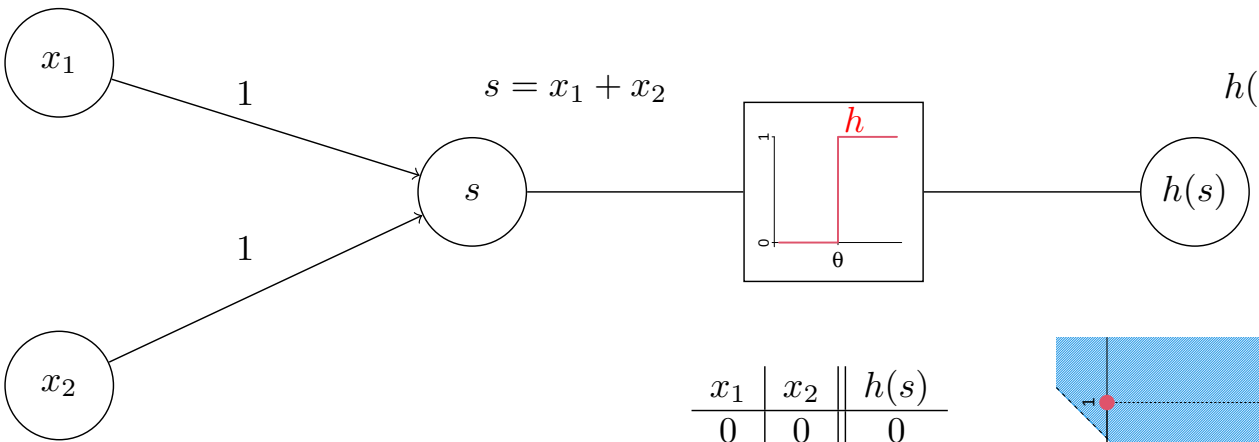


x_1	x_2	$h(s)$
0	0	0
0	1	0
1	0	0
1	1	1

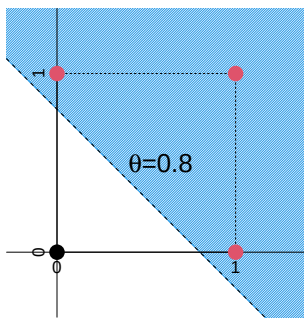


Implementing logical gates

OR

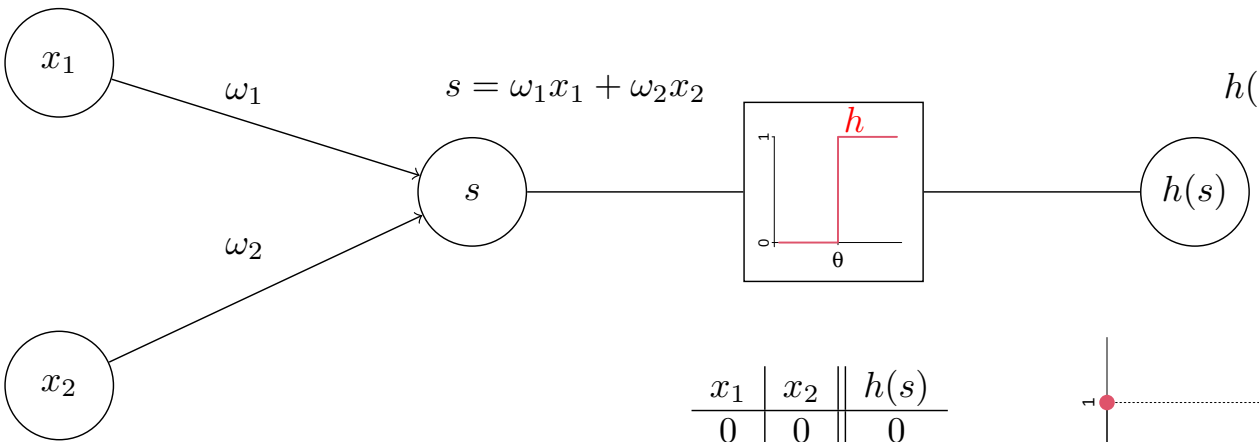


x_1	x_2	$h(s)$
0	0	0
0	1	1
1	0	1
1	1	1



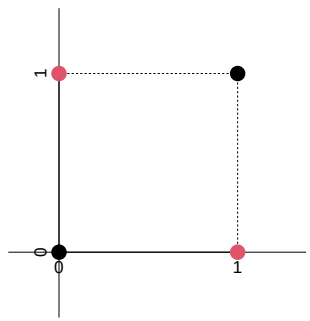
Implementing logical gates

XOR?



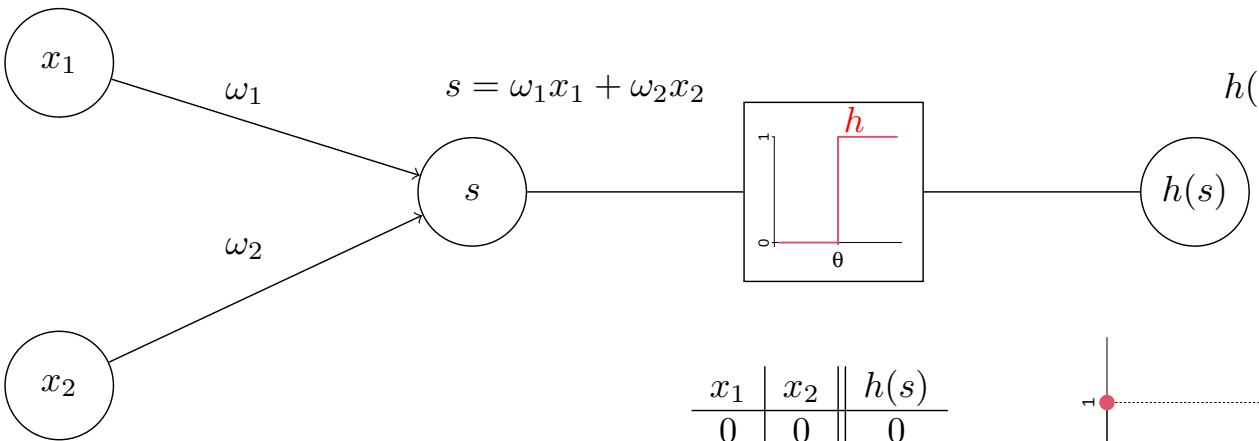
$$h(s) = \begin{cases} 1 & \text{if} \quad s \geq \theta \\ 0 & \text{if} \quad s < \theta \end{cases}$$

x_1	x_2	$h(s)$
0	0	0
0	1	1
1	0	1
1	1	0

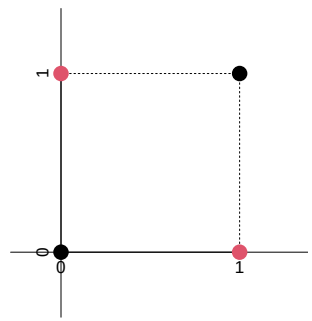


Implementing logical gates

XOR?



x_1	x_2	$h(s)$
0	0	0
0	1	1
1	0	1
1	1	0



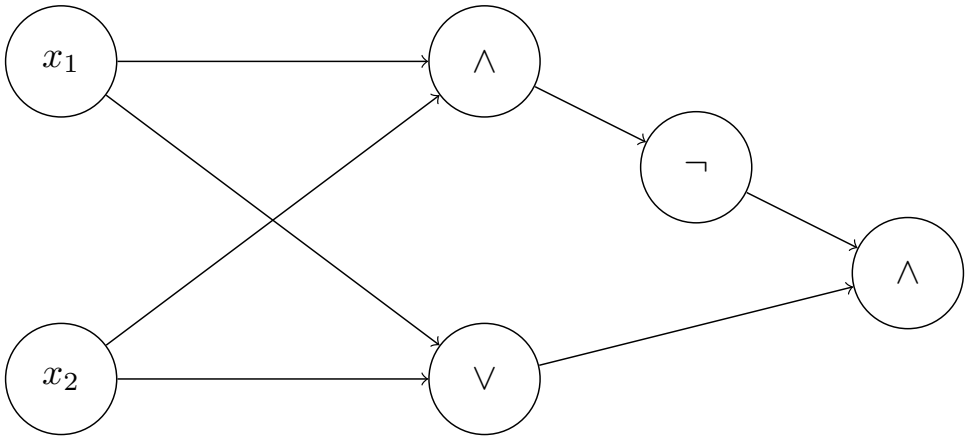
The problem is not linearly separable

Idea: combine neurons

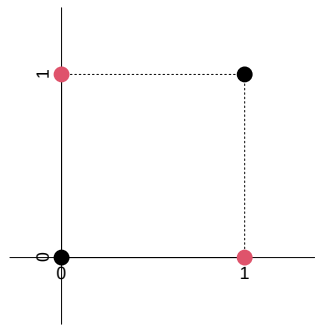
$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$$

Idea: combine neurons

$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$$

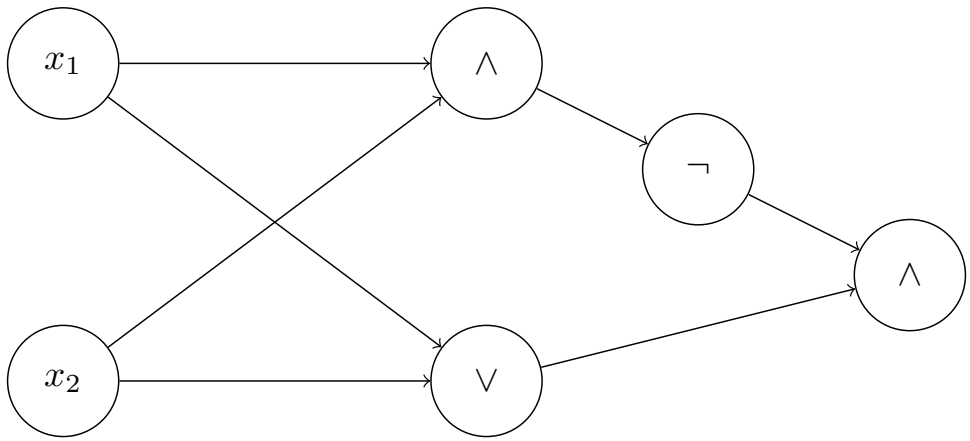


x_1	x_2	$h(s)$
0	0	0
0	1	1
1	0	1
1	1	0

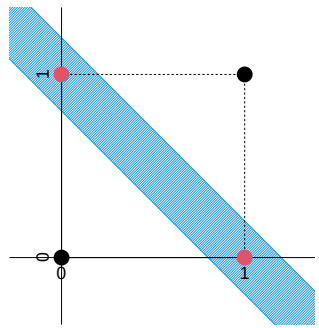


Idea: combine neurons

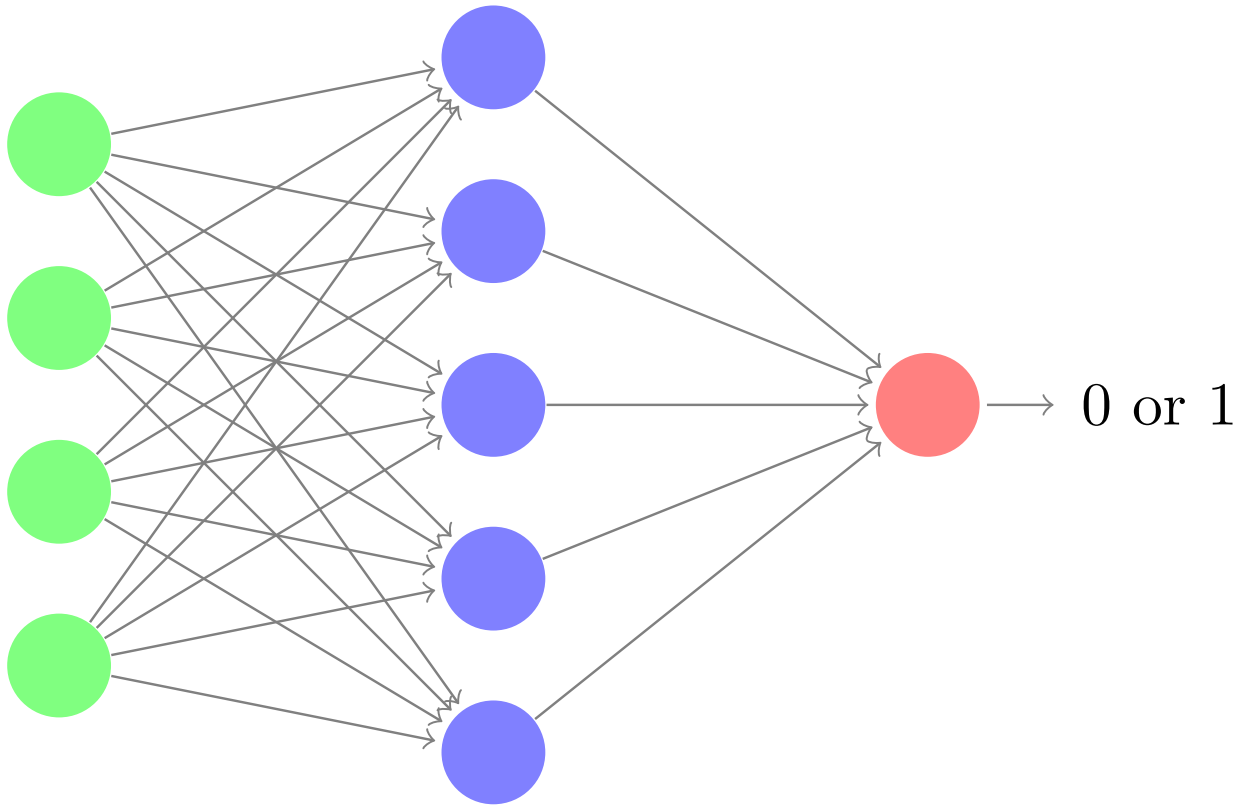
$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$$



x_1	x_2	$h(s)$
0	0	0
0	1	1
1	0	1
1	1	0

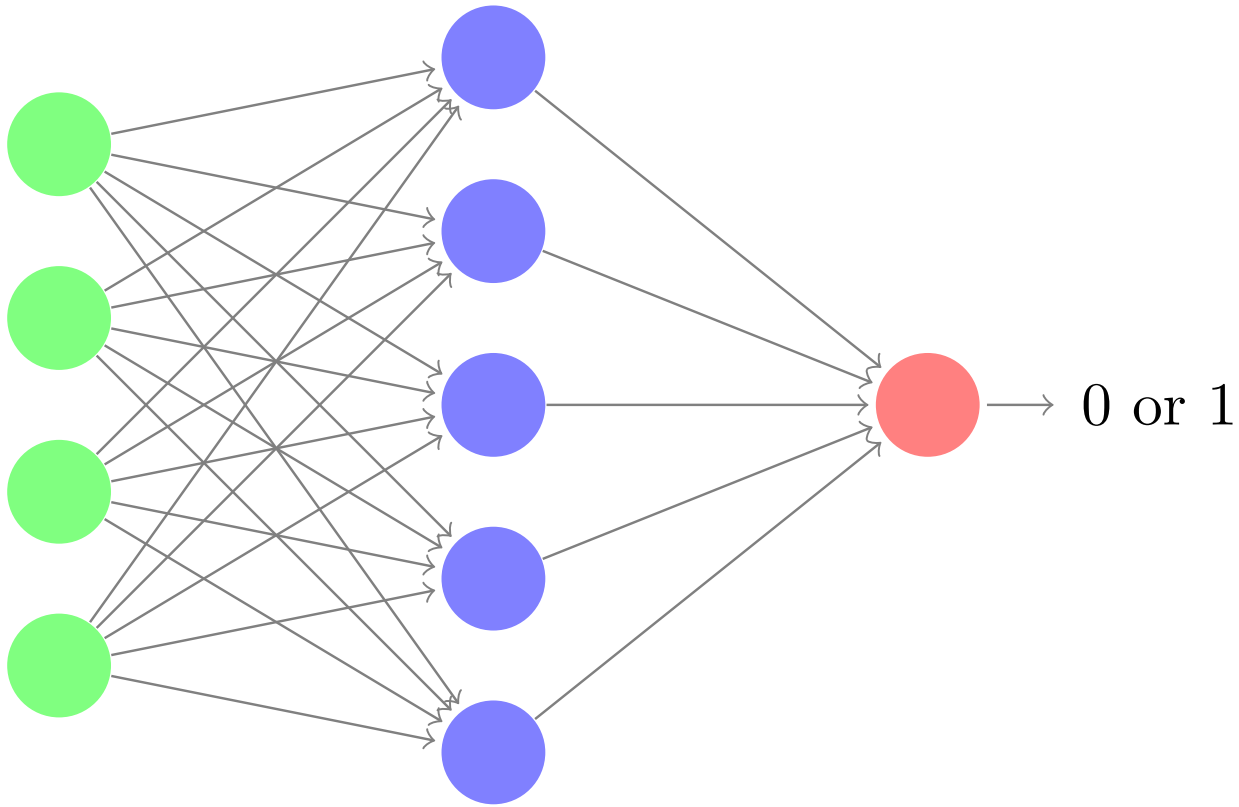


A first theorem on the capacity



- For every integer n, one can build a graph which contains all the functions from $\{0,1\}^n$ to $\{0,1\}$.

A first theorem on the capacity



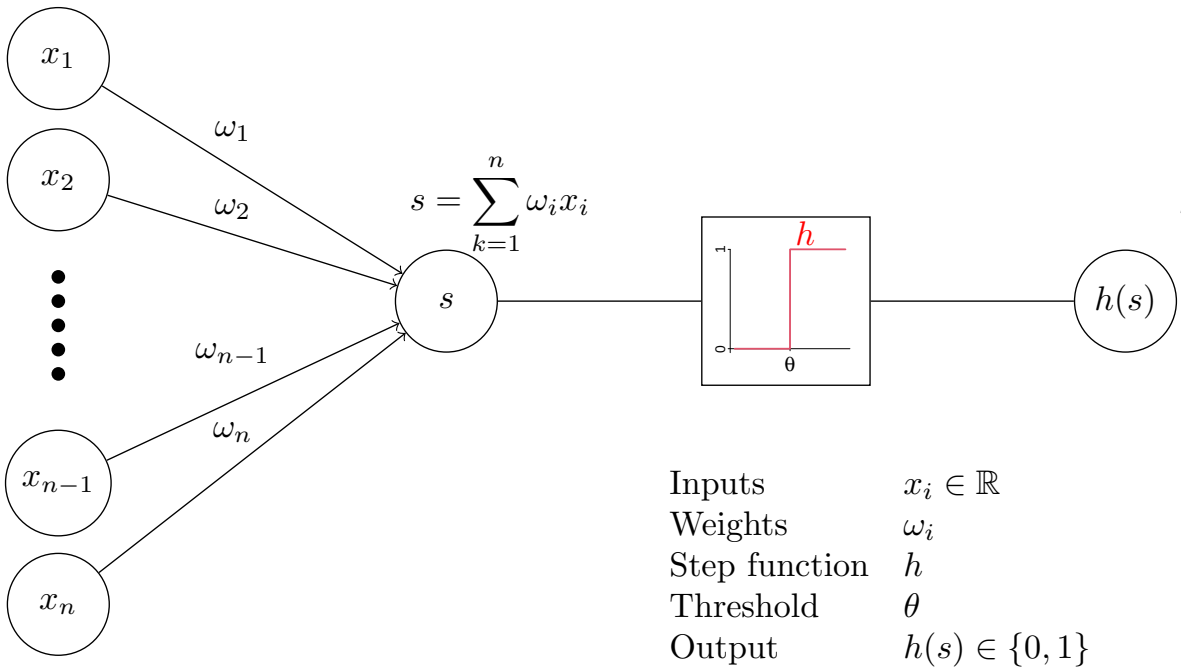
- For every integer n, one can build a graph which contains all the functions from $\{0,1\}^n$ to $\{0,1\}$.
- Unfortunately, the number of necessary neurons in the intermediate layer grows exponentially with n

First steps: the perceptron

Since the advent of electronic computers and modern servo systems, an increasing amount of attention has been focused on the feasibility of constructing a device possessing such human-like functions as perception, recognition, concept formation, and the ability to generalize from experience. In particular, interest has centered on the idea of a machine which would be capable of conceptualizing inputs impinging directly from the physical environment of light, sound, temperature, etc. -- the "phenomenal world" with which we are all familiar -- rather than requiring the intervention of a human agent to digest and code the necessary information.

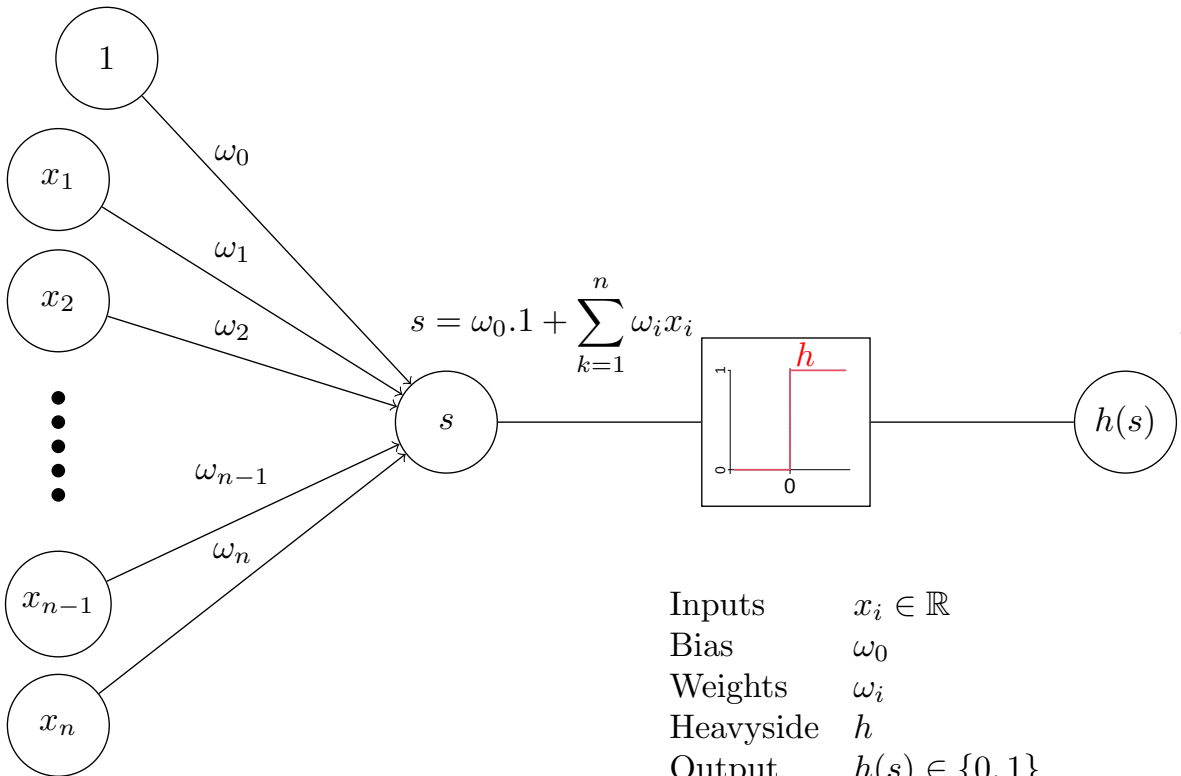
Rosenblatt, F. (1957) The perceptron. A perceiving and recognizing automaton. Cornell aeronautical laboratory, Inc.

The perceptron



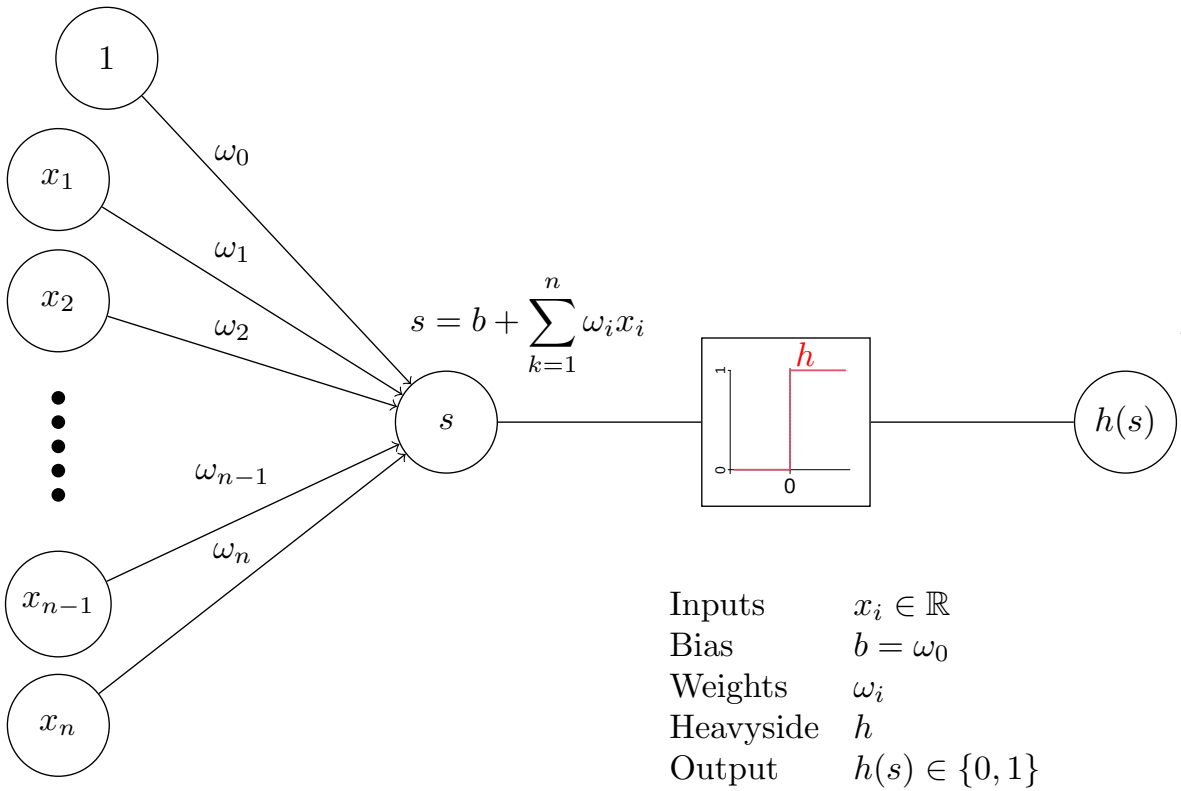
Inputs	$x_i \in \mathbb{R}$
Weights	ω_i
Step function	h
Threshold	θ
Output	$h(s) \in \{0, 1\}$

The perceptron



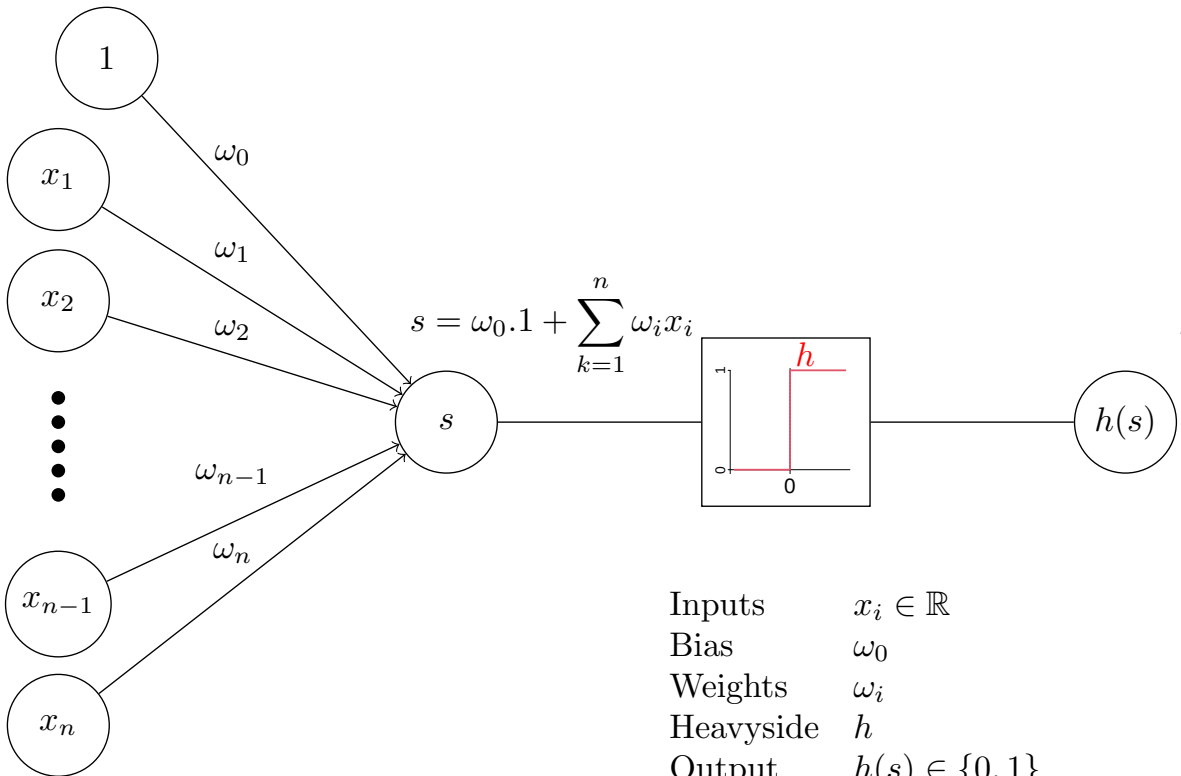
Inputs	$x_i \in \mathbb{R}$
Bias	ω_0
Weights	ω_i
Heavyside	h
Output	$h(s) \in \{0, 1\}$

The perceptron



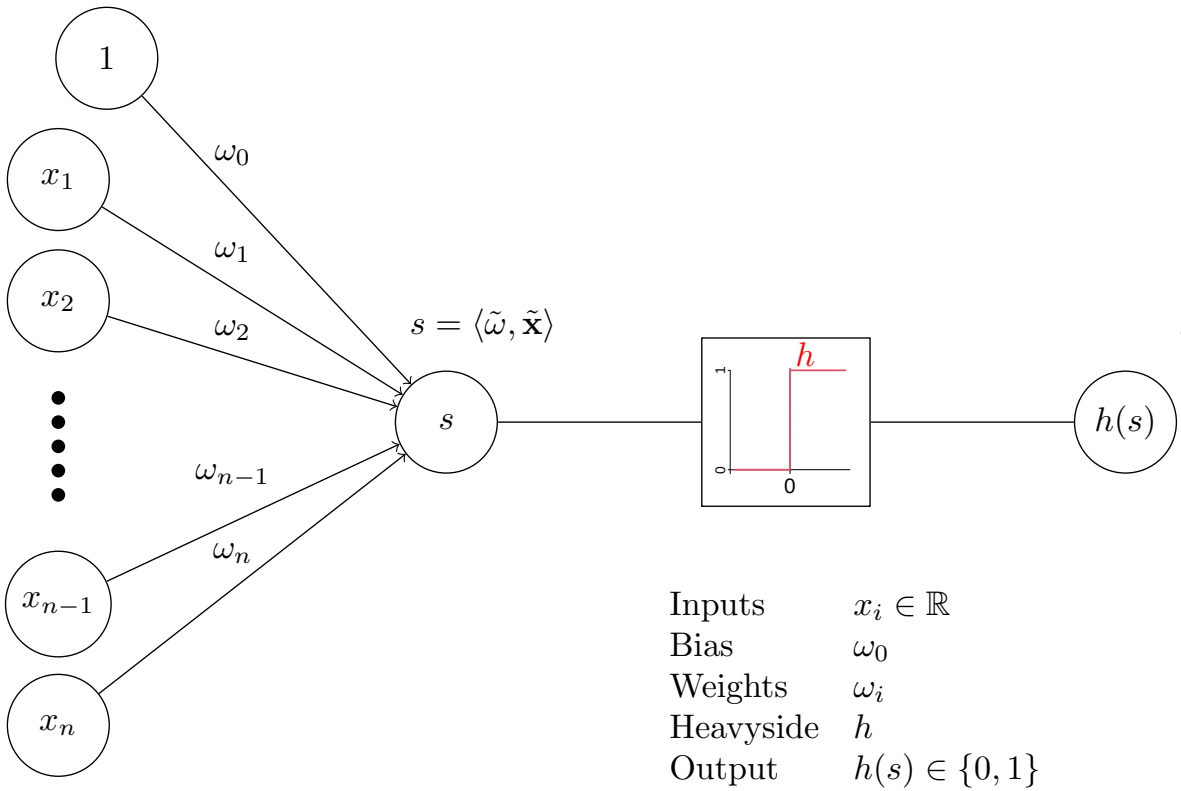
Inputs	$x_i \in \mathbb{R}$
Bias	$b = \omega_0$
Weights	ω_i
Heavyside	h
Output	$h(s) \in \{0, 1\}$

The perceptron



Inputs	$x_i \in \mathbb{R}$
Bias	ω_0
Weights	ω_i
Heavyside	h
Output	$h(s) \in \{0, 1\}$

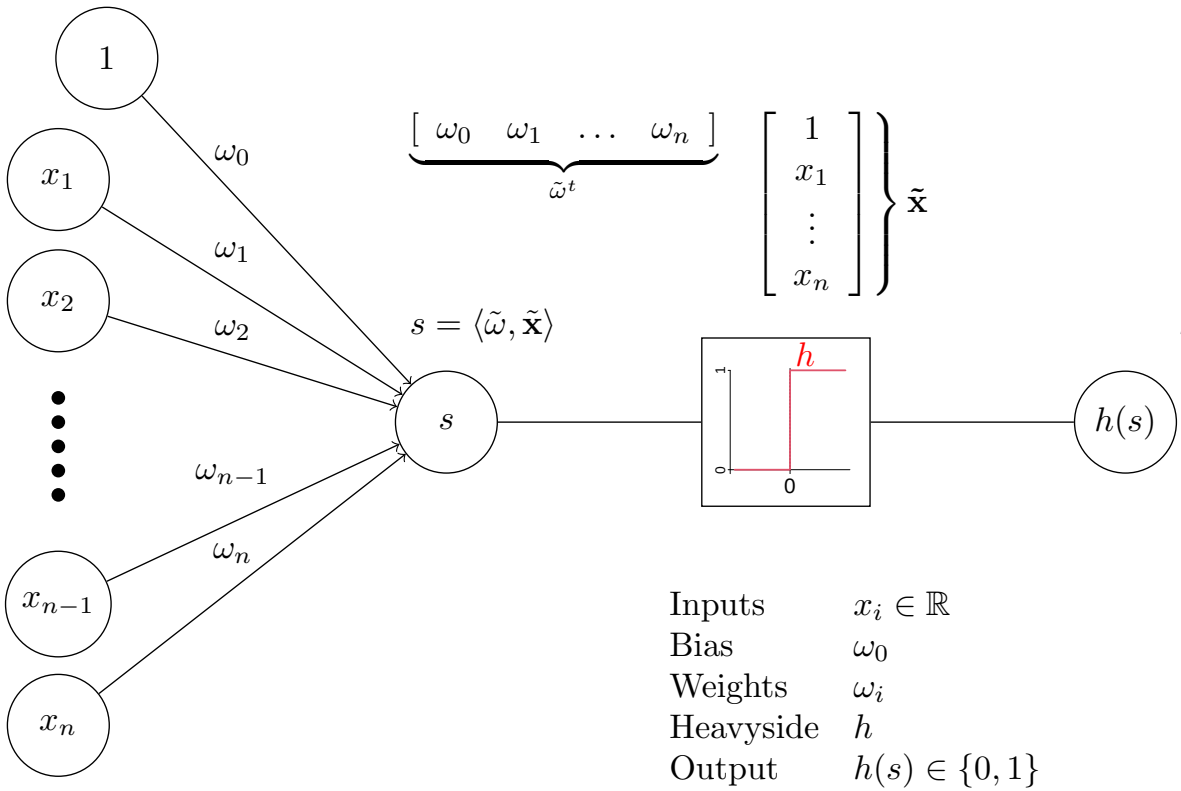
The perceptron



$$h(s) = \begin{cases} 1 & \text{if} \quad s \geq 0 \\ 0 & \text{if} \quad s < 0 \end{cases}$$

Inputs	$x_i \in \mathbb{R}$
Bias	ω_0
Weights	ω_i
Heavyside	h
Output	$h(s) \in \{0, 1\}$

The perceptron



Inputs	$x_i \in \mathbb{R}$
Bias	ω_0
Weights	ω_i
Heavyside	h
Output	$h(s) \in \{0, 1\}$

Learning algorithm

- We have N observations

$$((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N))$$

where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in \{0, 1\}$

- Set

$$\widetilde{\mathbf{x}}_i = \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}$$

- We are looking for a weight vector $\tilde{\omega} \in \mathbb{R}^{n+1}$ such as For all $i \geq N$,

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0 \text{ if } \mathbf{y}_i = 1$$

and

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle < 0 \text{ if } \mathbf{y}_i = 0$$

Perceptron algorithm

Choose an initial weight vector $\tilde{\omega}$
continue $\leftarrow$ **true**
while continue **do**
  continue $\leftarrow$ **false**
  for i from 1 to N **do**
    if

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0 \text{ and } \mathbf{y}_i = 0$$

    or

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle < 0 \text{ and } \mathbf{y}_i = 1$$

     then
      $\tilde{\omega} \leftarrow \tilde{\omega} + (\mathbf{y}_i - 1_{\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0})\mathbf{x}_i$
      continue $\leftarrow$ **true**
    end if
  end for
end while
Return $\tilde{\omega}$

Perceptron algorithm

Choose an initial weight vector $\tilde{\omega}$
continue $\leftarrow$ **true**
while continue **do**
  continue $\leftarrow$ **false**
  for i from 1 to N **do**
    if

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0 \text{ and } \mathbf{y}_i = 0$$

    or

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle < 0 \text{ and } \mathbf{y}_i = 1$$

     then
      $\tilde{\omega} \leftarrow \tilde{\omega} + (\mathbf{y}_i - 1_{\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0})\mathbf{x}_i$
      continue $\leftarrow$ **true**
    end if
  end for
end while
Return $\tilde{\omega}$

This algorithm converges in a finite number of iterations

Perceptron algorithm

Choose an initial weight vector $\tilde{\omega}$
continue $\leftarrow$ **true**
while continue **do**
  continue $\leftarrow$ **false**
  for i from 1 to N **do**
    if

$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0 \text{ and } \mathbf{y}_i = 0$$

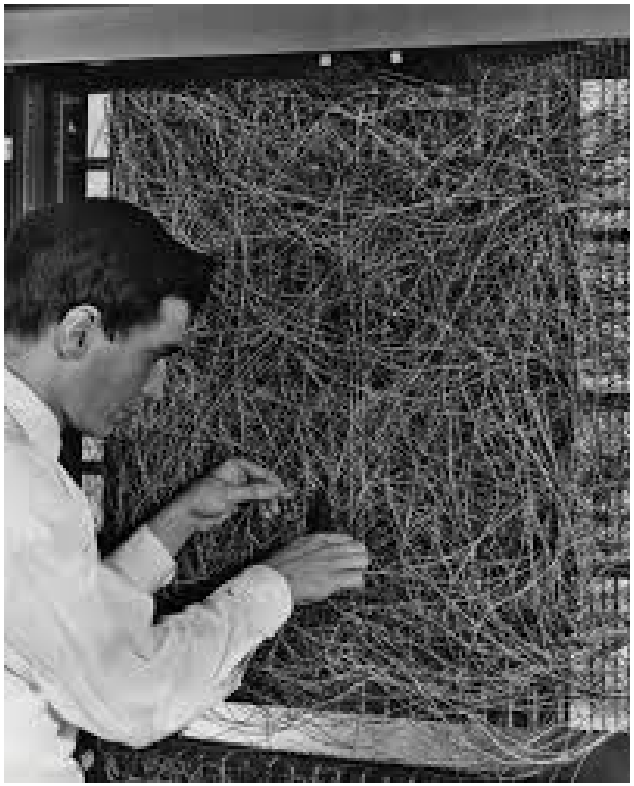
    or

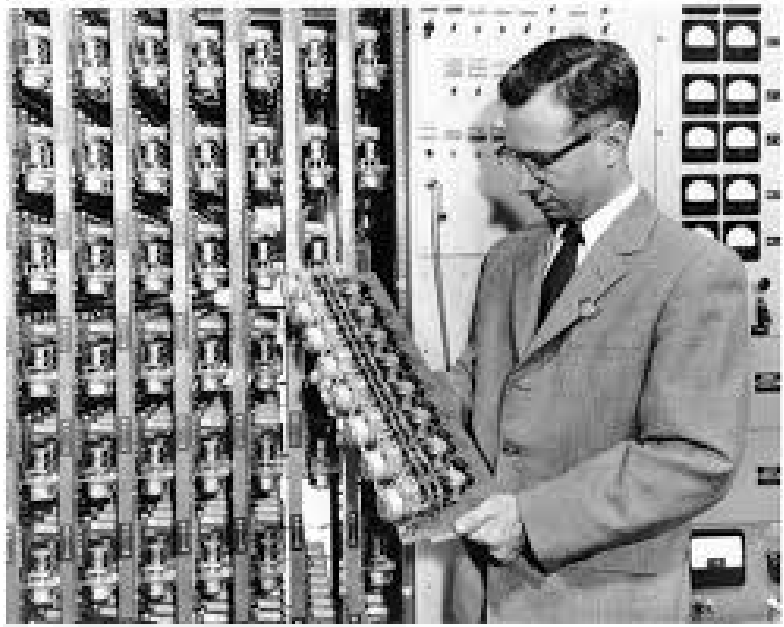
$$\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle < 0 \text{ and } \mathbf{y}_i = 1$$

     then
      $\tilde{\omega} \leftarrow \tilde{\omega} + (\mathbf{y}_i - 1_{\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle \geq 0})\mathbf{x}_i$
      continue $\leftarrow$ **true**
    end if
  end for
end while
Return $\tilde{\omega}$

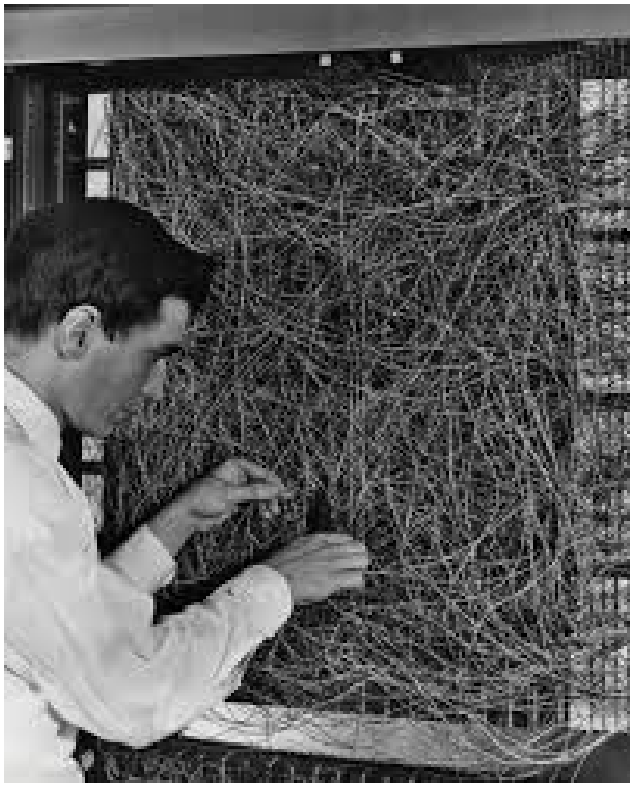
This algorithm converges in a finite number of iterations
if a solution exists!

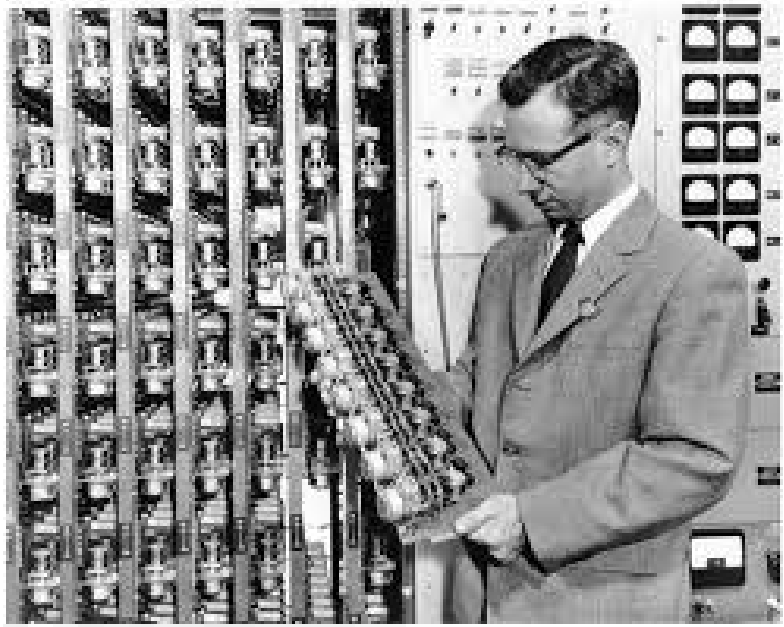
Implementation





Implementation





Method only adapted for linearly separable problems

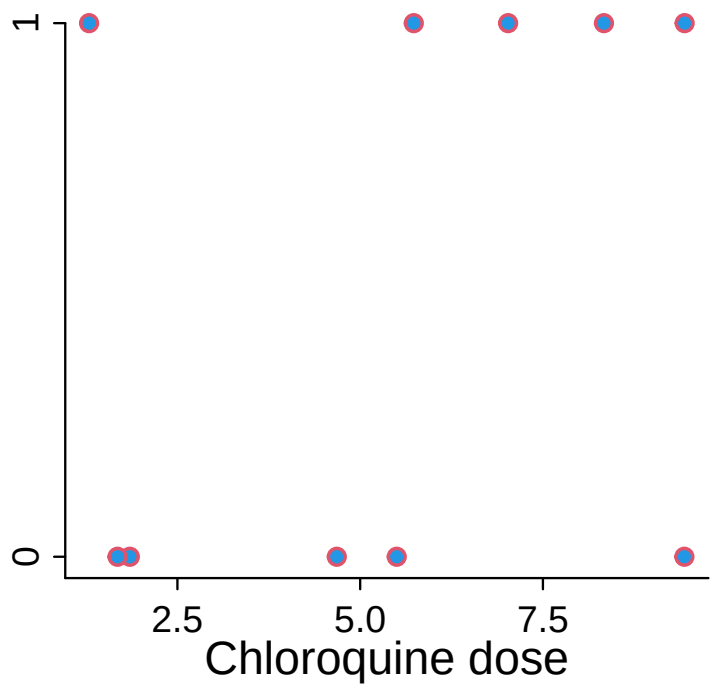
Regression for binary outcomes

Probit model (Bliss and Fisher, 1935) and Logistic Regression (Berkson, 1944)

$\mathbf{y}_i \in \{0, 1\}$ is a realization of a Bernoulli distribution with probability

$$p_i(\tilde{\omega}) = h(\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle)$$

for a link function h



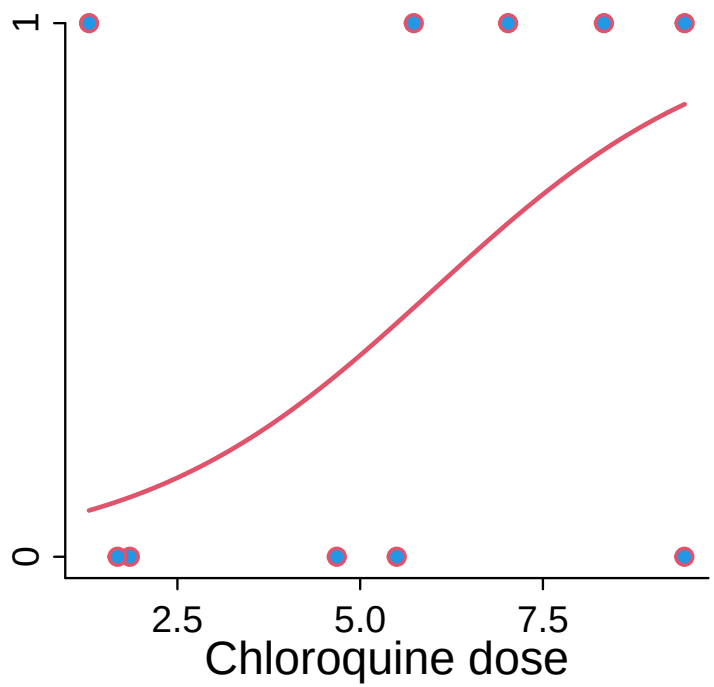
Regression for binary outcomes

Probit model (Bliss and Fisher, 1935) and Logistic Regression (Berkson, 1944)

$\mathbf{y}_i \in \{0, 1\}$ is a realization of a Bernoulli distribution with probability

$$p_i(\tilde{\omega}) = h(\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle)$$

for a link function h



Regression for binary outcomes

Probit model (Bliss and Fisher, 1935) and Logistic Regression (Berkson, 1944)

$\mathbf{y}_i \in \{0, 1\}$ is a realization of a Bernoulli distribution with probability

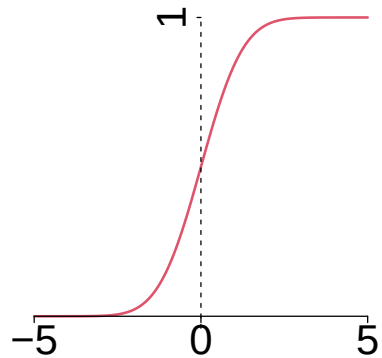
$$p_i(\tilde{\omega}) = h(\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle)$$

for a link function h

- Probit model

$$h = G$$

the c.d.f of the Gaussian



Regression for binary outcomes

Probit model (Bliss and Fisher, 1935) and Logistic Regression (Berkson, 1944)

$\mathbf{y}_i \in \{0, 1\}$ is a realization of a Bernoulli distribution with probability

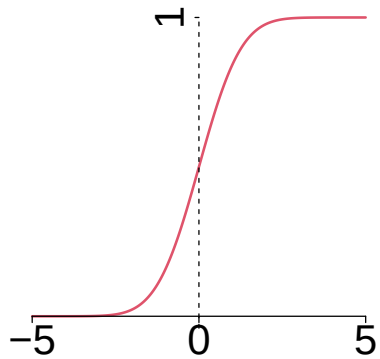
$$p_i(\tilde{\omega}) = h(\langle \tilde{\omega}, \widetilde{\mathbf{x}}_i \rangle)$$

for a link function h

- Probit model

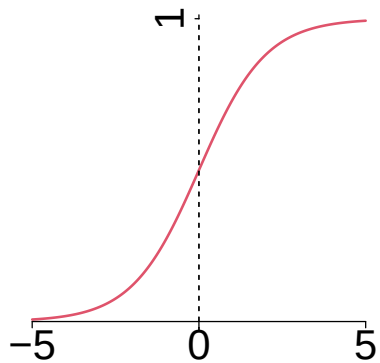
$$h = G$$

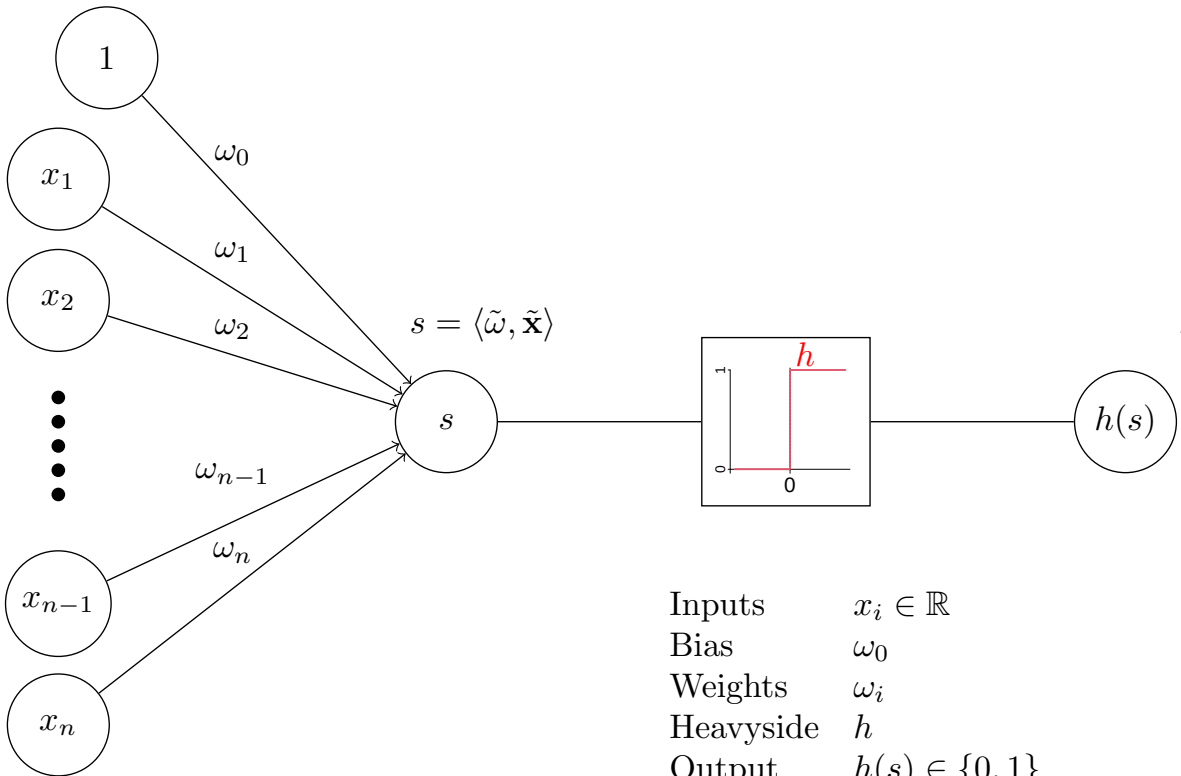
the c.d.f of the Gaussian



- Logistic model

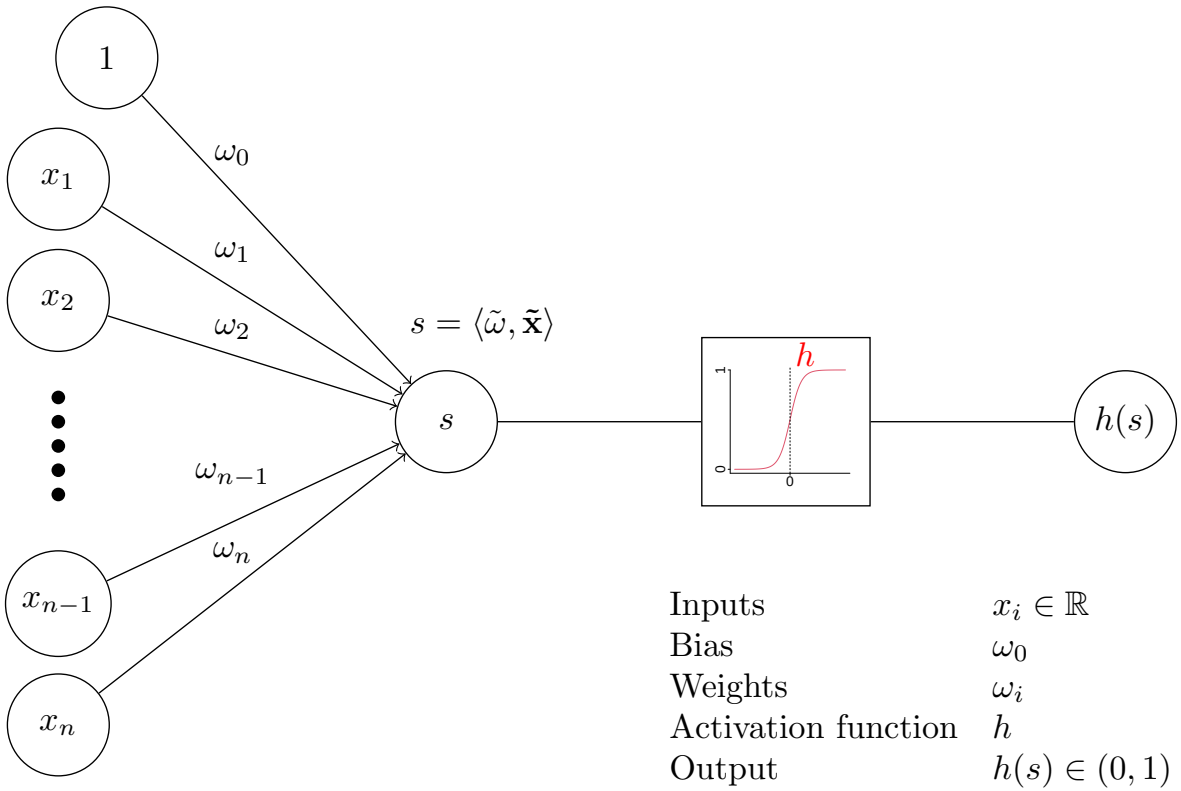
$$h(x) = \frac{1}{1 + e^{-x}}$$





$$h(s) = \begin{cases} 1 & \text{if} \quad s \geq 0 \\ 0 & \text{if} \quad s < 0 \end{cases}$$

Inputs	$x_i \in \mathbb{R}$
Bias	ω_0
Weights	ω_i
Heavyside	h
Output	$h(s) \in \{0, 1\}$



Inputs	$x_i \in \mathbb{R}$
Bias	ω_0
Weights	ω_i
Activation function	h
Output	$h(s) \in (0, 1)$

Inference by maximum-likelihood

The likelihood of observation i is

$$P(\, Y = \mathbf{y}_i | \mathbf{x}_i) = p_i(\tilde{\omega})^{\mathbf{y}_i} (1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

Inference by maximum-likelihood

The likelihood of observation i is

$$P(\,Y = \mathbf{y}_i|\mathbf{x}_i) = p_i(\tilde{\omega})^{\mathbf{y}_i}(1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

For N independant observations

$$\text{Likelihood}(\tilde{\omega}; \mathbf{y}_1, \ldots, \mathbf{y}_N) = \prod_{i=1}^{N} p_i(\tilde{\omega})^{\mathbf{y}_i}(1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

Inference by maximum-likelihood

The likelihood of observation i is

$$P(Y = \mathbf{y}_i | \mathbf{x}_i) = p_i(\tilde{\omega})^{\mathbf{y}_i}(1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

For N independant observations

$$\text{Likelihood}(\tilde{\omega}; \mathbf{y}_1, \ldots, \mathbf{y}_N) = \prod_{i=1}^{N} p_i(\tilde{\omega})^{\mathbf{y}_i}(1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

It is equivalent to maximize the log-likelihood

$$\sum_{i=1}^{N} [\mathbf{y}_i \log p_i(\tilde{\omega}) + (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))]$$

Inference by maximum-likelihood

The likelihood of observation i is

$$P(Y = \mathbf{y}_i|\mathbf{x}_i) = p_i(\tilde{\omega})^{\mathbf{y}_i}(1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

For N independant observations

$$\text{Likelihood}(\tilde{\omega}; \mathbf{y}_1, \ldots, \mathbf{y}_N) = \prod_{i=1}^{N} p_i(\tilde{\omega})^{\mathbf{y}_i}(1 - p_i(\tilde{\omega}))^{1-\mathbf{y}_i}$$

It is equivalent to maximize the log-likelihood

$$\sum_{i=1}^{N} [\mathbf{y}_i \log p_i(\tilde{\omega}) + (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))]$$

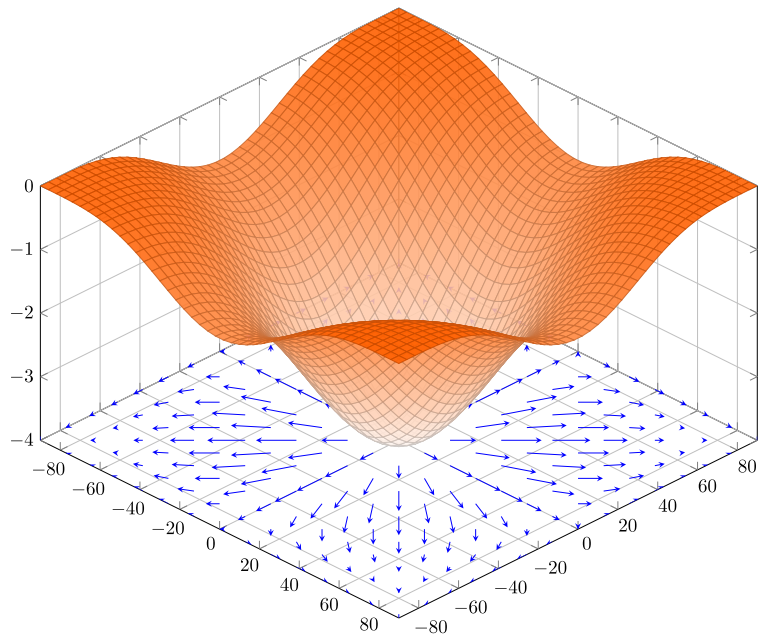
or to minimize the binary cross-entropy

$$\mathcal{L}(\tilde{\omega}) = -\sum_{i=1}^{N} [\mathbf{y}_i \log p_i(\tilde{\omega}) + (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))]$$

Gradient descent

The gradient

$$\nabla \mathcal{L}(\tilde{\omega}) = \begin{bmatrix} \frac{d\mathcal{L}}{d\omega_0}(\tilde{\omega}) \\ \vdots \\ \frac{d\mathcal{L}}{d\omega_n}(\tilde{\omega}) \end{bmatrix}$$



Credits: Martin Thoma

Gradient descent

- Initialisation: choose

$$\tilde{\omega}_{(0)} \in \mathbb{R}^{n+1}$$

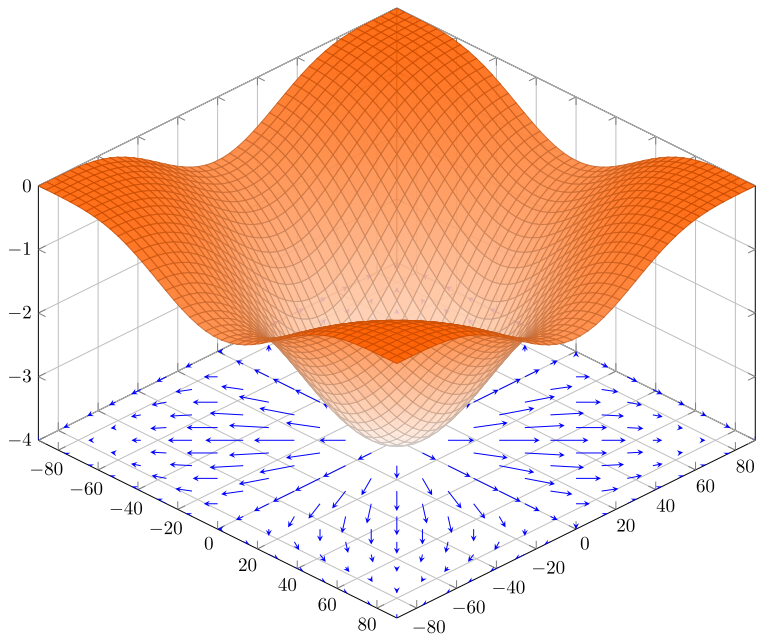
- Iterate until convergence

$$\tilde{\omega}_{(t+1)} = \tilde{\omega}_{(t)} - \frac{\eta}{N} \nabla \mathcal{L} \left(\tilde{\omega}_{(t)} \right)$$

$\eta > 0$ is the learning rate chosen by the user

The gradient

$$\nabla \mathcal{L}(\tilde{\omega}) = \begin{bmatrix} \frac{d\mathcal{L}}{d\omega_0}(\tilde{\omega}) \\ \vdots \\ \frac{d\mathcal{L}}{d\omega_n}(\tilde{\omega}) \end{bmatrix}$$



Credits: Martin Thoma

Exercise

Compute the gradient of the cost function

$$\mathcal{L}(\tilde{\omega}) = -\sum_{i=1}^{N} [\mathbf{y}_i \log p_i(\tilde{\omega}) + (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))]$$

where

$$p_i(\tilde{\omega}) = h(w_0 + w_1 x_{1,i})$$

and

$$h(x) = \frac{1}{1 + e^{-x}}$$

Exercise

Compute the gradient of the cost function

$$\mathcal{L}(\tilde{\omega}) = -\sum_{i=1}^{N} [\mathbf{y}_i \log p_i(\tilde{\omega}) + (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))]$$

where

$$p_i(\tilde{\omega}) = h(w_0 + w_1 x_{1,i})$$

and

$$h(x) = \frac{1}{1 + e^{-x}}$$

Hint: use the chain rule $(f \circ g)'(x) = f'(g(x))g'(x)$

Solution (1/2)

Set

$$\mathcal{L}_i(\tilde{\omega}) = -\mathbf{y}_i \log p_i(\tilde{\omega}) - (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))$$

Solution (1/2)

Set

$$\mathcal{L}_i(\tilde{\omega}) = -\mathbf{y}_i \log p_i(\tilde{\omega}) - (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))$$

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\frac{\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{1 - \mathbf{y}_i}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

Solution (1/2)

Set

$$\mathcal{L}_i(\tilde{\omega}) = -\mathbf{y}_i \log p_i(\tilde{\omega}) - (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))$$

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\frac{\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{1 - \mathbf{y}_i}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

$$\frac{d\mathcal{L}_i}{dw_1}(\tilde{\omega}) = -\frac{x_{1,i}\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{x_{1,i}(1 - \mathbf{y}_i)}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

Solution (1/2)

Set

$$\mathcal{L}_i(\tilde{\omega}) = -\mathbf{y}_i \log p_i(\tilde{\omega}) - (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))$$

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\frac{\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{1 - \mathbf{y}_i}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

$$\frac{d\mathcal{L}_i}{dw_1}(\tilde{\omega}) = -\frac{x_{1,i}\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{x_{1,i}(1 - \mathbf{y}_i)}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

with

$$h'(x) = h(x)(1 - h(x))$$

Solution (1/2)

Set

$$\mathcal{L}_i(\tilde{\omega}) = -\mathbf{y}_i \log p_i(\tilde{\omega}) - (1 - \mathbf{y}_i) \log(1 - p_i(\tilde{\omega}))$$

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\frac{\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{1 - \mathbf{y}_i}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

$$\frac{d\mathcal{L}_i}{dw_1}(\tilde{\omega}) = -\frac{x_{1,i}\mathbf{y}_i}{p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i}) + \frac{x_{1,i}(1 - \mathbf{y}_i)}{1 - p_i(\tilde{\omega})} h'(w_0 + w_1 x_{1,i})$$

with

$$h'(x) = h(x)(1 - h(x))$$

We can use

$$h'(w_0 + w_1 x_{1,i}) = p_i(\tilde{\omega})(1 - p_i(\tilde{\omega}))$$

Solution (2/2)

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\mathbf{y}_i(1 - p_i(\tilde{\omega})) + (1 - \mathbf{y}_i)p_i(\tilde{\omega})$$

Solution (2/2)

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\mathbf{y}_i(1 - p_i(\tilde{\omega})) + (1 - \mathbf{y}_i)p_i(\tilde{\omega}) = p_i(\tilde{\omega}) - \mathbf{y}_i$$

Solution (2/2)

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\mathbf{y}_i(1 - p_i(\tilde{\omega})) + (1 - \mathbf{y}_i)p_i(\tilde{\omega}) = p_i(\tilde{\omega}) - \mathbf{y}_i$$

$$\frac{d\mathcal{L}_i}{dw_1}(\tilde{\omega}) = x_{1,i}(p_i(\tilde{\omega}) - \mathbf{y}_i)$$

Solution (2/2)

$$\frac{d\mathcal{L}_i}{dw_0}(\tilde{\omega}) = -\mathbf{y}_i(1 - p_i(\tilde{\omega})) + (1 - \mathbf{y}_i)p_i(\tilde{\omega}) = p_i(\tilde{\omega}) - \mathbf{y}_i$$

$$\frac{d\mathcal{L}_i}{dw_1}(\tilde{\omega}) = x_{1,i}(p_i(\tilde{\omega}) - \mathbf{y}_i)$$

$$\nabla\mathcal{L}(\tilde{\omega}) = \widetilde{X}\mathbf{E}$$

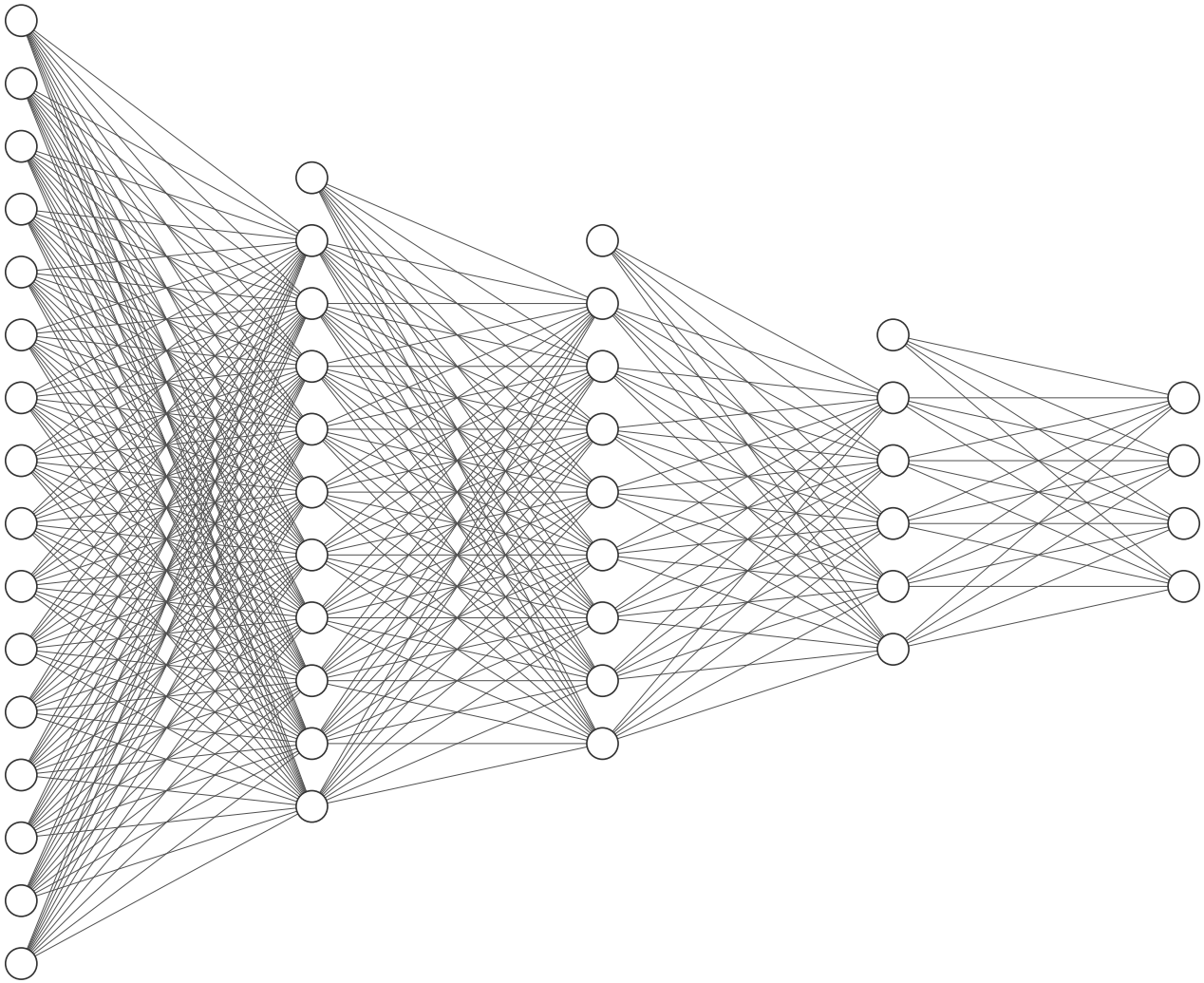
with

$$\widetilde{X} = \begin{bmatrix} 1 & \dots & 1 \\ x_{1,1} & \dots & x_{1,N} \end{bmatrix} \text{ and } \mathbf{E} = \begin{bmatrix} p_1(\tilde{\omega}) - \mathbf{y}_1 \\ \vdots \\ p_N(\tilde{\omega}) - \mathbf{y}_N \end{bmatrix}$$

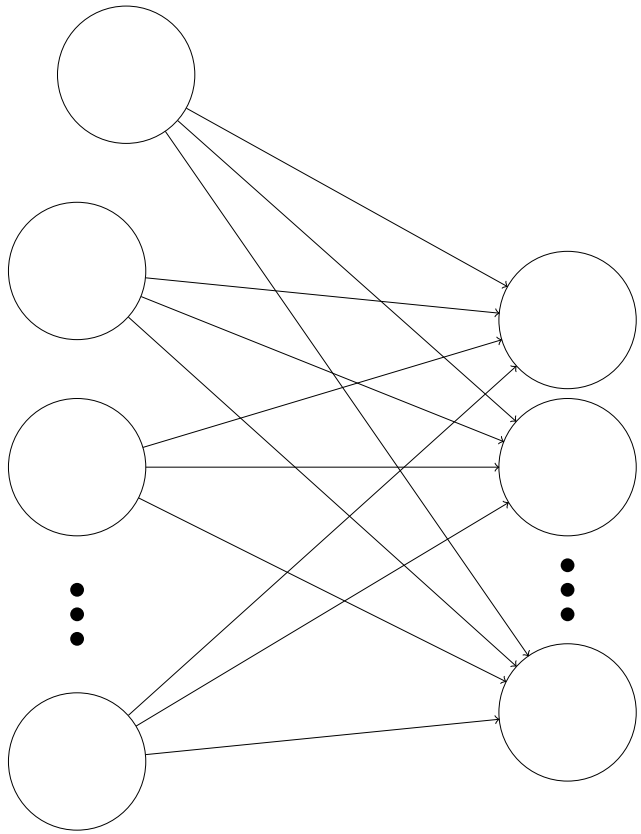
Neural network

- A neural network is a graph in which outputs of neurons (nodes) are become the inputs of some other neurons (oriented graph)
- In this day, we will restrict to a particular subclass of the neural networks, the feed-forward neural networks (no cycle)
- For convenience, we will use multi-layer perceptron (MLP)

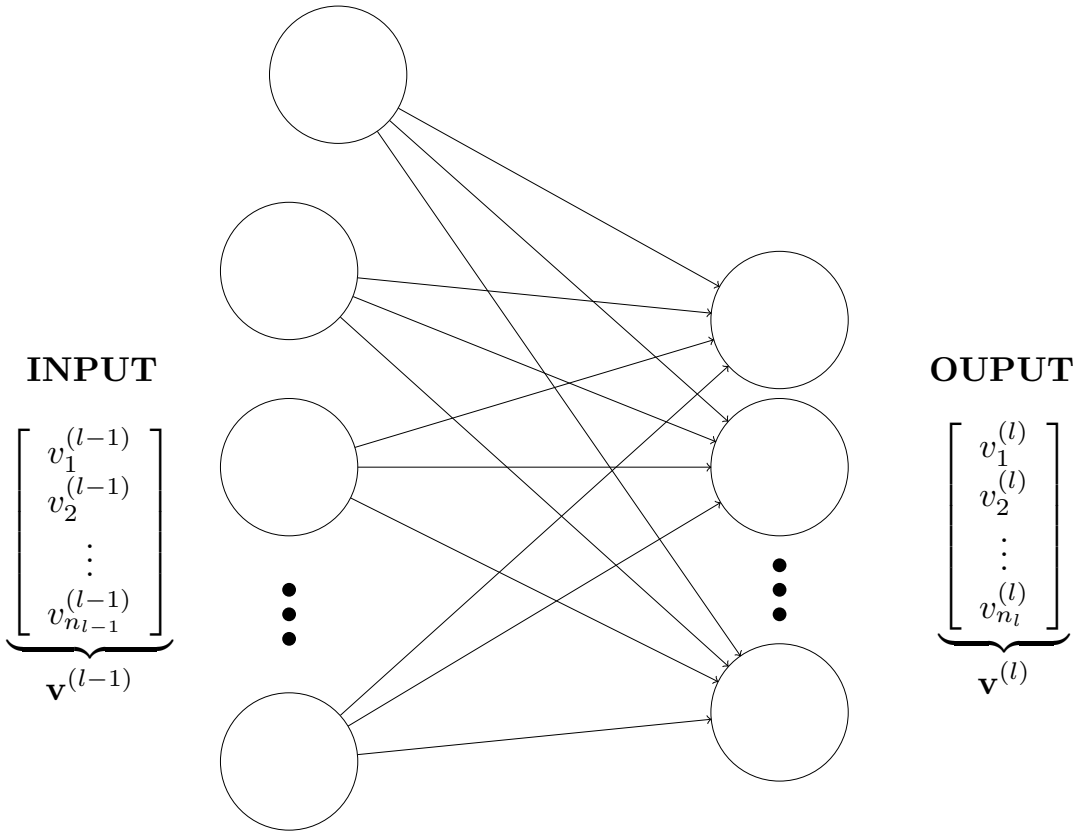
Multi-Layer perceptron



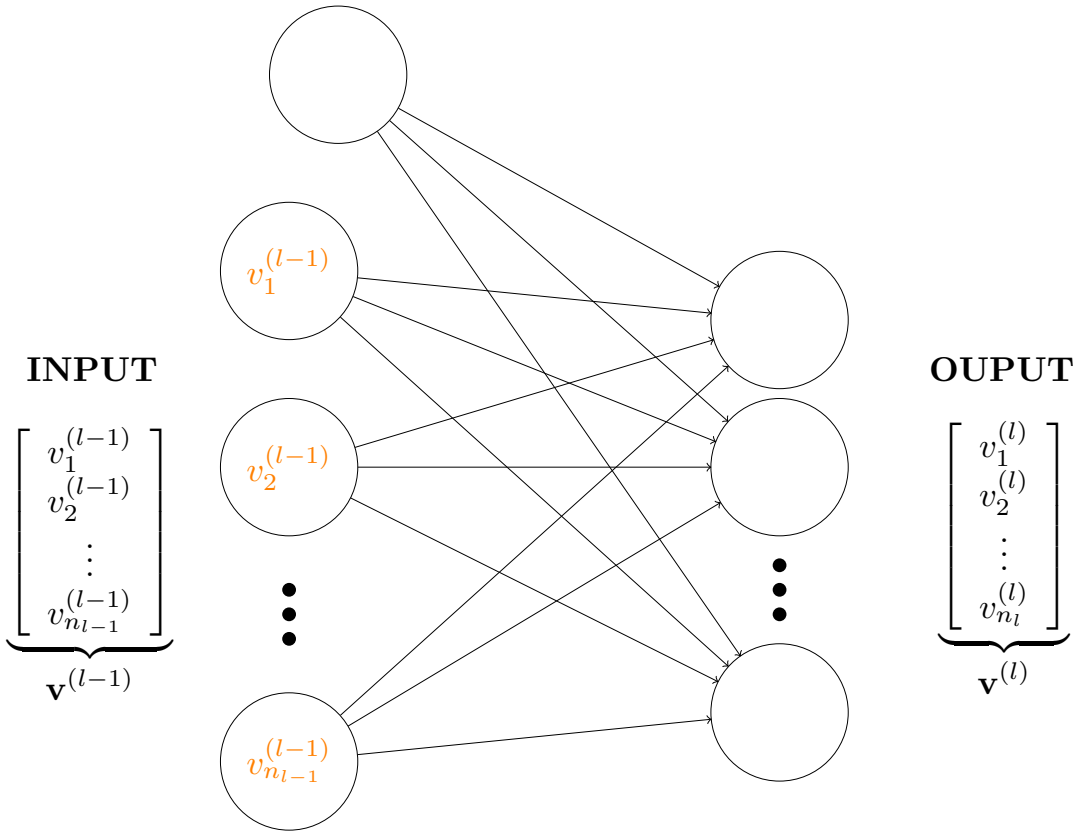
One layer



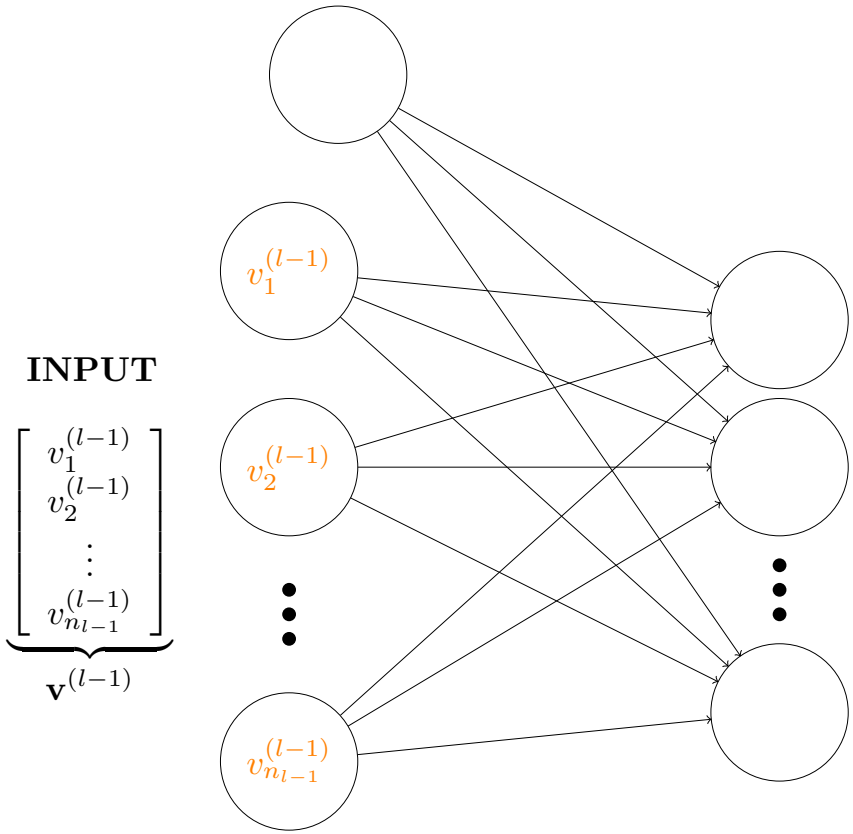
One layer


INPUT
$\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}$
$\mathbf{v}^{(l-1)}$
OUPUT
$\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}$
$\mathbf{v}^{(l)}$

One layer

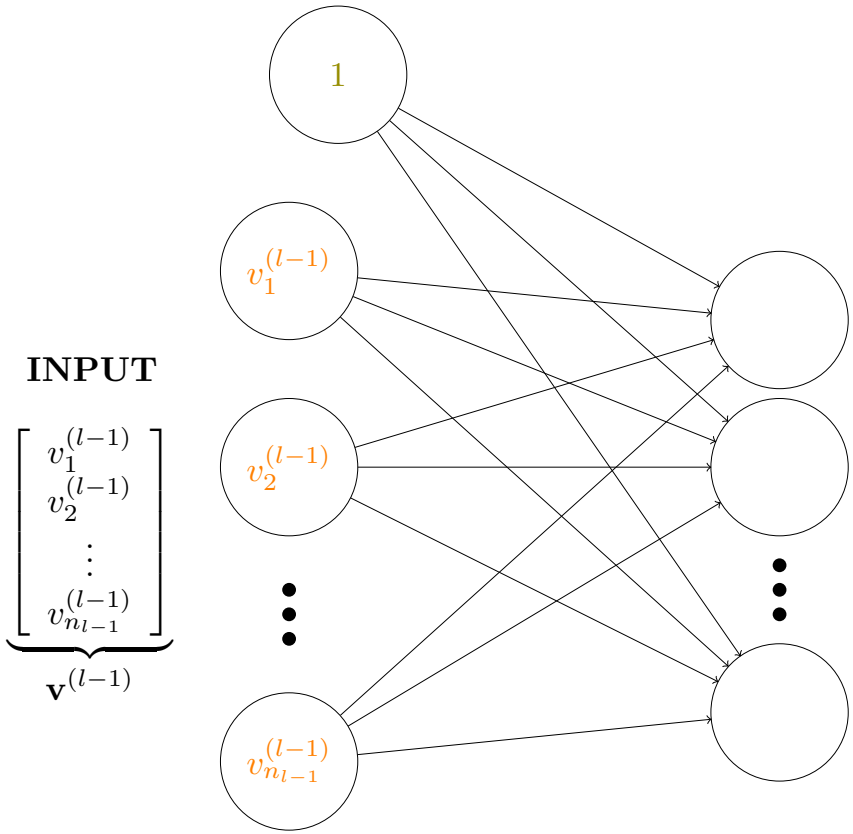

INPUT
$\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}$
$\mathbf{v}^{(l-1)}$
$v_1^{(l-1)}$
$v_2^{(l-1)}$
$v_{n_{l-1}}^{(l-1)}$
OUPUT
$\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}$
$\mathbf{v}^{(l)}$

One layer



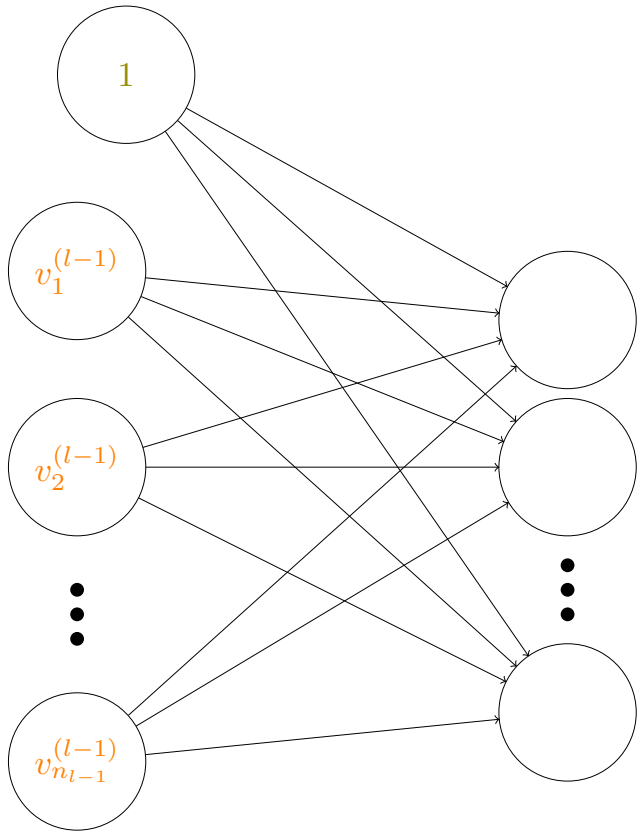
3 steps:

One layer



3 steps:
1) Stack a 1 to the input

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

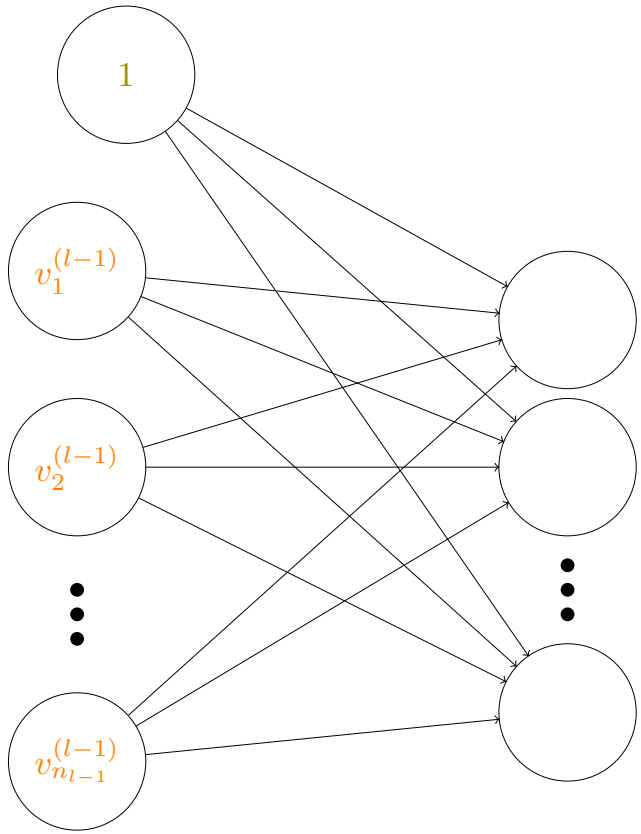
OUPUT

$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}}$$

3 steps:
1) Stack a 1 to the input

$$\underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

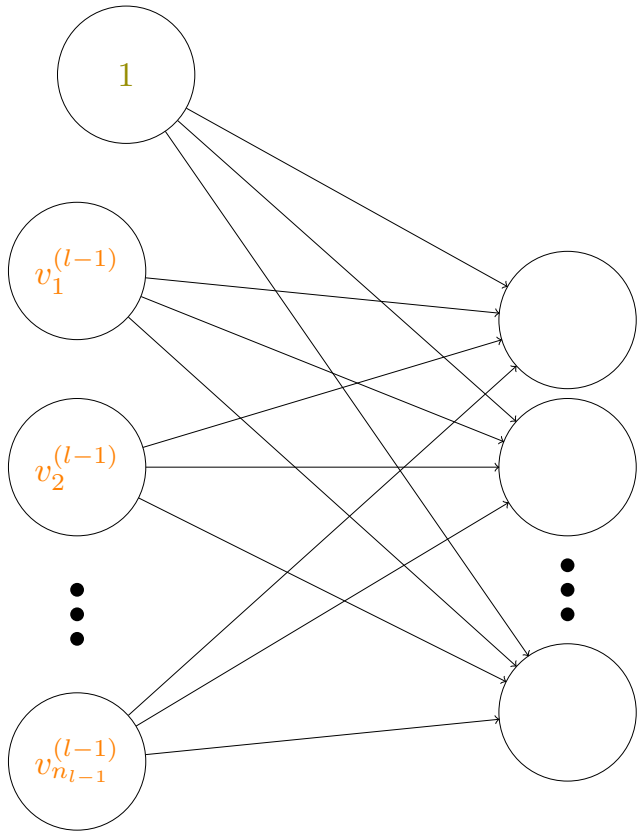
OUPUT

$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}}$$

3 steps:
1) Stack a 1 to the input

$$\underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

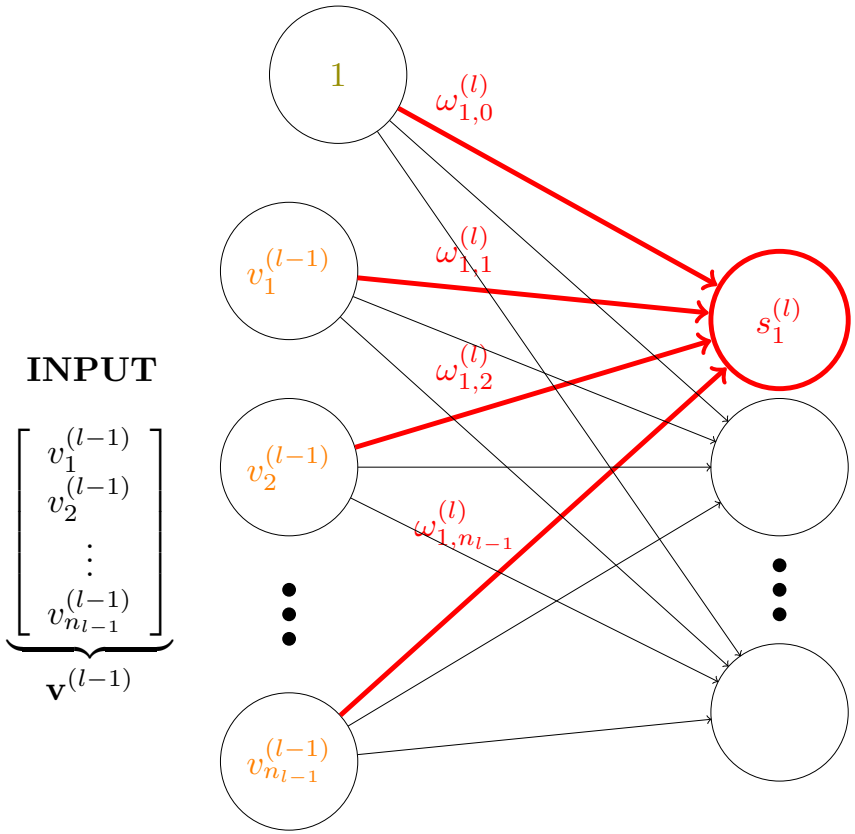
OUPUT

$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}}$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)}\tilde{\mathbf{v}}^{(l-1)}$

$$\underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

OUPUT

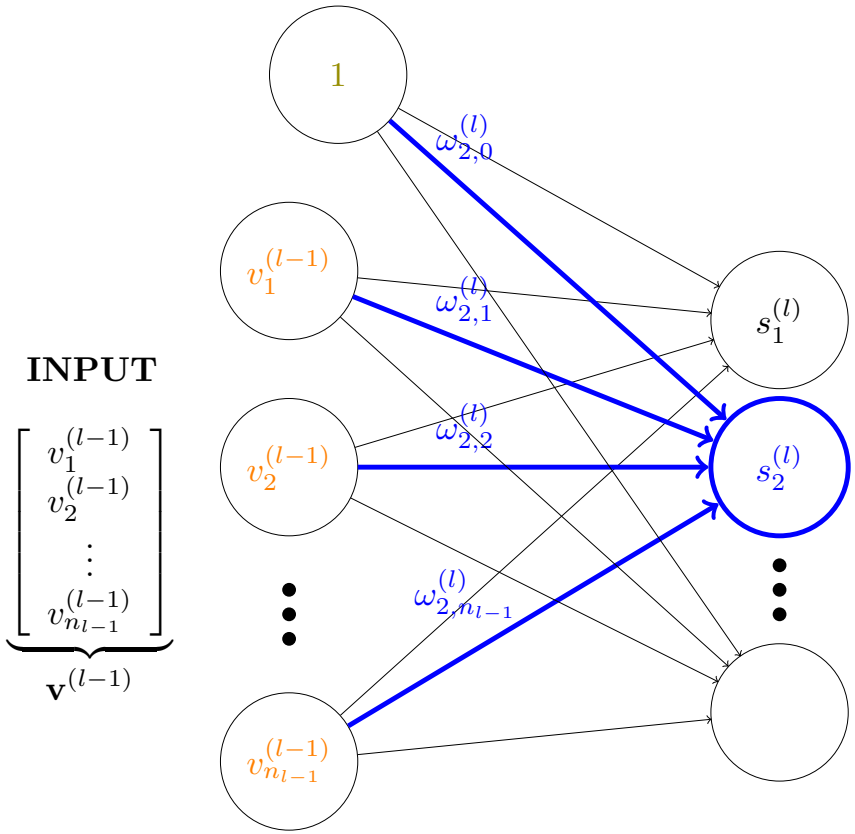
$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}}$$

$$s_1^{(l)} = \omega_{1,0}^{(l)} \cdot 1 + \sum_{j=1}^{n_{l-1}} \omega_{1,j}^{(l)} v_j^{(l-1)}$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)} \tilde{\mathbf{v}}^{(l-1)}$

$$\underbrace{\begin{bmatrix} s_1^{(l)} \\ \\ \\ \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer

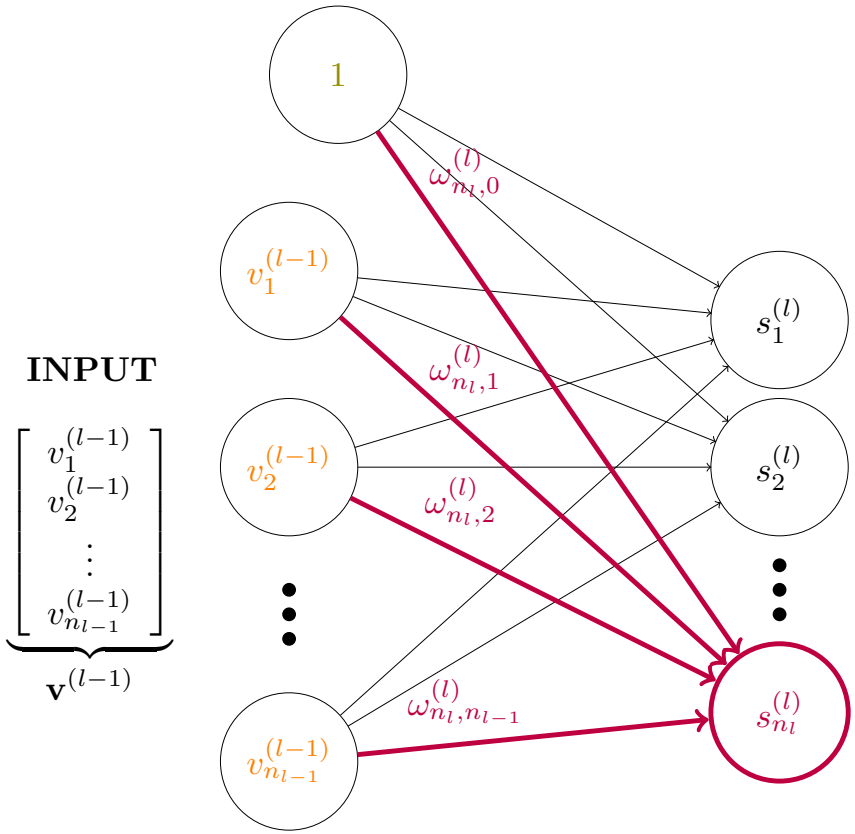


$$s_2^{(l)} = \omega_{2,0}^{(l)} \cdot 1 + \sum_{j=1}^{n_{l-1}} \omega_{2,j}^{(l)} v_j^{(l-1)}$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)} \tilde{\mathbf{v}}^{(l-1)}$

$$\underbrace{\begin{bmatrix} s_1^{(l)} \\ s_2^{(l)} \\ \\ \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

OUPUT

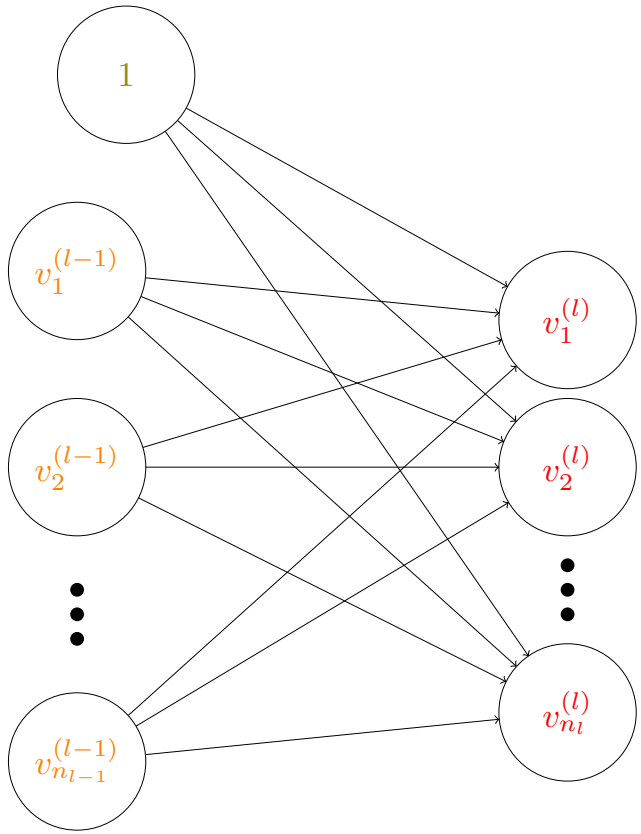
$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}}$$

$$s^{(l)}_{n_l} = \omega^{(l)}_{n_l,0}.1 + \sum_{j=1}^{n_{l-1}} \omega^{(l)}_{n_l,j} v^{(l-1)}_j$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)} \tilde{\mathbf{v}}^{(l-1)}$

$$\underbrace{\begin{bmatrix} s_1^{(l)} \\ s_2^{(l)} \\ \vdots \\ s_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

OUPUT

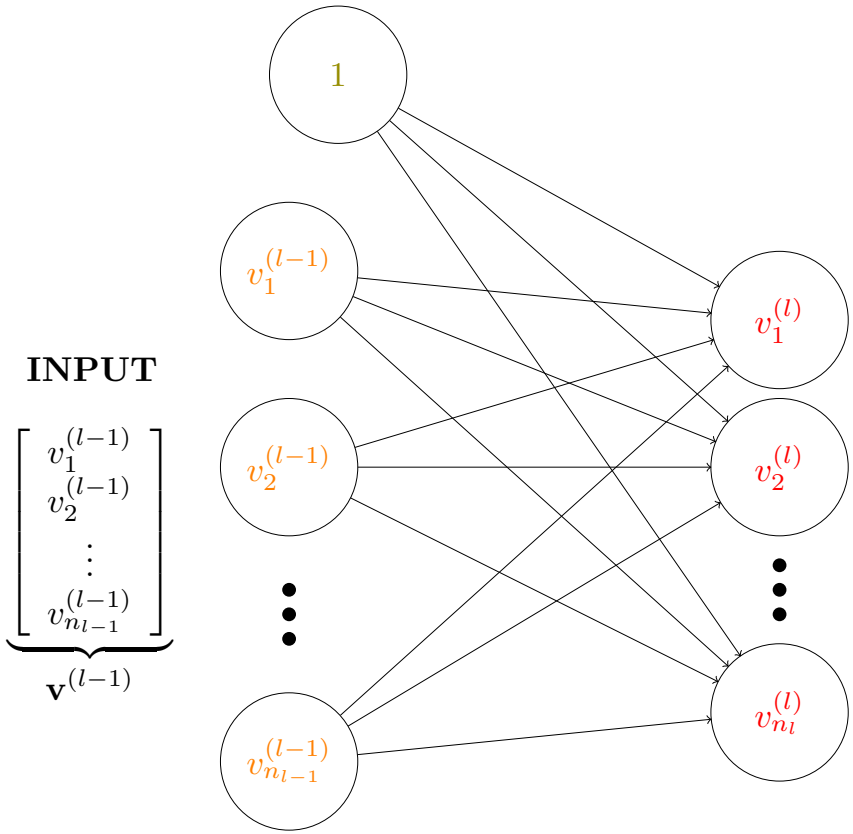
$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}} = \underbrace{\begin{bmatrix} h_{(l)}\left(s_1^{(l)}\right) \\ h_{(l)}\left(s_2^{(l)}\right) \\ \vdots \\ h_{(l)}\left(s_{n_l}^{(l)}\right) \end{bmatrix}}_{h_{(l)}(\mathbf{s}^{(l)})}$$

$$s_i^{(l)} = \omega_{i,0}^{(l)} \cdot 1 + \sum_{j=1}^{n_{l-1}} \omega_{i,j}^{(l)} v_j^{(l-1)}$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)} \tilde{\mathbf{v}}^{(l-1)}$
3) Return $\mathbf{v}^{(l)} = h_{(l)}(\mathbf{s}^{(l)})$

$$\underbrace{\begin{bmatrix} s_1^{(l)} \\ s_2^{(l)} \\ \vdots \\ s_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

One layer



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

OUPUT

$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}} = \underbrace{\begin{bmatrix} h_{(l)}\left(s_1^{(l)}\right) \\ h_{(l)}\left(s_2^{(l)}\right) \\ \vdots \\ h_{(l)}\left(s_{n_l}^{(l)}\right) \end{bmatrix}}_{h_{(l)}(\mathbf{s}^{(l)})}$$

2 attributes for layer l:
1) A weight matrix $\tilde{W}^{(l)}$
2) An activation function $h_{(l)}$

$$s_i^{(l)} = \omega_{i,0}^{(l)} \cdot 1 + \sum_{j=1}^{n_{l-1}} \omega_{i,j}^{(l)} v_j^{(l-1)}$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)} \tilde{\mathbf{v}}^{(l-1)}$
3) Return $\mathbf{v}^{(l)} = h_{(l)}(\mathbf{s}^{(l)})$

$$\underbrace{\begin{bmatrix} s_1^{(l)} \\ s_2^{(l)} \\ \vdots \\ s_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

Summary

The l^{th} layer of perceptrons is a function $f^{(l)}_{\tilde{W}^{(l)}}$ from $\mathbb{R}^{n_{l-1}}$ to $\mathbb{R}^{n_l}$ defined by:

$$\mathbf{v}^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(\mathbf{v}^{(l-1)})$$

Summary

The l^{th} layer of perceptrons is a function $f^{(l)}_{\tilde{W}^{(l)}}$ from $\mathbb{R}^{n_{l-1}}$ to $\mathbb{R}^{n_l}$ defined by:

$$\mathbf{v}^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(\mathbf{v}^{(l-1)}) = h_{(l)}(\tilde{W}^{(l)}\tilde{\mathbf{v}}^{(l-1)})$$

Summary

The l^{th} layer of perceptrons is a function $f^{(l)}_{\tilde{W}^{(l)}}$ from $\mathbb{R}^{n_{l-1}}$ to $\mathbb{R}^{n_l}$ defined by:

$$\mathbf{v}^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(\mathbf{v}^{(l-1)}) = h_{(l)}(\tilde{W}^{(l)}\tilde{\mathbf{v}}^{(l-1)}) = h_{(l)}(\mathbf{b}^{(l)} + W^{(l)}\mathbf{v}^{(l-1)})$$

with $W^{(l)} = \begin{bmatrix} \omega^{(l)}_{1,1} & \dots & \omega^{(l)}_{1,n_{l-1}} \\ \vdots & \ddots & \vdots \\ \omega^{(l)}_{n_l,1} & \dots & \omega^{(l)}_{n_l,n_{l-1}} \end{bmatrix}$ and $\mathbf{b}^{(l)} = \begin{bmatrix} \omega^{(l)}_{1,0} \\ \vdots \\ \omega^{(l)}_{n_l,0} \end{bmatrix}$

Summary

Matricial form

The l^{th} layer of perceptrons is a function $f^{(l)}_{\tilde{W}^{(l)}}$ from $\mathbb{R}^{n_{l-1}\times N}$ to $\mathbb{R}^{n_l \times N}$ defined by:

$$V^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(\, V^{(l-1)})$$

Summary

Matricial form

The l^{th} layer of perceptrons is a function $f^{(l)}_{\tilde{W}^{(l)}}$ from $\mathbb{R}^{n_{l-1}\times N}$ to $\mathbb{R}^{n_l \times N}$ defined by:

$$V^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(\, V^{(l-1)}) = h_{(l)}(\, \tilde{W}^{(l)}\, \tilde{V}^{(l-1)})$$

with

$$V^{(l)} = \begin{bmatrix} v^{(l)}_{1,1} & \cdots & v^{(l)}_{1,N} \\ \vdots & \ddots & \vdots \\ v^{(l)}_{n_l,1} & \cdots & v^{(l)}_{n_l,N} \end{bmatrix} \text{ and } \tilde{V}^{(l)} = \begin{bmatrix} 1 & \cdots & 1 \\ v^{(l)}_{1,1} & \cdots & v^{(l)}_{1,N} \\ \vdots & \ddots & \vdots \\ v^{(l)}_{n_l,1} & \cdots & v^{(l)}_{n_l,N} \end{bmatrix}$$

Multi-Layer Perceptron

- Feed the input vector $\mathbf{x}$ to the input layer:

$$\mathbf{v}^{(0)} = \mathbf{x}$$

- For $l = 1, \ldots, L$ compute

$$\mathbf{v}^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(\mathbf{v}^{(l-1)})$$

- Return the content of the output layer

$$\mathbf{v}^{(L)} = f_{\mathbf{W}}(\mathbf{x})$$

$\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(L-1)}$ are the hidden layers

Matricial form

- Feed the input matrix X to the input layer:

$$V^{(0)} = X$$

- For $l = 1, \ldots, L$ compute

$$V^{(l)} = f^{(l)}_{\tilde{W}^{(l)}}(V^{(l-1)})$$

- Return the content of the output layer

$$V^{(L)} = f_{\mathbf{W}}(X)$$

$V^{(1)}, \ldots, V^{(L-1)}$ are the hidden layers

Capacity of a neural network (2)

Universal approximation theorem (Hornik, 1991)

Let f be a continuous function $f : [0,1]^n \to \mathbb{R}$ and $\varepsilon > 0$.
Let $h^{(1)}$ be a non-constant, increasing, bounded real function.
Then, there exists

- an integer q,
- a matrix of weights $\tilde{W}^{(1)} \in \mathbb{R}^{q \times n}$
- a vector of weights $\omega^{(2)} \in \mathbb{R}^q$
- a real $b^{(2)}$

such as for all $\mathbf{x} \in [0,1]^n$,

$$\left| f(\mathbf{x}) - b^{(2)} - \left\langle \omega^{(2)}, h^{(1)} \left(\langle \tilde{W}^{(1)}, \widetilde{\mathbf{x}} \rangle \right) \right\rangle \right| < \varepsilon$$

Capacity of a neural network (2)

Universal approximation theorem (Hornik, 1991)

Let f be a continuous function $f : [0,1]^n \to \mathbb{R}$ and $\varepsilon > 0$.
Let $h^{(1)}$ be a non-constant, increasing, bounded real function.
Then, there exists

- an integer q,
- a matrix of weights $\tilde{W}^{(1)} \in \mathbb{R}^{q \times n}$
- a vector of weights $\omega^{(2)} \in \mathbb{R}^q$
- a real $b^{(2)}$

such as for all $\mathbf{x} \in [0,1]^n$,

$$\left| f(\mathbf{x}) - b^{(2)} - \left\langle \omega^{(2)}, h^{(1)} \left(\langle \tilde{W}^{(1)}, \widetilde{\mathbf{x}} \rangle \right) \right\rangle \right| < \varepsilon$$

Any continuous real-valued function on a bounded subset of $\mathbb{R}^n$ can be approximated arbitrarily well by a MLP with one hidden layer (for appropriate activation function $h^{(1)}$)

Remarks

- The theorems are discovered after the practice
- They just give a flavour on what a NN can express
- The number of neurons rapidly increases with the complexity of the function
- Some versions exist with a fixed number of neurals in each layer but an increasing number of layers
- Capacity of a neural network is sometimes used as a starting hypothesis to prove the correctness of an algorithm (e.g Goodfellow et al., 2014, for the generative adversarial networks)

Loss functions

The cost function is a sum of loss functions δ between each prediction $f_{\mathbf{W}}(\mathbf{x}_i)$ and each label $\mathbf{y}_i$:

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N} \delta(\mathbf{y}_i, f_{\mathbf{W}}(\mathbf{x}_i))$$

Loss functions

The cost function is a sum of loss functions δ between each prediction $f_{\mathbf{W}}(\mathbf{x}_i)$ and each label $\mathbf{y}_i$:

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N} \delta(\mathbf{y}_i, f_{\mathbf{W}}(\mathbf{x}_i))$$

Regression

$$\delta(\mathbf{y}, f_{\mathbf{W}}(\mathbf{x})) = \sum_{k=1}^{N} (y_k - f_{\mathbf{W}}(\mathbf{x}))^2$$

- Sum of squares
- Likelihood for Gaussian noise
- Simplifications occur when the activation function of the output layer is the identity

Loss functions

The cost function is a sum of loss functions δ between each prediction $f_{\mathbf{W}}(\mathbf{x}_i)$ and each label $\mathbf{y}_i$:

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N} \delta(\mathbf{y}_i, f_{\mathbf{W}}(\mathbf{x}_i))$$

Binary classification

$$\delta(\mathbf{y}, f_{\mathbf{W}}(\mathbf{x})) = -\mathbf{y} \log(f_{\mathbf{W}}(\mathbf{x})) - (1 - \mathbf{y}) \log(1 - f_{\mathbf{W}}(\mathbf{x}))$$

- Binary cross-entropy
- Likelihood for Bernouilli
- Simplifications occur when the activation function of the output layer is the sigmoid (a.k.a logistic)

Loss functions

The cost function is a sum of loss functions δ between each prediction $f_{\mathbf{W}}(\mathbf{x}_i)$ and each label $\mathbf{y}_i$:

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N} \delta(\mathbf{y}_i, f_{\mathbf{W}}(\mathbf{x}_i))$$

Multiclass classification

$$\delta(\mathbf{y}, f_{\mathbf{W}}(\mathbf{x})) = -\sum_{k=1}^{K} y_k \log(f_{\mathbf{W}}(\mathbf{x})_k)$$

- Cross-entropy
- Likelihood for Multinomial distribution
- Simplifications occur when the activation function of the output layer is the softmax

Softmax activation function

- For a multiclass problem with K categories, one can use the softmax activation on the last layer:

$$\mathbf{v} = \text{Softmax}\left(\mathbf{s}\right) = \frac{1}{\sum_{i=1}^{n_L} e^{s_i}} \begin{bmatrix} e^{s_1} \\ \vdots \\ e^{s_{n_L}} \end{bmatrix}$$

Softmax activation function

- For a multiclass problem with K categories, one can use the softmax activation on the last layer:

$$\mathbf{v} = \text{Softmax}\left(\mathbf{s}\right) = \frac{1}{\sum_{i=1}^{n_L} e^{s_i}} \begin{bmatrix} e^{s_1} \\ \vdots \\ e^{s_{n_L}} \end{bmatrix}$$

- The result is an element of the simplex of dimension n_L:

$$\sum_{i=1}^{n_L} v_i = 1 \text{ and } v_i > 0$$

Train a neural-network

- To minimize the loss function $\mathcal{L}(\mathbf{W})$ with respect to the weights, we need to compute its gradient with respect to all the weights $\omega_{i,j}^{(l)}$
- We will compute the gradient for the loss of each training pair $(\mathbf{x}_i, \mathbf{y}_i)$ and then sum the results
- So we need to compute

$$\frac{d\delta}{d\omega_{i,j}^{(l)}}$$

- We have already done this work in the logistic regression case (for the weights of the last layer)
- The algorithm to generalize to all the layers is known as the back-propagation

Chain rule (1/4)

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

Chain rule (1/4)

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

$(f \circ g \circ h)'(x)$

Chain rule (1/4)

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

$$(f \circ g \circ h)'(x) = (f \circ g)'(h(x))h'(x)$$

Chain rule (1/4)

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

$$(f \circ g \circ h)'(x) = (f \circ g)'(h(x))h'(x) = f'(g(h(x)))g'(h(x))h'(x)$$

Chain rule (1/4)

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

$$(f \circ g \circ h)'(x) = (f \circ g)'(h(x))h'(x) = f'(g(h(x)))g'(h(x))h'(x)$$

Leibniz notation

$$x \xrightarrow{h} y \xrightarrow{g} u \xrightarrow{f} z$$

$$\frac{dz}{dx} = \frac{dz}{du}\frac{du}{dy}\frac{dy}{dx}$$

Chain rule (2/4)

Function of several variables

- Let consider a function f of two variables
- And two other functions x and y
- Consider the function u defined by $u(t) = f(x(t), y(t))$
- We have

$$\frac{du}{dt} = \frac{du}{dx}\frac{dx}{dt} + \frac{du}{dy}\frac{dy}{dt}$$

Chain rule (2/4)

Function of several variables

- Let consider a function f of two variables
- And two other functions x and y
- Consider the function u defined by $u(t) = f(x(t), y(t))$
- We have

$$\frac{du}{dt} = \frac{du}{dx}\frac{dx}{dt} + \frac{du}{dy}\frac{dy}{dt} = \left[\frac{du}{dx}, \frac{du}{dy}\right] \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}$$

Vectorial function

- Let consider a function $F : \mathbb{R}^m \to \mathbb{R}^p$ with

$$F(\mathbf{y}) = \begin{bmatrix} F_1(\mathbf{y}) \\ \vdots \\ F_p(\mathbf{y}) \end{bmatrix}$$

- The Jacobian of F is

$$J_F(\mathbf{y}) = \begin{bmatrix} \frac{dF_1}{dy_1}(\mathbf{y}) & \cdots & \frac{dF_1}{dy_m}(\mathbf{y}) \\ \vdots & \ddots & \vdots \\ \frac{dF_p}{dy_1}(\mathbf{y}) & \cdots & \frac{dF_p}{dy_m}(\mathbf{y}) \end{bmatrix}$$

- Now, let $H : \mathbb{R}^n \to \mathbb{R}^m$ a function
- Chain rule: the Jacobian of $F \circ H$ is

$$J_{F \circ H}(\mathbf{x}) = J_F(H(\mathbf{x}))J_H(\mathbf{x})$$

Vectorial function

Chain rule: the Jacobian of $F \circ H$ is

$$J_{F \circ H}(\mathbf{x}) = J_F(H(\mathbf{x}))J_H(\mathbf{x})$$

Leibniz notation

$$\mathbf{x} \xrightarrow{H} \mathbf{y} \xrightarrow{F} \mathbf{z}$$

$$\frac{d\mathbf{z}}{d\mathbf{x}} = \frac{d\mathbf{z}}{d\mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} =$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} =$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} =$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} =$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} =$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\mathbf{v}^{(L-2)}} \frac{d\mathbf{v}^{(L-2)}}{d\mathbf{s}^{(L-2)}} \frac{d\mathbf{s}^{(L-2)}}{d\omega_{i,j}^{(L-2)}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\mathbf{v}^{(L-2)}} \frac{d\mathbf{v}^{(L-2)}}{d\mathbf{s}^{(L-2)}} \frac{d\mathbf{s}^{(L-2)}}{d\omega_{i,j}^{(L-2)}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} = \underbrace{\frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}}_{E^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\mathbf{v}^{(L-2)}} \frac{d\mathbf{v}^{(L-2)}}{d\mathbf{s}^{(L-2)}} \frac{d\mathbf{s}^{(L-2)}}{d\omega_{i,j}^{(L-2)}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} = \underbrace{\underbrace{\frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}}_{E^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\mathbf{v}^{(L-2)}}}_{E^{(L-1)}} \frac{d\mathbf{v}^{(L-2)}}{d\mathbf{s}^{(L-2)}} \frac{d\mathbf{s}^{(L-2)}}{d\omega_{i,j}^{(L-2)}}$$

Back-propagation

$$\frac{d\delta}{d\omega_{i,j}^{(L)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-1)}} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(L-2)}} = \underbrace{\underbrace{\underbrace{\frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}}_{E^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}}}_{E^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\mathbf{v}^{(L-2)}} \frac{d\mathbf{v}^{(L-2)}}{d\mathbf{s}^{(L-2)}}}_{E^{(L-2)}} \frac{d\mathbf{s}^{(L-2)}}{d\omega_{i,j}^{(L-2)}}$$

Back-propagation

- Initialization: Compute

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- For l from L to 1
  - Compute

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}}$$

  - If $l \geq 1$, compute

$$E^{(l-1)} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\mathbf{v}^{(l-1)}} \frac{d\mathbf{v}^{(l-1)}}{d\mathbf{s}^{(l-1)}}$$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- $\frac{d\delta}{d\mathbf{v}^{(L)}} = \left[\frac{d\delta}{dv_1^{(L)}}, \ldots, \frac{d\delta}{dv_{n_L}^{(L)}} \right]$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- $\frac{d\delta}{d\mathbf{v}^{(L)}} = \left[\frac{d\delta}{dv_1^{(L)}}, \ldots, \frac{d\delta}{dv_{n_L}^{(L)}} \right]$
- $\frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} = \mathrm{Diag}(h'_{(L)}(\mathbf{s}^{(L)}))$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- $\frac{d\delta}{d\mathbf{v}^{(L)}} = \left[\frac{d\delta}{dv_1^{(L)}}, \ldots, \frac{d\delta}{dv_{n_L}^{(L)}} \right]$ $(1 \times n_L)$
- $\frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} = \mathrm{Diag}(h'_{(L)}(\mathbf{s}^{(L)}))$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- $\frac{d\delta}{d\mathbf{v}^{(L)}} = \left[\frac{d\delta}{dv_1^{(L)}}, \ldots, \frac{d\delta}{dv_{n_L}^{(L)}} \right]$ $(1 \times n_L)$
- $\frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} = \mathrm{Diag}(h'_{(L)}(\mathbf{s}^{(L)}))$ $(n_L \times n_L)$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- $\frac{d\delta}{d\mathbf{v}^{(L)}} = \left[\frac{d\delta}{dv_1^{(L)}}, \ldots, \frac{d\delta}{dv_{n_L}^{(L)}} \right]$ $(1 \times n_L)$
- $\frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} = \mathrm{Diag}(h'_{(L)}(\mathbf{s}^{(L)}))$ $(n_L \times n_L)$

$$E^{(L)} = \underbrace{\frac{d\delta}{d\mathbf{v}^{(L)}} \odot h'_{(L)}(\mathbf{s}^{(L)\,t})}_{\text{pointwise } *}$$

$$E^{(L)} = \frac{d\delta}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}$$

- $\frac{d\delta}{d\mathbf{v}^{(L)}} = \left[\frac{d\delta}{dv_1^{(L)}}, \ldots, \frac{d\delta}{dv_{n_L}^{(L)}} \right]$ $(1 \times n_L)$
- $\frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} = \mathrm{Diag}(h'_{(L)}(\mathbf{s}^{(L)}))$ $(n_L \times n_L)$

$$E^{(L)} = \underbrace{\frac{d\delta}{d\mathbf{v}^{(L)}} \odot h'_{(L)}(\mathbf{s}^{(L)\,t})}_{\text{pointwise } *} \quad (1 \times n_L)$$

$$E^{(l-1)} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\mathbf{v}^{(l-1)}} \frac{d\mathbf{v}^{(l-1)}}{d\mathbf{s}^{(l-1)}}$$

$$E^{(l-1)} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\mathbf{v}^{(l-1)}} \frac{d\mathbf{v}^{(l-1)}}{d\mathbf{s}^{(l-1)}}$$

$$E^{(l-1)} = E^{(l)} W^{(l)} \mathrm{Diag}(h'_{(l-1)}(\mathbf{s}^{(l-1)}))$$

where $W^{(l)}$ is the weights matrix without the first column (bias term)

$$E^{(l-1)} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\mathbf{v}^{(l-1)}} \frac{d\mathbf{v}^{(l-1)}}{d\mathbf{s}^{(l-1)}}$$

$$E^{(l-1)} = E^{(l)} W^{(l)} \mathrm{Diag}(h'_{(l-1)}(\mathbf{s}^{(l-1)}))$$

where $W^{(l)}$ is the weights matrix without the first column (bias term)
Finally

$$E^{(l-1)} = (E^{(l)} W^{(l)}) \odot h'_{(l-1)}(\mathbf{s}^{(l-1)})^t \qquad (1 \times n_{l-1})$$

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}}$$

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}}$$

Set

$$\tilde{v}_j^{(l-1)} = \begin{cases} v_j^{(l-1)} & \text{if} \quad j > 0 \\ 1 & \text{if} \quad j = 0 \end{cases}$$

$$\frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}} = \tilde{v}_j^{(l-1)} e_i(n_l + 1) = \left.\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tilde{v}_j^{(l-1)} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right\} \begin{array}{l} \\ \\ \\ \leftarrow i+1 \\ \\ \\ \\ \end{array} \Bigg\} n_l + 1$$

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}}$$

Set

$$\tilde{v}_j^{(l-1)} = \begin{cases} v_j^{(l-1)} & \text{if} \quad j > 0 \\ 1 & \text{if} \quad j = 0 \end{cases}$$

$$\frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}} = \tilde{v}_j^{(l-1)} e_i(n_l + 1) = \left.\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tilde{v}_j^{(l-1)} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right\} \begin{array}{l} \\ \\ \\ \leftarrow i+1 \\ \\ \\ \\ \end{array} \Bigg\} n_l + 1$$

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E_i^{(l)} \tilde{v}_j^{(l-1)}$$

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E^{(l)} \frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}}$$

Set

$$\tilde{v}_j^{(l-1)} = \begin{cases} v_j^{(l-1)} & \text{if} \quad j > 0 \\ 1 & \text{if} \quad j = 0 \end{cases}$$

$$\frac{d\mathbf{s}^{(l)}}{d\omega_{i,j}^{(l)}} = \tilde{v}_j^{(l-1)} e_i(n_l + 1) = \left.\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tilde{v}_j^{(l-1)} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right\} \begin{array}{l} \\ \\ \\ \leftarrow i+1 \\ \\ \\ \\ \end{array} \Bigg\} n_l + 1$$

$$\frac{d\delta}{d\omega_{i,j}^{(l)}} = E_i^{(l)} \tilde{v}_j^{(l-1)}$$

Matrix of derivatives $$\left[\frac{d\delta}{d\,\tilde{W}^{(l)}}\right] = E^{(l)\,t} \tilde{\mathbf{v}}^{(l-1)^t} \quad (n_l \times n_{l-1})$$

Back-propagation for multiple inputs

$$\frac{d\mathcal{L}}{d\omega_{i,j}^{(l)}}(\mathbf{W}) = \sum_{u=1}^{N} \frac{d\delta}{d\omega_{i,j}^{(l)}}(\mathbf{y}_u, f_{\mathbf{W}}(\mathbf{x}_u))$$

Back-propagation for multiple inputs

$$\frac{d\mathcal{L}}{d\omega_{i,j}^{(l)}}(\mathbf{W}) = \sum_{u=1}^{N} \frac{d\delta}{d\omega_{i,j}^{(l)}}(\mathbf{y}_u, f_{\mathbf{W}}(\mathbf{x}_u))$$

We just have to stack the derivatives of the individual loss functions:

$$\left[\frac{d\delta_u}{d\mathbf{v}^{(L)}}\right]_{u=1\ldots,N}$$

is the $N \times n_L$ matrix with line u defined by

$$\frac{d\delta}{d\mathbf{v}^{(L)}}(\mathbf{y}_u, f_{\mathbf{W}}(\mathbf{x}_u))$$

Back-propagation

Summary

- Initialization: Compute

$$\mathbf{E}^{(L)} = \left[\frac{d\delta_u}{d\mathbf{v}^{(L)}}\right]_{u=1\ldots,N} \odot h'_{(L)}\left(S^{(L)}\right)^t$$

- For l from L to 1
  - Compute

$$\left[\frac{d\mathcal{L}}{d\,\tilde{W}^{(l)}}(\mathbf{W})\right] = \left(\mathbf{E}^{(l)}\right)^t \left(\tilde{V}^{(l)}\right)^t$$

  - If $l > 1$, compute

$$\mathbf{E}^{(l-1)} = \left(\mathbf{E}^{(l)}\, W^{(l)}\right) \odot h'_{(l-1)}\left(S^{(l-1)}\right)^t$$

Back-propagation

Summary

- Initialization: Compute

$$\mathbf{E}^{(L)} = \left[\frac{d\delta_u}{d\mathbf{v}^{(L)}}\right]_{u=1\ldots,N} \odot h'_{(L)}\left(S^{(L)}\right)^t$$

- For l from L to 1
  - Compute

$$\left[\frac{d\mathcal{L}}{d\,\tilde{W}^{(l)}}(\mathbf{W})\right] = (\mathbf{E}^{(l)})^t\left(\tilde{V}^{(l)}\right)^t$$

  - If $l > 1$, compute

$$\mathbf{E}^{(l-1)} = (\mathbf{E}^{(l)}\,W^{(l)}) \odot h'_{(l-1)}\,(S^{(l-1)})^t$$

The quantities $\tilde{V}^{(l)}$ and $S^{(l)}$ are computed and stored during the forward propagation (forward pass).

Stochastic gradient and variants

Groupe de lecture - Mike Pereira

- The evaluation of the loss function gradient can be costly when the training set is large
- Instead of computing the gradient on the entire training set, it will be computed on mini-batches of moderate size n_b (e.g $n_b = 20$ but can be 1).

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} - \eta \nabla \mathcal{L}_{n_b} \left(\mathbf{W}_{(t)} \right)$$

- The number of epochs is the number of times the entire data set is visited
- At the beginning of each epoch, the data-set is randomly shuffled
- The cost-function and its gradient are divided by n_b (to make the choice of η independent of n_b)

Stochastic gradient with momentum

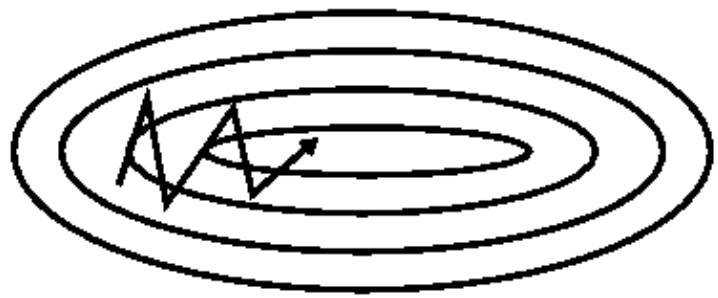
- In order to stabilize the convergence, one can introduce momentum

$$m_{(0)} = \mathbf{0}, \qquad \beta \in [0, 1) \qquad \text{e.g } \beta = 0.9$$

$$m_{(t+1)} = \beta m_{(t)} + (1 - \beta)\nabla \mathcal{L}_{n_b}(\mathbf{W}_{(t)})$$

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} - \eta m_{(t+1)}$$

Without momentum.



With momentum

Source: Genevieve B. Orr

Adaptive Moment Estimation (ADAM)

Kingma and Ba (2014)

Initialisation

$$m_{(0)} = v_{(0)} = \mathbf{0}, \qquad \beta_1, \beta_2 \in [0, 1) \qquad \text{e.g } \beta_1 = 0.9, \beta_2 = 0.999$$

For $t \geq 1$

$$m_{(t+1)} = \beta_1 m_{(t)} + (1 - \beta_1) \nabla \mathcal{L}_{n_b}(\mathbf{W}_{(t)})$$

$$v_{(t+1)} = \beta_2 v_{(t)} + (1 - \beta_2) \nabla \mathcal{L}^2_{n_b}(\mathbf{W}_{(t)})$$

$$\hat{m}_{(t+1)} = \frac{m_{(t+1)}}{1 - \beta_1^t}$$

$$\hat{v}_{(t+1)} = \frac{v_{(t+1)}}{1 - \beta_2^t}$$

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} - \eta \frac{\hat{m}_{(t+1)}}{\sqrt{\hat{v}_{(t+1)}} + \varepsilon}$$

References

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks 4*(2), 251–257.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Shalev-Shwartz, S. and S. Ben-David (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.