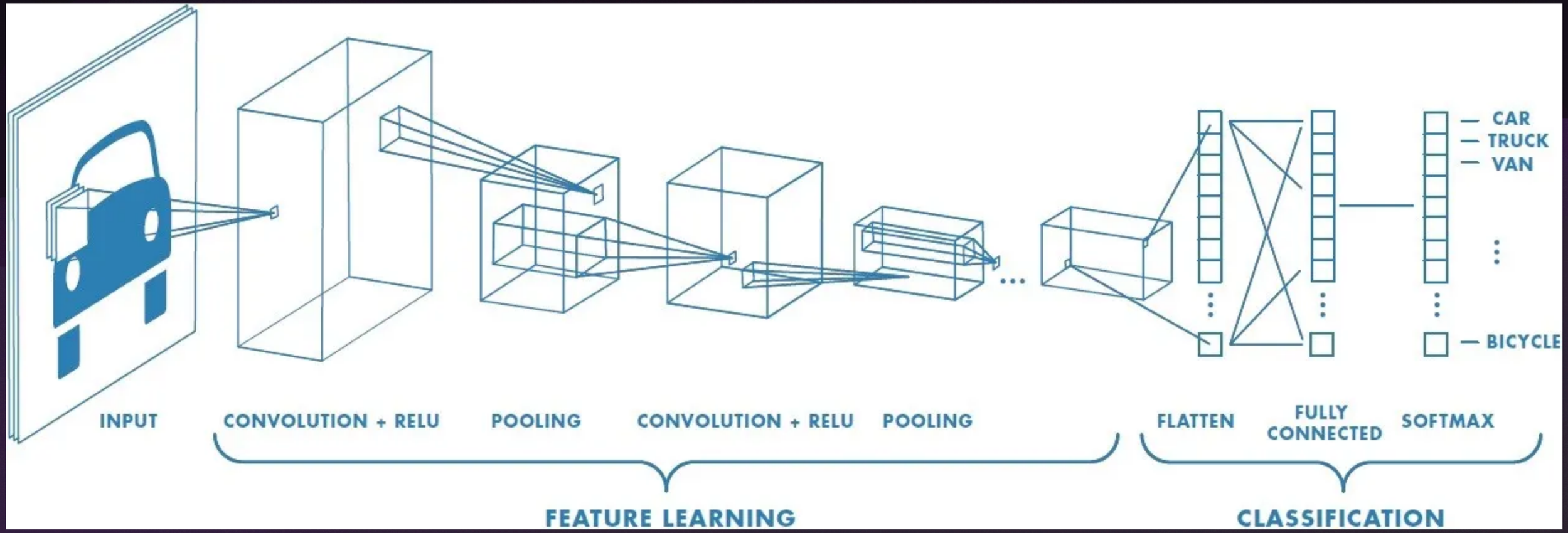
Convolutional Neural Networks

AI Systems Implementation

George Langroudi
Based on slides by Gareth Howells

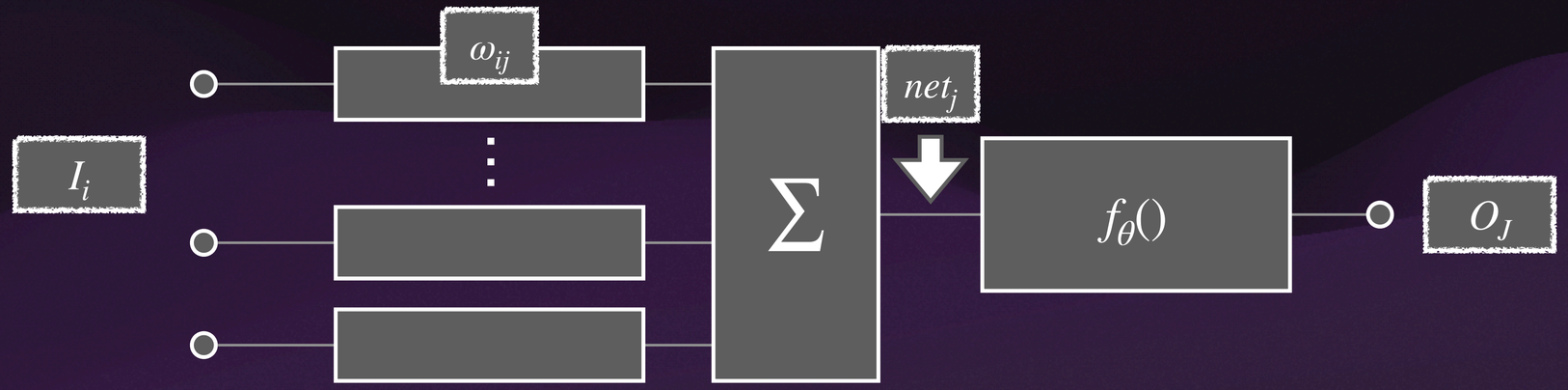

CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

Overview

- Examining Multi-layer networks
  - Multiple inputs
  - Limitations of two layer networks
  - Deep Learning
- Defining a network with Keras

Neurons

The McCulloch and Pitts Neuron (1943)



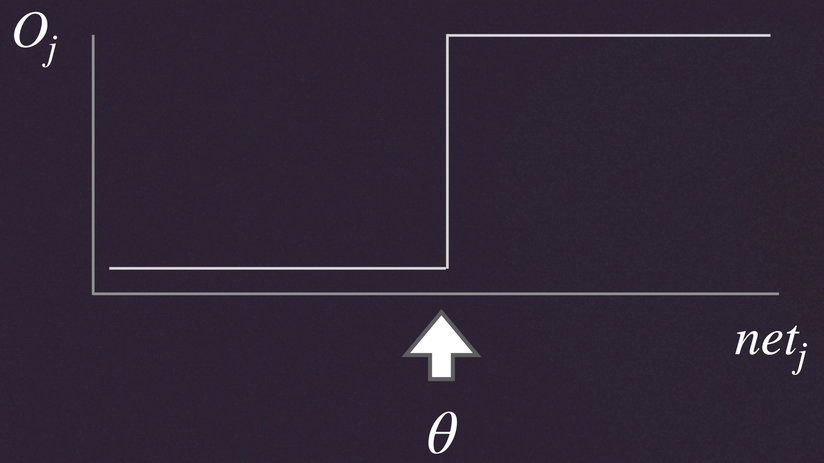
$$O_j = f_\theta(net_j)$$

$$net_j = \sum_i \omega_{ij} i_i$$

Where ω_{ij} is the weight, θ is the threshold, and:

$f_\theta(x) = 1$ if $x \geq \theta$

$f_\theta(x) = 0$ if $x < \theta$



Single Input Neuron

NB: i_1 not Boolean in this example

$$net = i_1 \times w_1$$

$$o = 1, \text{ if } net \geq \theta$$

$$o = 0, \text{ if } net < \theta$$

For threshold θ, may rewrite as:

$$net = i_1 \times w_1 - \theta$$

$$o = 1, \text{ if } net \geq 0$$

$$o = 0, \text{ if } net < 0$$

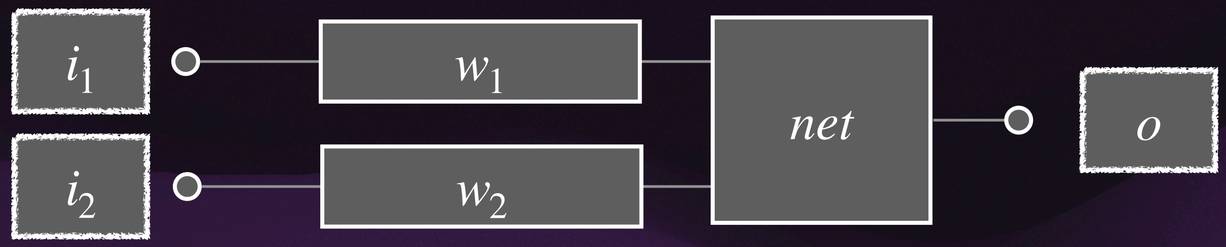
Limitations

- Consider again a neutron with one input. It take input x, multiplies it by a weight m, and tests if it exceeds threshold c.

$$z = mx - c$$

- It outputs 1 if $z > 0$, otherwise 0
- It is a one-dimensional object that divides a 2 dimensional space
- Although a neuron can divide a space, it can *only divide any space by a straight line*

Two Input Neuron



$$net = i_1 \times w_1 + i_2 \times w_2$$

$$o = 1, \text{ if } net \geq \theta$$

$$o = 0, \text{ if } net < \theta$$

For threshold θ, may rewrite as:

$$net = i_1 \times w_1 + i_2 \times w_2 - \theta$$

$$o = 1, \text{ if } net \geq 0$$

$$o = 0, \text{ if } net < 0$$

Analysis

- The values of *net* represent different points on a plane e.g.

$$net = i_1 \times w_1 + i_2 \times w_2 - \theta$$

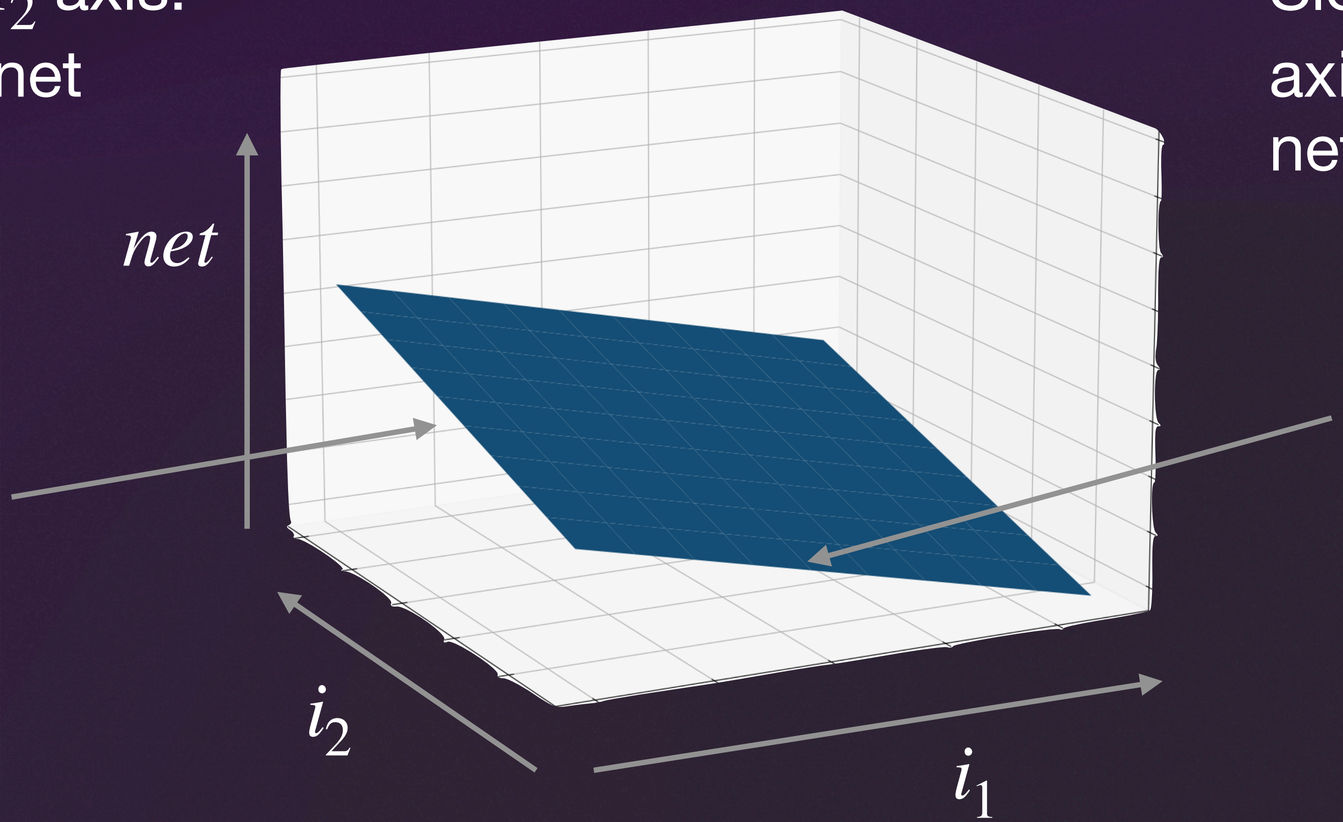
$$w_1 = -1$$

$$w_2 = 2$$

$$\theta = -1$$

Slope up along I_2 axis: as i_2 increases, net increases

i_1	i_2	net
0	0	1
0	1	3
0	2	5
0	3	7



Slope down along I_1 axis: as i_1 increases, net decreases

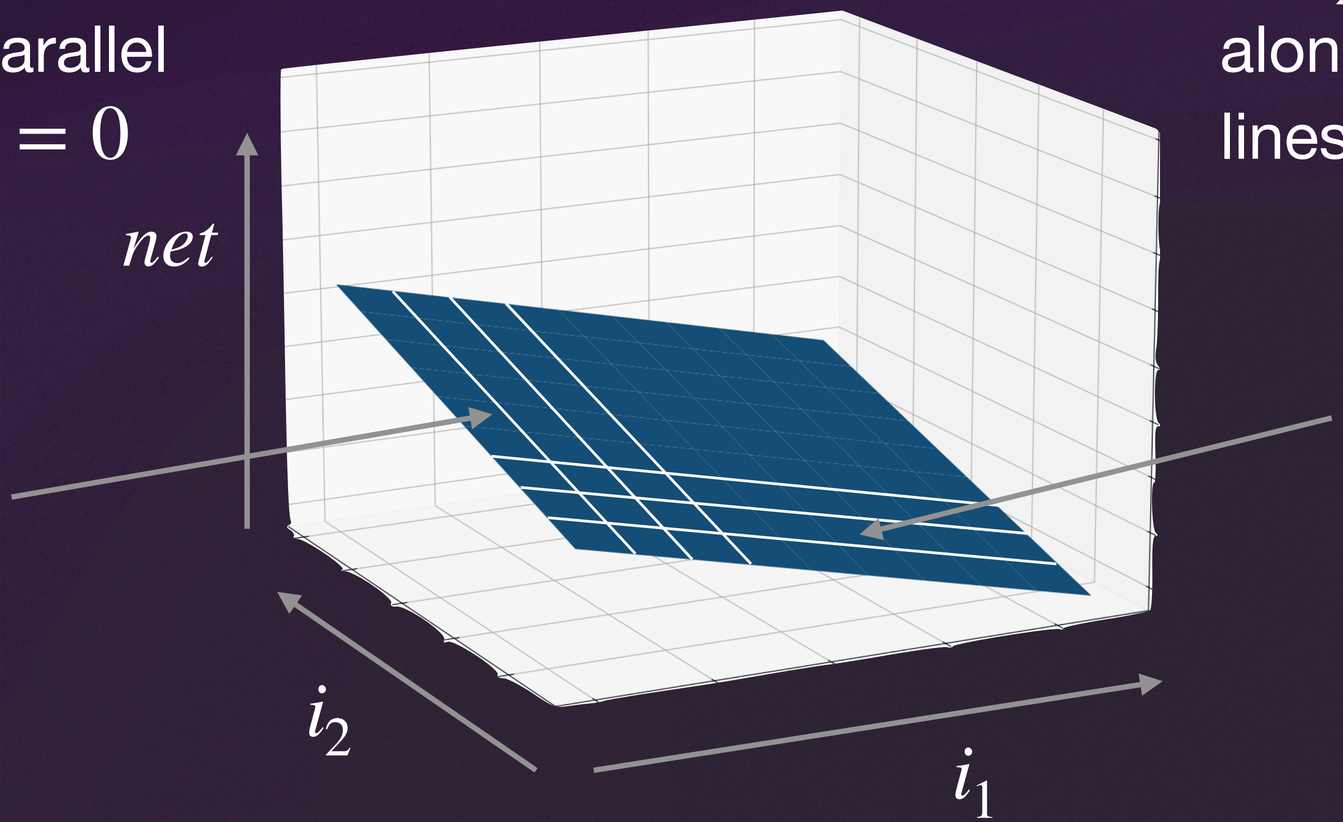
i_1	i_2	net
0	0	1
1	0	0
2	0	-1
3	0	-2

Analysis

- A plane is formed by extending the lines along the axes into two dimensions

As I_1 varies: Variation along I_2 forms parallel lines to line at $i_1 = 0$

i_1	i_2	net
1	0	0
1	1	2
1	2	4
1	3	6

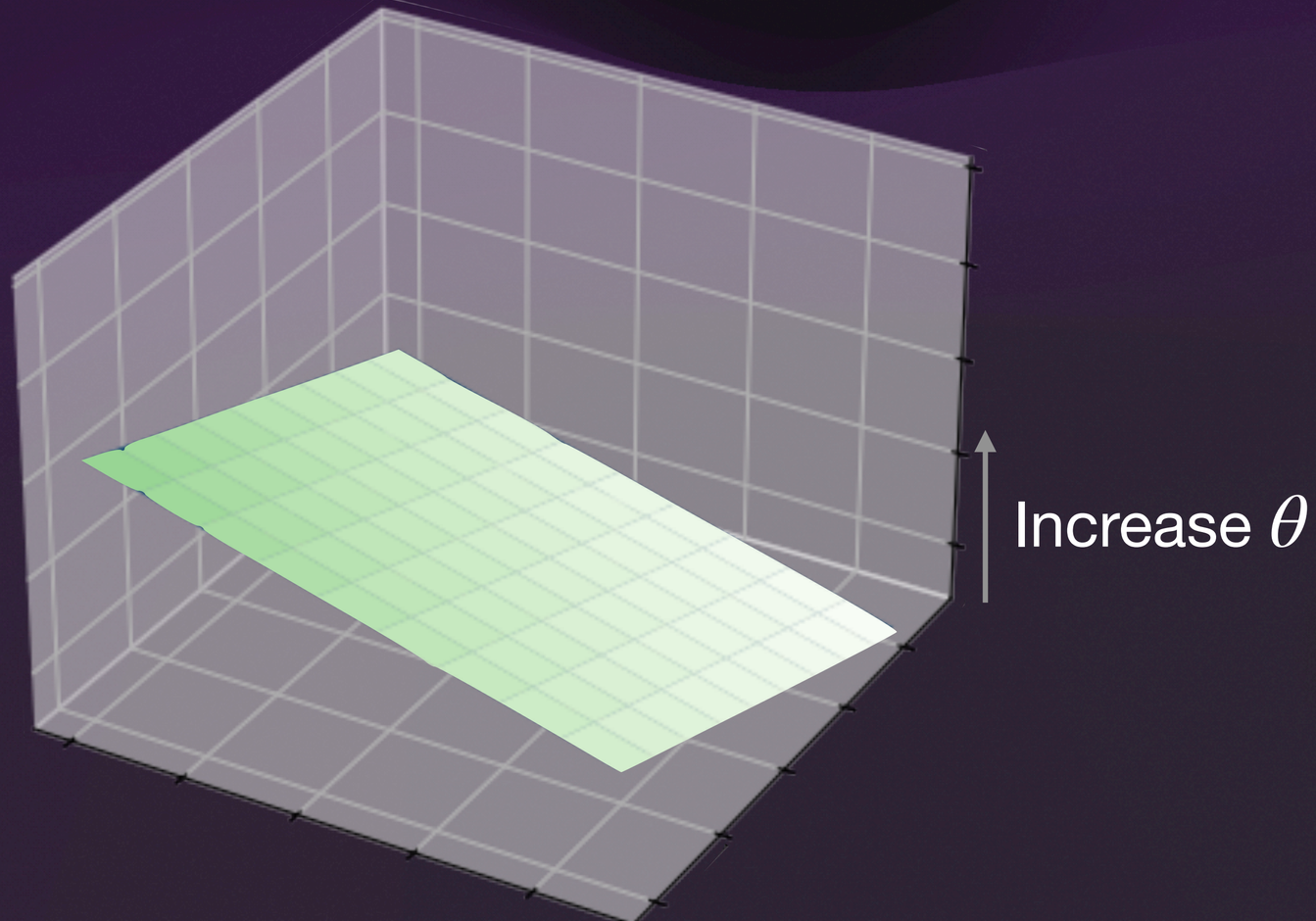


As I_2 varies: Variation along I_1 axis forms parallel lines to line at $i_2 = 0$

i_1	i_2	net
0	1	3
1	1	2
2	1	1
3	1	0

Analysis

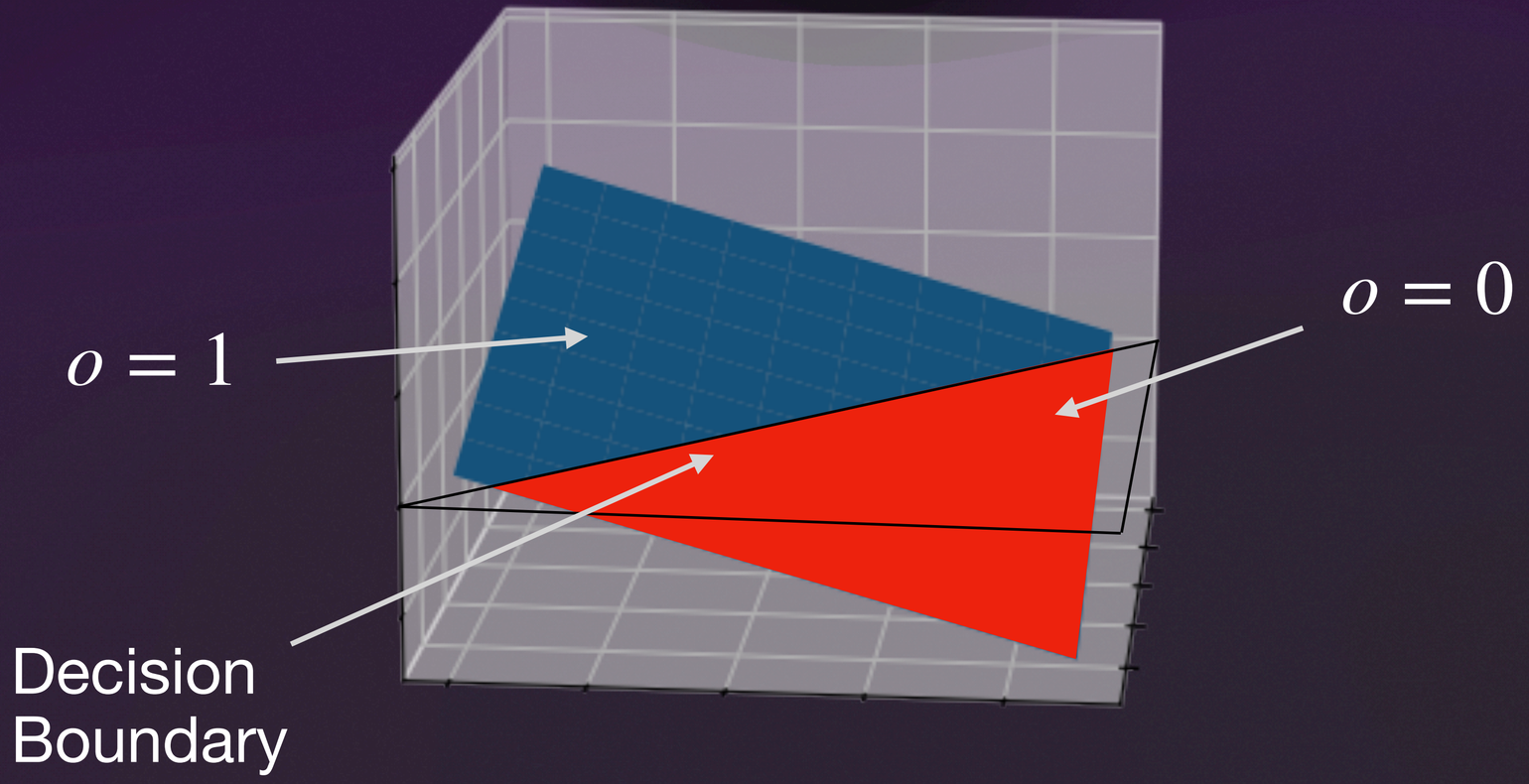
- Varying W_1 and W_2 varies gradient in each axis independently
- Varying θ moves entire plane along net axis



Decision Boundary

- Decision Boundary: Where plane formed by net equation meets plane formed by i_1 and i_2 axes

- Decision boundary is a straight line…
  - A 1 dimensional object, splitting a 2 dimensional space



Analysis

- Varying weights w_1 and w_2 varies the decision boundary
  - Always a straight line…
- If $w_1 = w_2 = 0$, plane will be parallel to i_1, i_2 axis and no decision boundary is formed
  - $net = 0$ in all cases and output will be constant for all values of i_1 and i_2

Limitations with two inputs

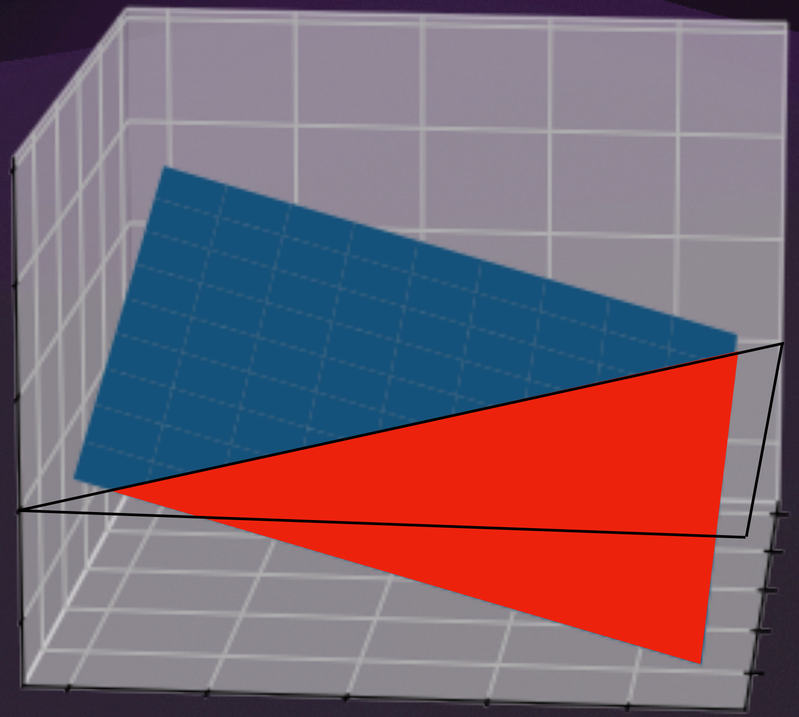
- Consider again the 2-input neuron. It takes 2 inputs, x and y, multiples them by weights m and n, adds the result and tests if it's above a threshold.

$$z = mx + ny - c$$

- Outputs 1 if $z > 0$ and 0 otherwise
- This is the equation of a *plane*
  - A 2-dimensional object that divides a 3-dimensional space

Decision Boundary

- A hyperplane is a higher dimensional equivalent of a plane
- With *n* inputs (dimensions), a neuron may be modelled as an *n*-dimensional hyperplane
- The neuron slides it's decision space in two in a regular fashion. This is a *linear* decision function

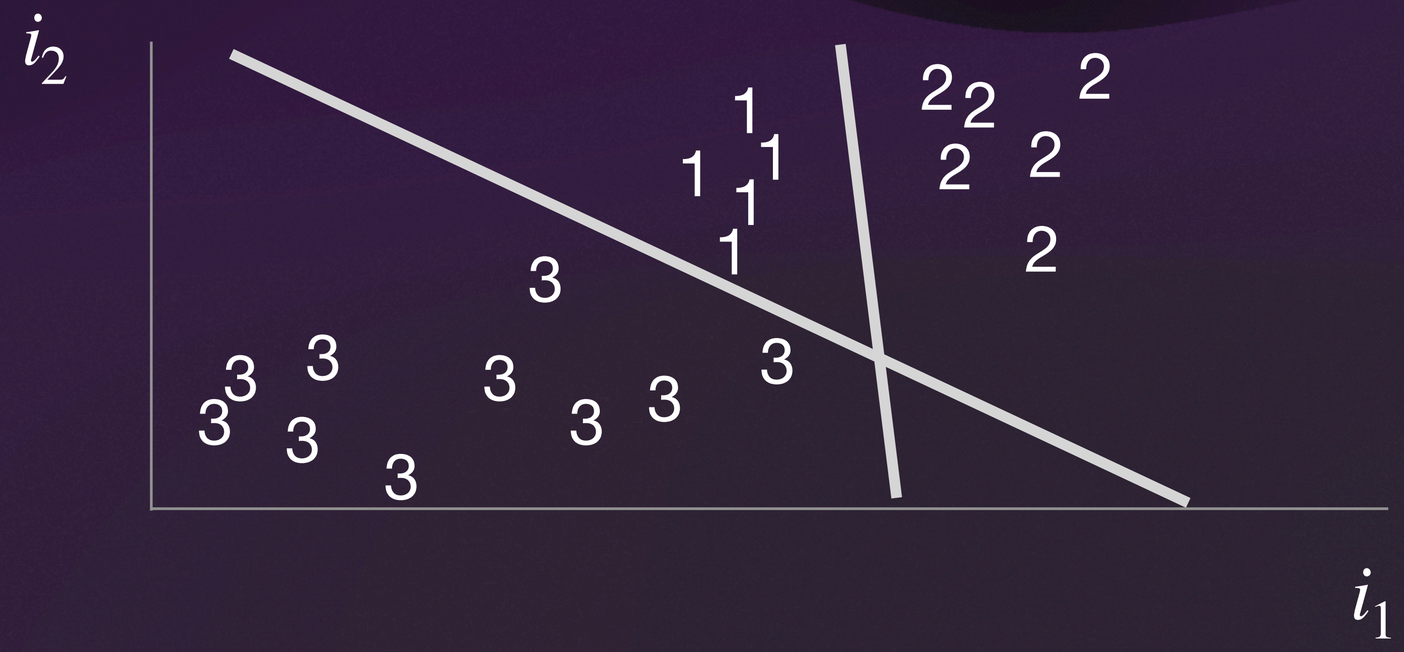


General Case (Many inputs)

- For *n* inputs:
  - Equation defining net is an n-dimensional hyperplane
    - 1 input: Straight line (1 dimension)
    - 2 inputs: Plane (2 dimensions)
    - 3 inputs: 3D hyperplane (3 dimensions)
  - Decision boundary is an n-1 dimensional hyperplane
    - 1 input: point (0 dimensions)
    - 2 inputs: Straight line (1 dimension)
    - 3 inputs: plane (2 dimensions)
- Boundary formed where net hyperplane intersects hyperplane formed by $i_1 \ldots i_n$ axes

Input spaces

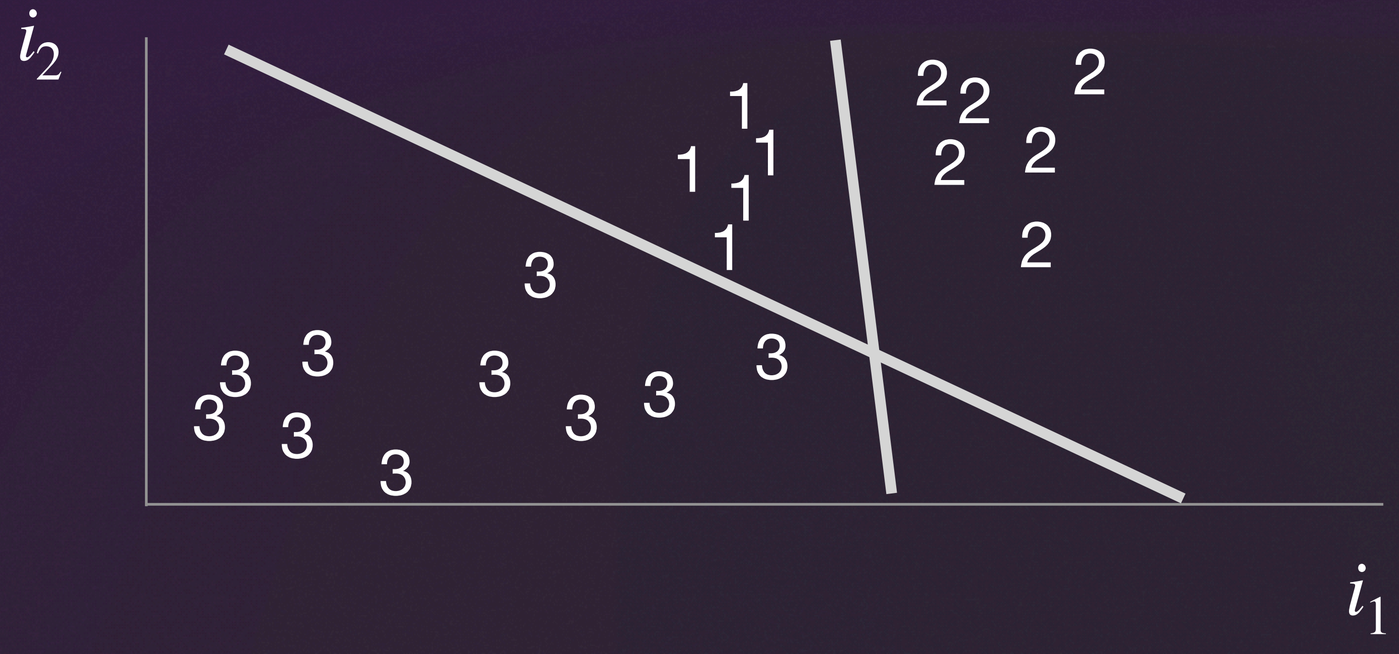
- Here’s an example with 3 classes



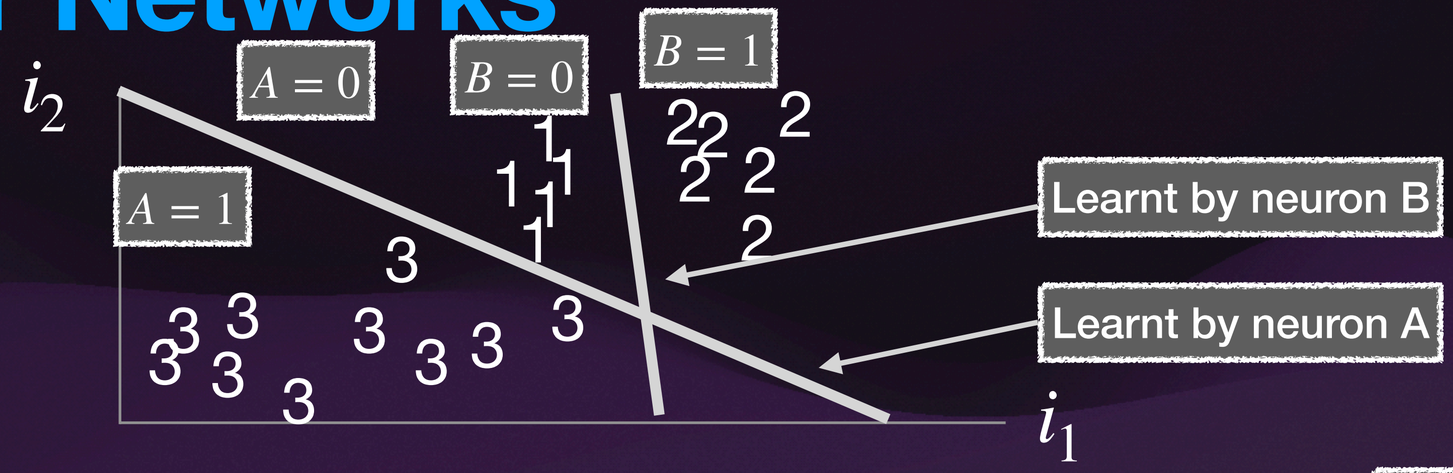
- The “2” class can be separated with a single plane, as can “3”, but “1” cannot

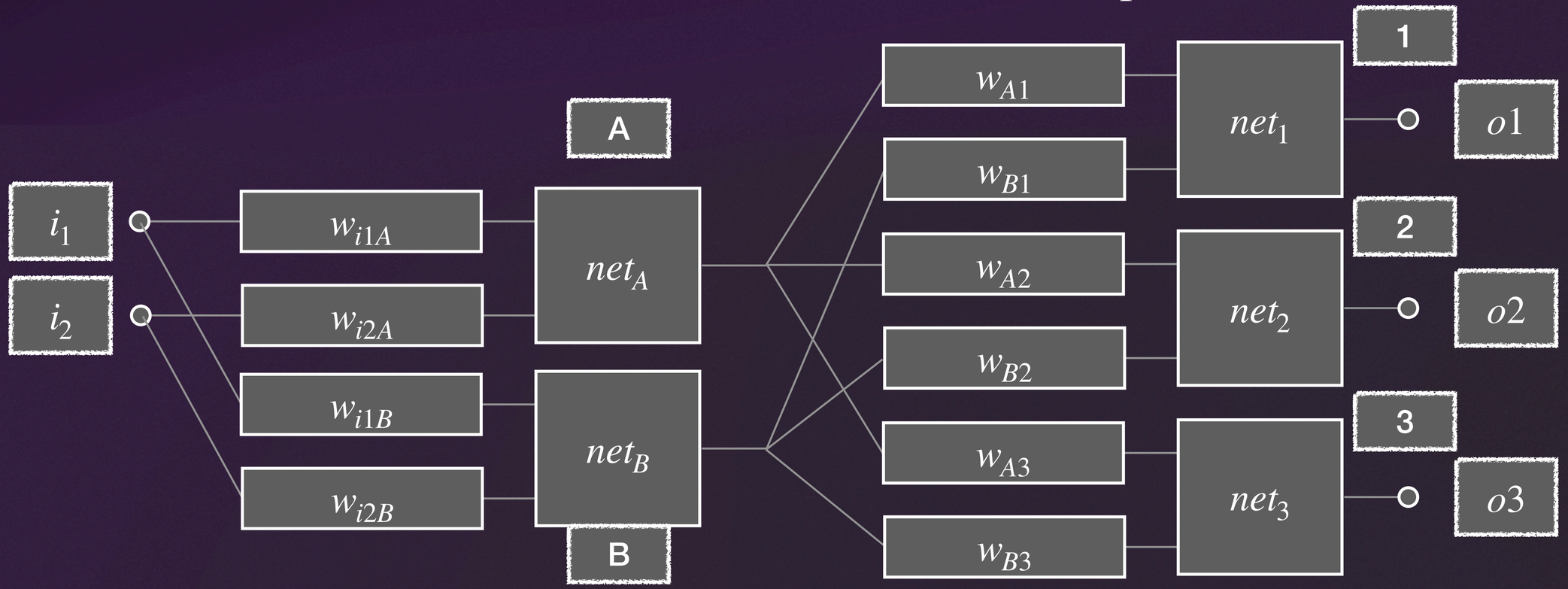
Let’s think for a minute…

- If we can separate 2’s from everything, and 3’s from everything, we’ve divided the space into 2 regions!
- Combine both neurons, we could tell which class was recognised!



Two-Layer Networks

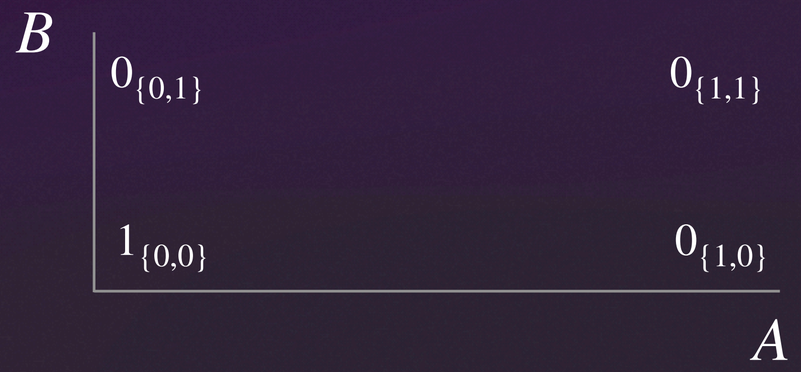




Neuron 1

- $O_1 = 1$ when class 1 recognised, otherwise 0

A	B	O_1
0	0	1
0	1	0
1	0	0
1	1	0



- This requires a linear decision function
  - i.e. Values of W_{A1} and W_{B1} exist to represent this function

Neurons 2 and 3

- $O_2 = 1$ when class 2 recognised

A	B	O_1
0	0	0
0	1	1
1	0	0
1	1	1

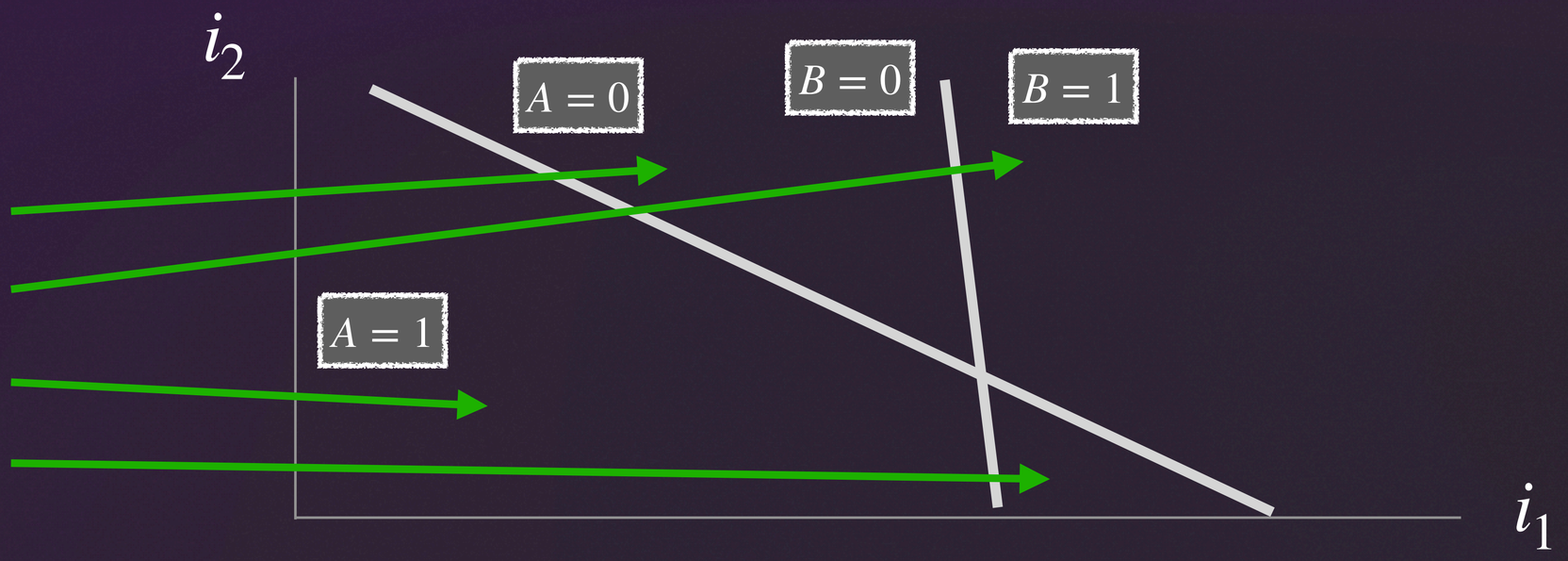
- $O_3 = 1$ when class 3 recognised

A	B	O_1
0	0	0
0	1	0
1	0	1
1	1	1

A and B Create Regions

- Neurons 1, 2, and 3 have simple Boolean inputs and learn simple Boolean functions
- Each combination of A and B outputs corresponds to one of the 4 reasons in the decision space

A	B
0	0
0	1
1	0
1	1



Multiple layers

- The addition of a neuron taking the outputs of other neurons as inputs can consider combinations of the *hyperplanes* formed by such neurons
  - The first layer places hyperplanes into the input space
  - Subsequent layers combine the outputs of the first layer neurons to create regions in the input space
  - *Intuitively: required output could occur “above” one “line” and “to the left” of another.*

Deep Learning

- Theoretically, no more than three layers needed
- However, this means rather than having general classes and subclasses, there are just very many classes
  - “Difficult” to learn — Exponential increase in neurons
- Deep networks allow more general classes which can subsequently be specialised.

Keras Functional API

- Define model

```
inputs = keras.Input(shape=<shape definition>)

first = layers.<layer definition>(inputs)
second = layers.<layer definition>(first)
outputs = layer.<layer definition>(second)

model = keras.Model(inputs=inputs, outputs=outputs)
```

Keras Functional API

- Compile model

```
model.compile(optimizer=<choose optimiser>,
              loss=<choose loss function>,
              metrics=<choose metrics>)
```

- Train model

```
model.fit(<train data>, <labels>, <etc>)
```

- Test model

```
model.predict(<test data>)
model.evaluate(<test data>, <labels>)
```