CSC321
Introduction to Neural Networks and Machine Learning

Lecture 2: Two simple learning algorithms

Geoffrey Hinton

Supervised Learning

- Each training case consists of an input vector x and a desired output y (there may be multiple desired outputs but we will ignore that for now)
  - Regression: Desired output is a real number
  - Classification: Desired output is a class label (1 or 0 is the simplest case).
- We start by choosing a model-class
  - A model-class is a way of using some numerical parameters, W, to map each input vector, x, into a predicted output $\hat{y}$
- Learning usually means adjusting the parameters to reduce the discrepancy between the desired output on each training case and the actual output produced by the model.

Linear neurons

- The neuron has a real-valued output which is a weighted sum of its inputs

weight vector

$$\hat{y} = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

Neuron's estimate of the desired output

input vector

- The aim of learning is to minimize the discrepancy between the desired output and the actual output
  – How de we measure the discrepancies?
  – Do we update the weights after every training case?
  – Why don't we solve it analytically?

A motivating example

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and beer.
  - You get several portions of each
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portion.
- Each meal price gives a linear constraint on the prices of the portions:

$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{beer} w_{beer}$$

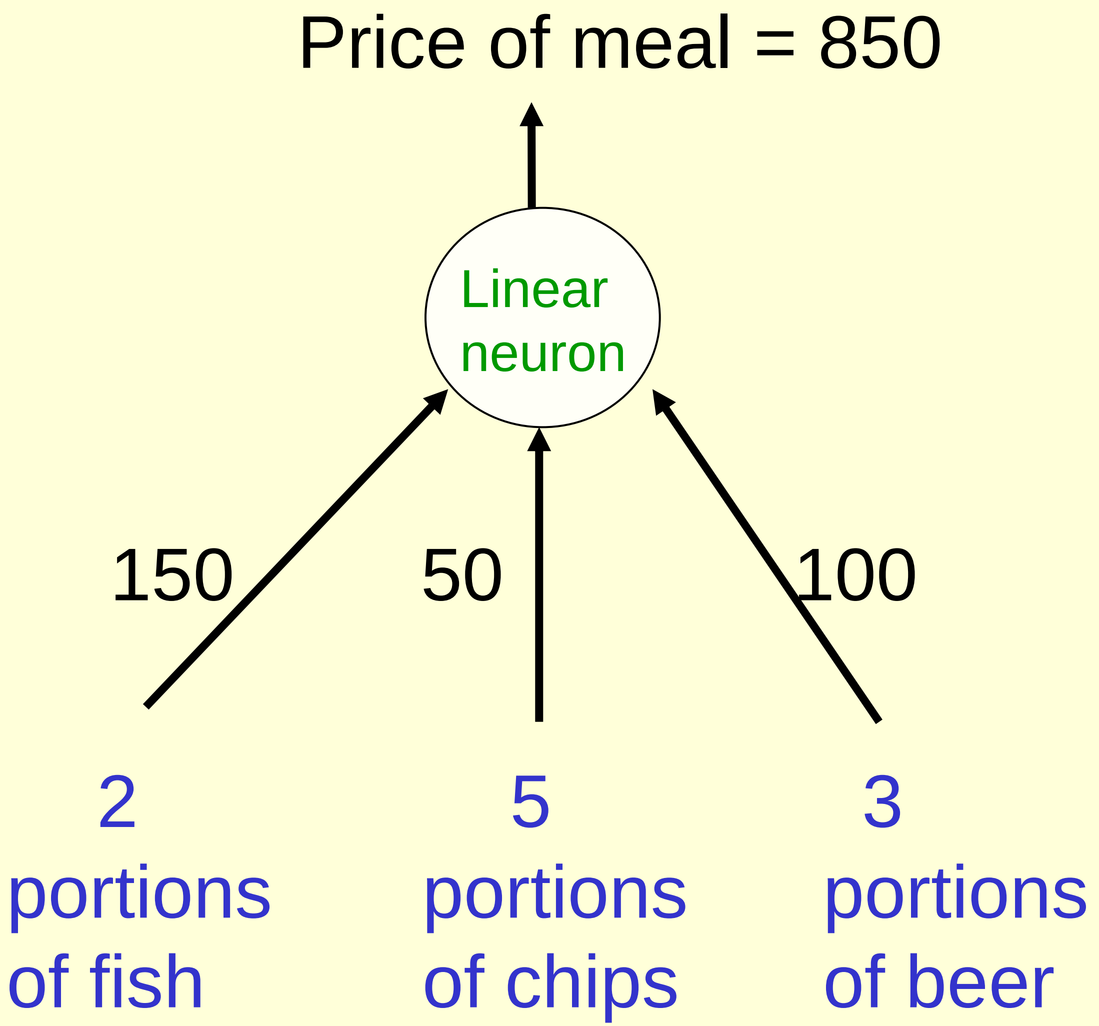
Two ways to solve the equations

- The obvious approach is just to solve a set of simultaneous linear equations, one per meal.
- But we want a method that could be implemented in a neural network.
- The prices of the portions are like the weights in of a linear neuron.

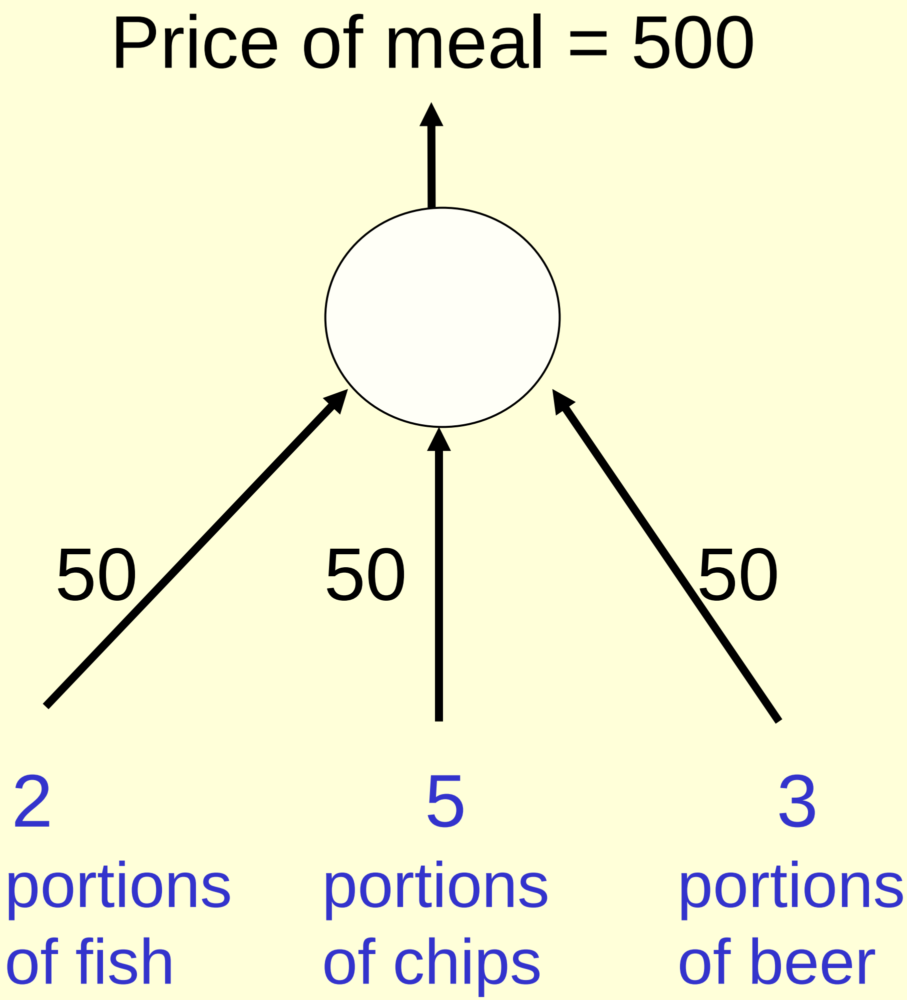
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{beer})$$

- We will start with guesses for the weights and then adjust the guesses to give a better fit to the prices given by the cashier.

The cashier's brain



A model of the cashier's brain with arbitrary initial weights



- Residual error = 350
- The learning rule is:

$$\Delta w_i = \varepsilon \, x_i \, (y - \hat{y})$$

- With a learning rate ε of 1/35, the weight changes are +20, +50, +30
- This gives new weights of 70, 100, 80
- Notice that the weight for chips got worse!

Behaviour of the iterative learning procedure

- Do the updates to the weights always make them get closer to their correct values? No!
- Does the online version of the learning procedure eventually get the right answer? Yes, if the learning rate gradually decreases in the appropriate way.
- How quickly do the weights converge to their correct values? It can be very slow if two input dimensions are highly correlated (e.g. ketchup and chips).
- Can the iterative procedure be generalized to much more complicated, multi-layer, non-linear nets? YES!

Deriving the delta rule

- Define the error as the squared residuals summed over all training cases:

$$E = \sum_n \tfrac{1}{2}(y_n - \hat{y}_n)^2$$

- Now differentiate to get error derivatives for weights

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial \hat{y}_n}{\partial w_i} \frac{\partial E_n}{\partial \hat{y}_n}$$

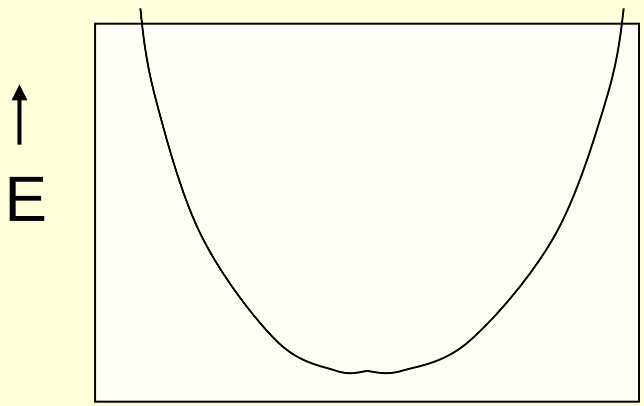
$$= -\sum_n x_{i,n}\,(y_n - \hat{y}_n)$$

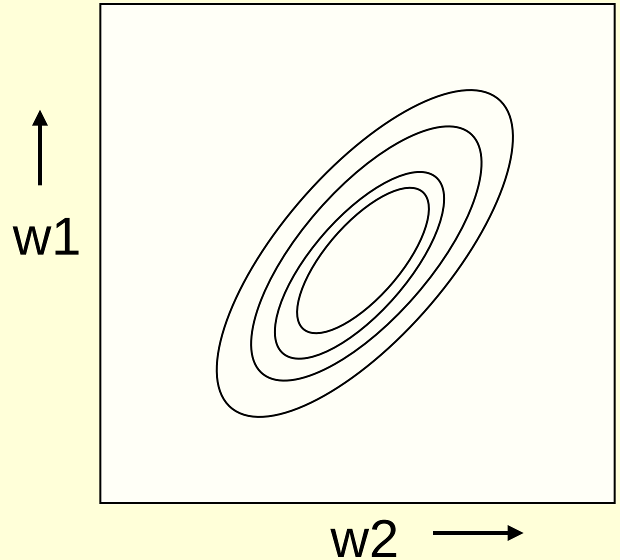
- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$

The error surface

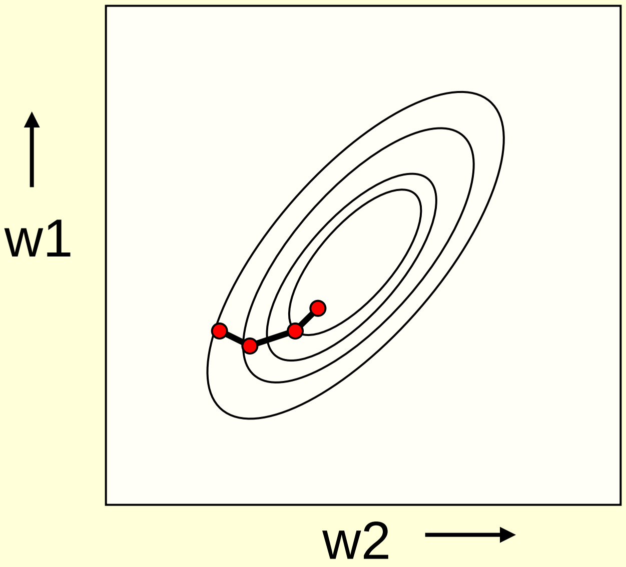
- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - For a linear neuron, it is a quadratic bowl.
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses.

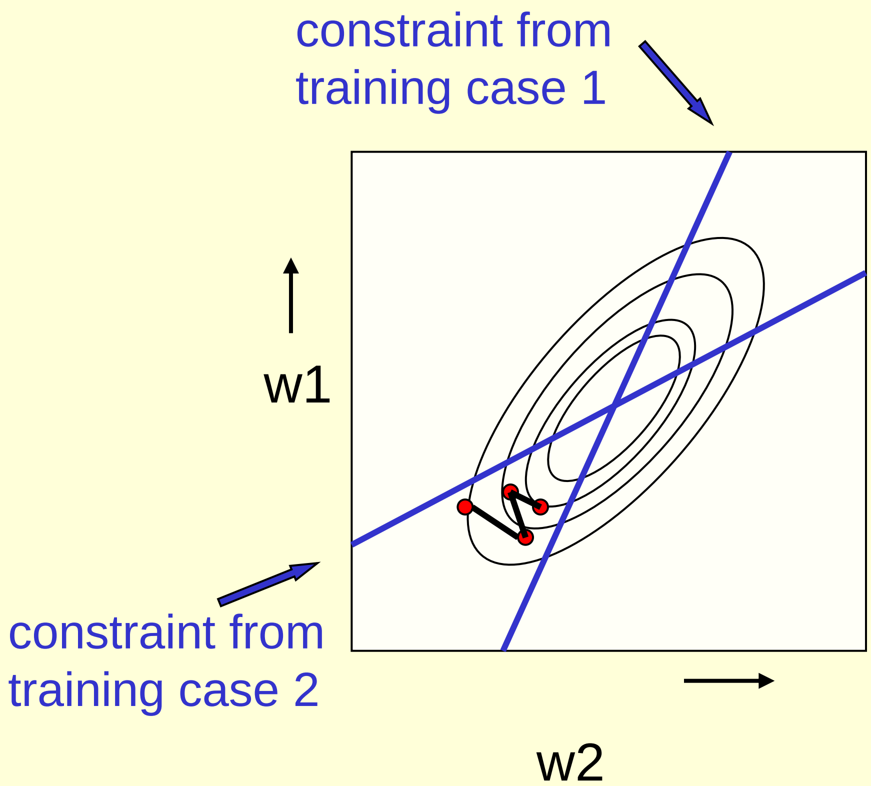




Online versus batch learning

- Batch learning does steepest descent on the error surface
- Online learning zig-zags around the direction of steepest descent

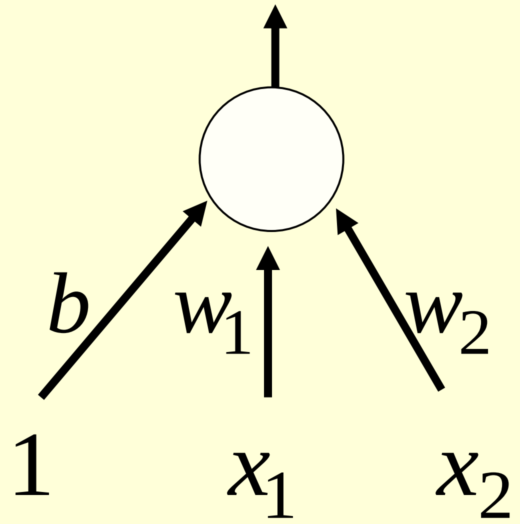




Adding biases

- A linear neuron is a more flexible model if we include a bias.
- We can avoid having to figure out a separate learning rule for the bias by using a trick:
  - A bias is exactly equivalent to a weight on an extra input line that always has an activity of 1.

$$\hat{y} = b + \sum_i x_i w_i$$

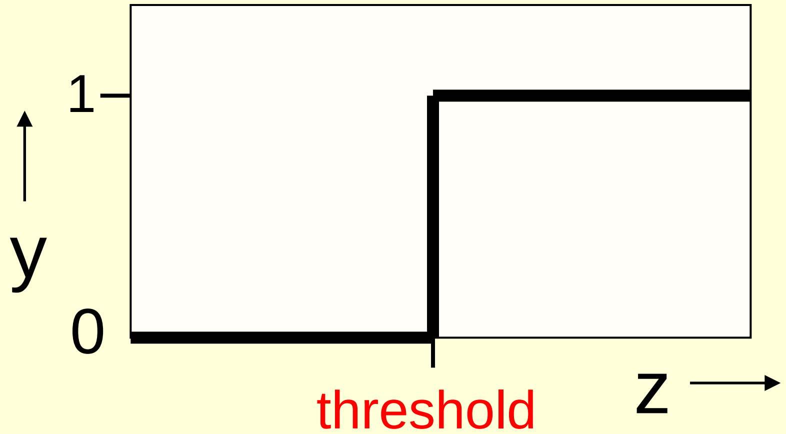


Binary threshold neurons

- McCulloch-Pitts (1943)
  - First compute a weighted sum of the inputs from other neurons
  - Then output a 1 if the weighted sum exceeds the threshold.

$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } & z \geq \theta \\ 0 \text{ otherwise} & \end{cases}$$

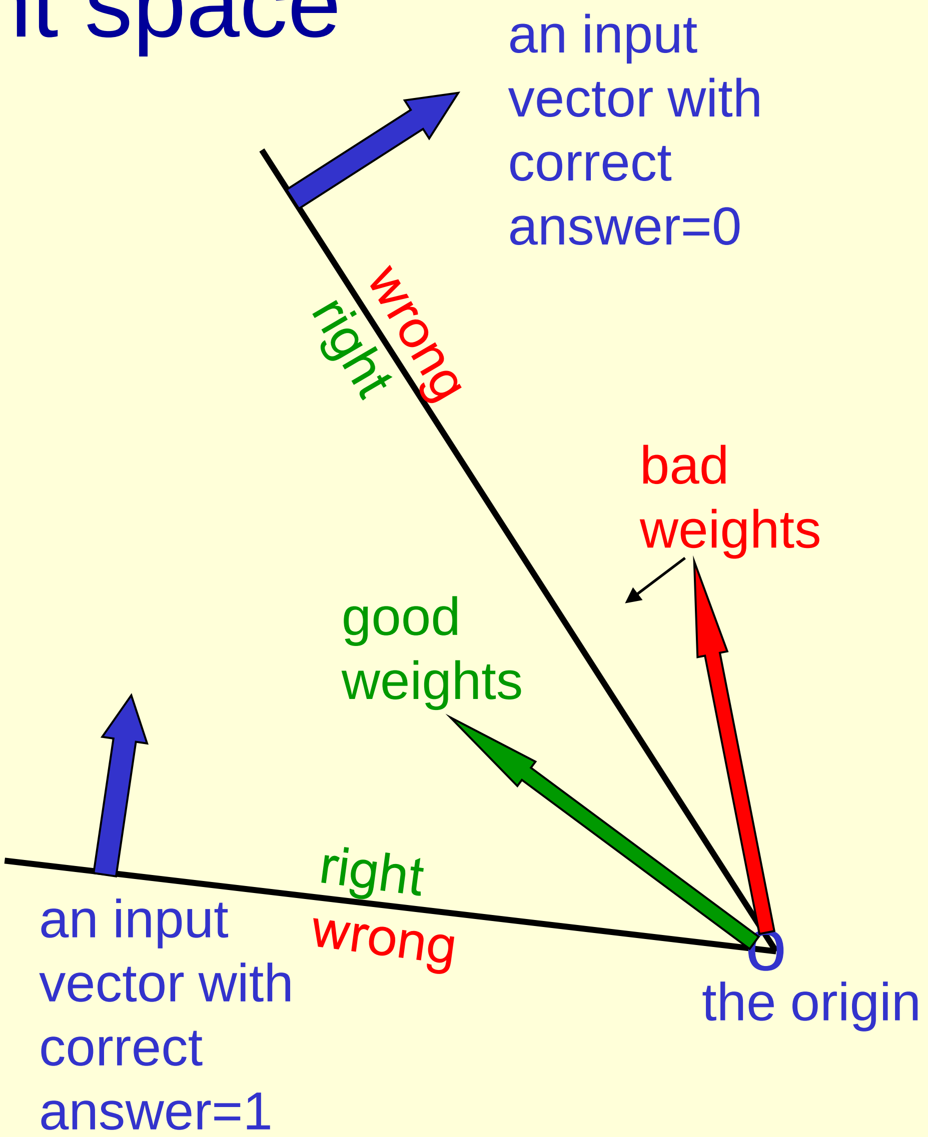


The perceptron convergence procedure: Training binary output neurons as classifiers

- Add an extra component with value 1 to each input vector. The “bias” weight on this component is minus the threshold. Now we can forget the threshold.
- Pick training cases using any policy that ensures that every training case will keep getting picked
  – If the output unit is correct, leave its weights alone.
  – If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  – If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector.
- This is guaranteed to find a suitable set of weights if any such set exists.

Weight space

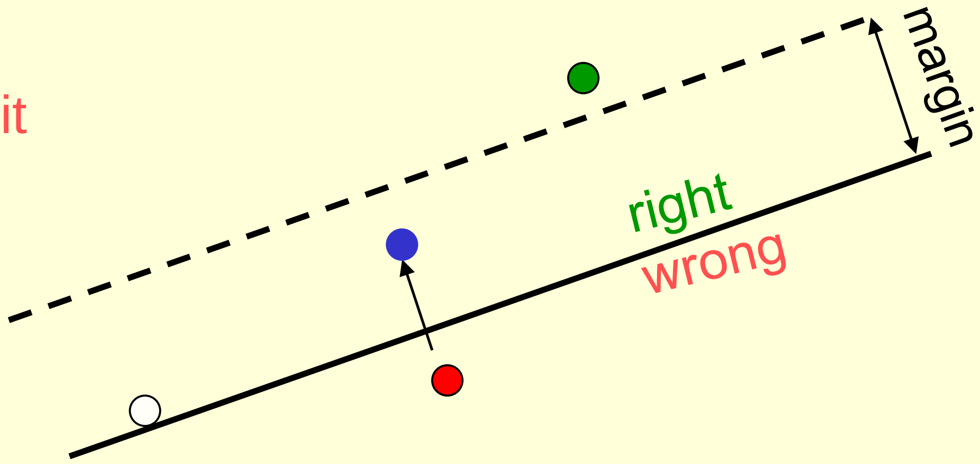
- Imagine a space in which each axis corresponds to a weight.
  - A point in this space is a weight vector.
- Each training case defines a plane.
  - On one side of the plane the output is wrong.
- To get all training cases right we need to find a point on the right side of all the planes.



Why the learning procedure works

- Consider the squared distance between any satisfactory weight vector and the current weight vector.
  – Every time the perceptron makes a mistake, the learning algorithm moves the current weight vector towards all satisfactory weight vectors (unless it crosses the constraint plane).

- So consider “generously satisfactory” weight vectors that lie within the feasible cone by a margin at least as great as the largest update.
  – Every time the perceptron makes a mistake, the squared distance to all of these weight vectors is always decreased by at least the squared length of the smallest update vector.



What binary threshold neurons cannot do

- A binary threshold output unit cannot even tell if two single bit numbers are the same!

  Same: (1,1) → 1; (0,0) → 1

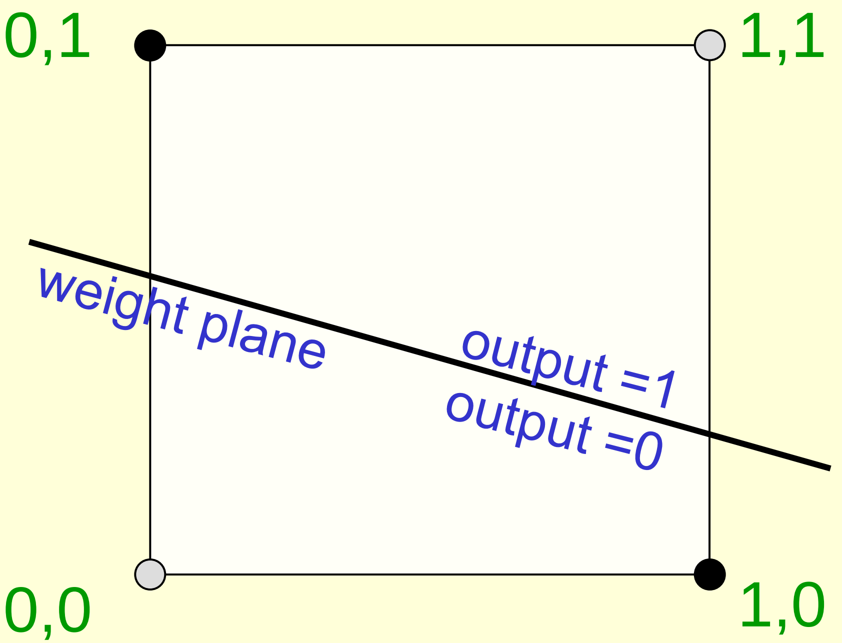
  Different: (1,0) → 0; (0,1) → 0

- The four input-output pairs give four inequalities that are impossible to satisfy:

$$w_1 + w_2 \geq \theta, \quad 0 \geq \theta$$

$$w_1 < \theta, \qquad w_2 < \theta$$

Data Space
(not weight space)



The positive and negative cases cannot be separated by a plane