Perceptron (neural) branch predictor

Daniel A. Jiménez

Slides adapted by: Dr Sparsh Mittal

Conditional Branch Prediction is a Machine Learning Problem

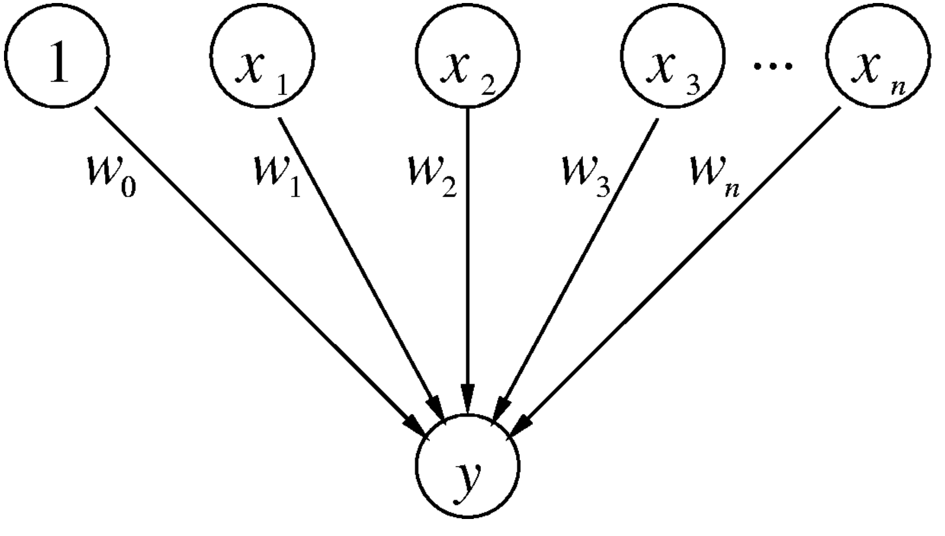
- The machine learns to predict conditional branches
- So why not apply a machine learning algorithm?
- Artificial neural networks
  - Simple model of neural networks in brain cells
  - Learn to recognize and classify patterns
- Idea: Use fast and accurate perceptrons [Rosenblatt `62, Block `62] for dynamic branch prediction

Input and Output of the Perceptron

- The inputs to the perceptron are branch outcome histories
  - Just like in 2-level adaptive branch prediction
  - Can be global or local (per-branch) or both (alloyed)
  - Conceptually, branch outcomes are represented as
    - +1, for taken
    - -1, for not taken
- The output of the perceptron is
  - Non-negative, if the branch is predicted taken
  - Negative, if the branch is predicted not taken
- Ideally, each static branch is allocated its own perceptron

Branch-Predicting Perceptron

- Inputs (x's) are from branch history and are -1 or +1
- $n + 1$ small integer weights (w's) learned by on-line training
- Output (y) is dot product of x's and w's; predict taken if $y \geq 0$
- Training finds correlations between history and outcome



$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

Training Algorithm

$x_{1..n}$ is the n-bit history register, x_0 is 1.
$w_{0..n}$ is the weights vector.
t is the Boolean branch outcome.
θ is the training threshold.

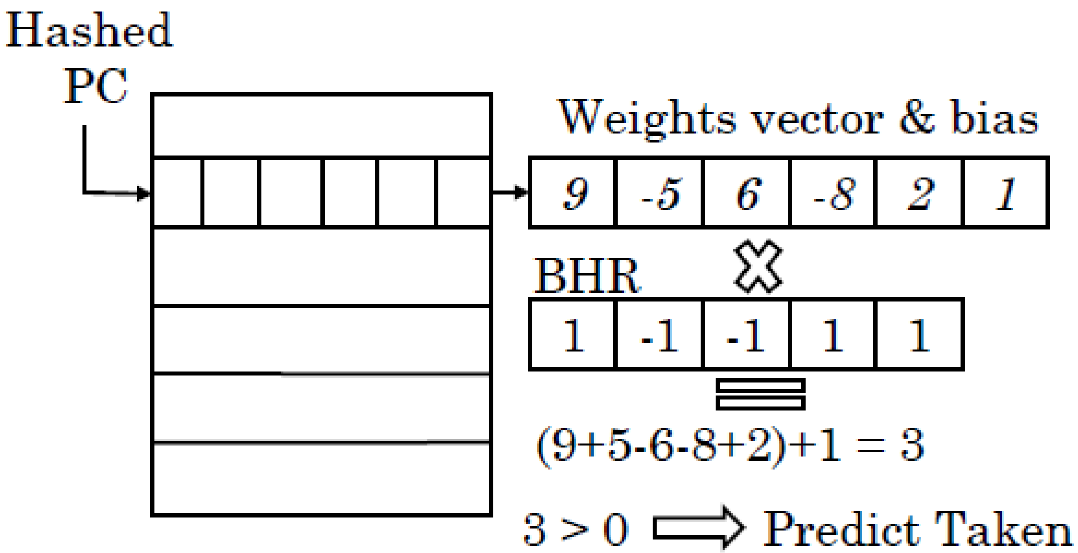
```
if |y| ≤ θ or ((y ≥ 0) ≠ t) then
    for each 0 ≤ i ≤ n in parallel
        if t = x_i then
            w_i := w_i + 1
        else
            w_i := w_i − 1
        end if
    end for
end if
```

For our course, we ignore this condition => we will always retrain the predictor after each prediction

What Do The Weights Mean?

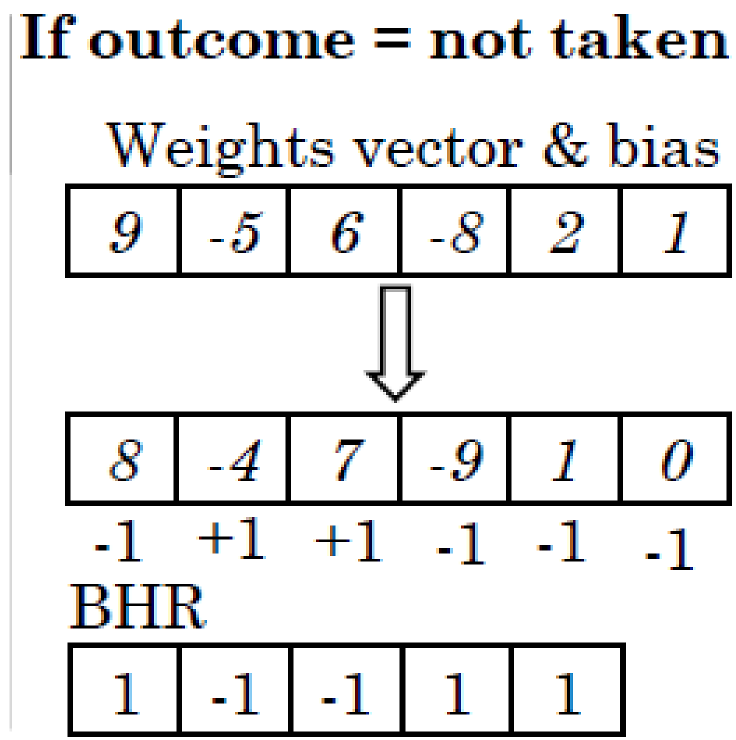
- The bias weight, w_0:
  - Proportional to the probability that the branch is taken
  - Doesn't take into account other branches; just like a Smith predictor
- The correlating weights, w_1 through w_n:
  - w_i is proportional to the probability that the predicted branch agrees with the i^{th} branch in the history
- The dot product of the w's and x's
  - $w_i \times x_i$ is proportional to the probability that the predicted branch is taken based on the correlation between this branch and the i^{th} branch
  - Sum takes into account all estimated probabilities
- What's θ?
  - Keeps from overtraining; adapt quickly to changing behavior

Example: (1 of 3)



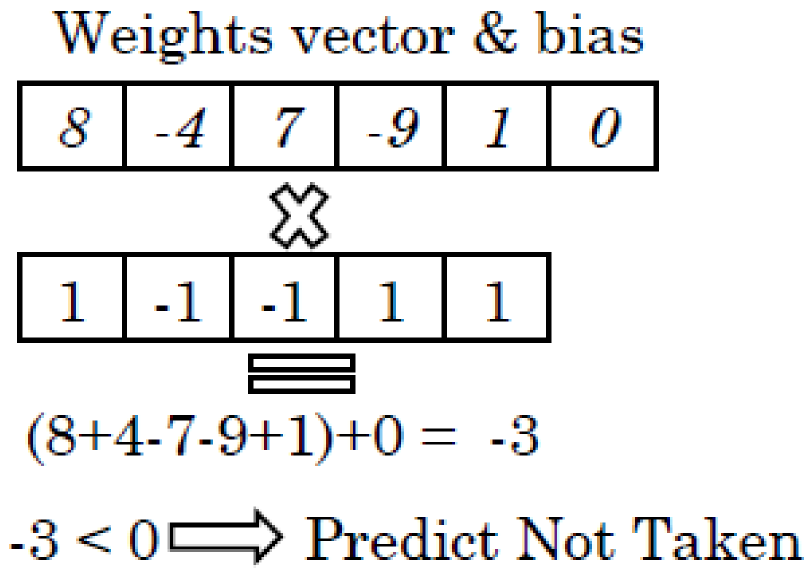
(a) First time prediction

Example: (2 of 3)



(b) Retraining

Example: (3 of 3)



(c) Next time prediction

Mathematical Intuition

A perceptron defines a hyperplane in *n*+1-dimensional space:

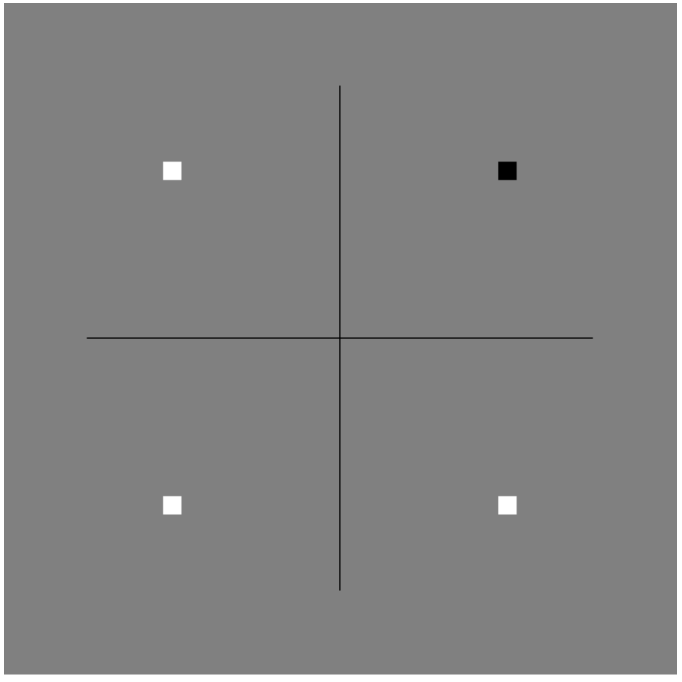
$$y = w_n x_n + w_{n-1} x_{n-1} + \ldots + w_1 x_1 + w_0$$

For instance, in 2D space we have: $y = w_1 x_1 + w_0$

This is the equation of a line, the same as $y = mx + b$

Example: AND

- Here is a representation of the AND function
- White means *false*, black means *true* for the output
- -1 means *false*, +1 means *true* for the input



-1 AND -1 = false

-1 AND +1 = false

+1 AND -1 = false

+1 AND +1 = true

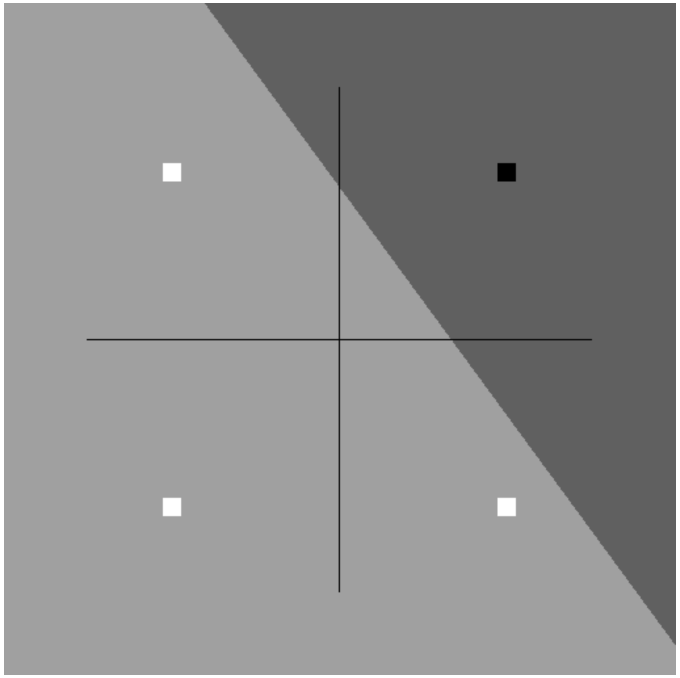
Program to Compute AND

```
int f () {
	int a, b, x, i, s = 0;

	for (i=0; i<100; i++) {
		a = rand () % 2;
		b = rand () % 2;
		if (a) {
			if (b)
				x = 1;
			else
				x = 0;
		} else {
			if (b)
				x = 0;
			else
				x = 0;
		}
		if (x) s++; /* this is the branch */
	}
	return s;
}
```

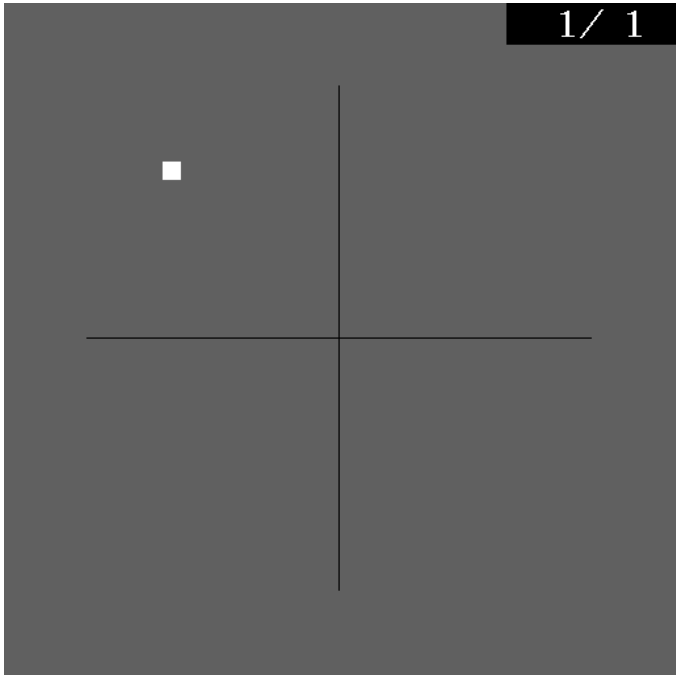
Example: AND continued

- A linear decision surface (i.e. a plane in 3D space) intersecting the feature space (i.e. the 2D plane where z=0) separates *false* from *true* instances



Example: AND continued

- Watch a perceptron learn the AND function:



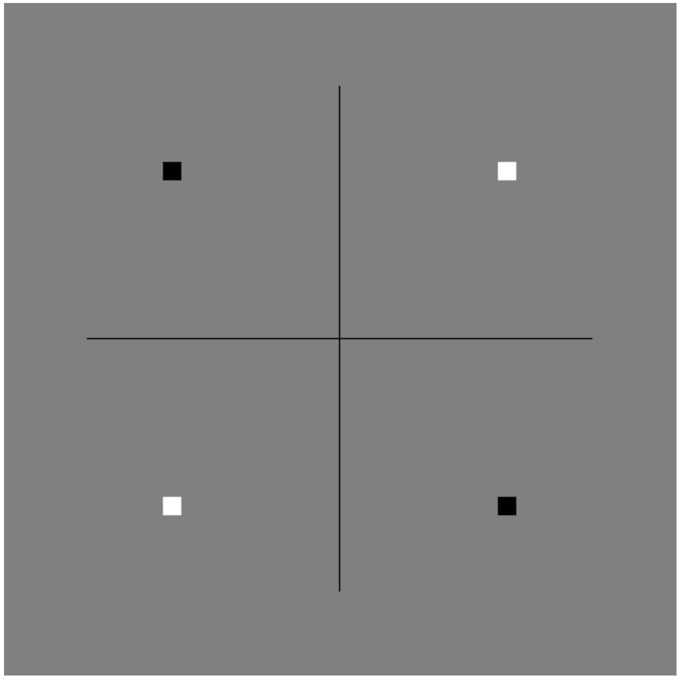
Program to Compute XOR

```
int f () {
        int a, b, x, i, s = 0;

        for (i=0; i<100; i++) {
                a = rand () % 2;
                b = rand () % 2;
                if (a) {
                        if (b)
                                x = 0;
                        else
                                x = 1;
                } else {
                        if (b)
                                x = 1;
                        else
                                x = 0;
                }
                if (x) s++; /* this is the branch */
        }
        return s;
}
```

Example: XOR

- Here's the XOR function:



-1 XOR -1 = *false*

-1 XOR +1 = *true*

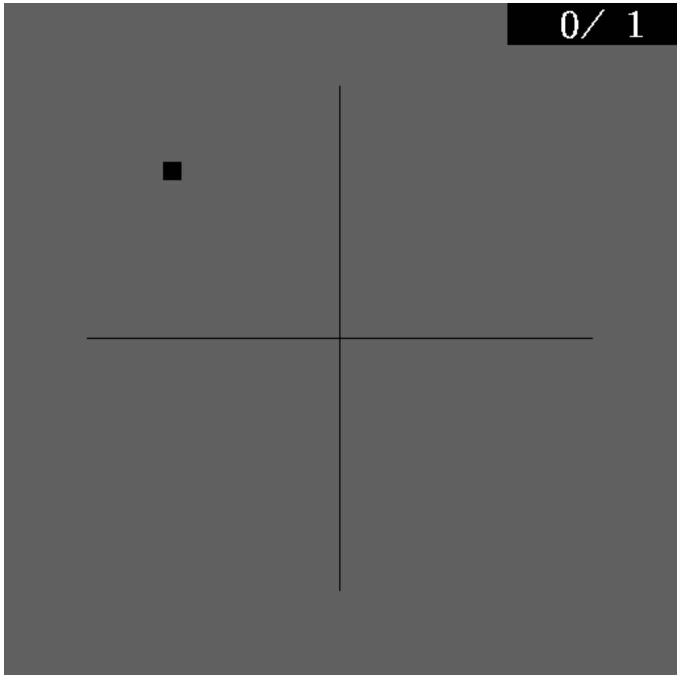
+1 XOR -1 = *true*

+1 XOR +1 = *false*

Perceptrons cannot learn such *linearly inseparable* functions

Example: XOR continued

- Watch a perceptron try to learn XOR



Concluding Remarks

- Perceptron is an alternative to traditional branch predictors
- Provides high accuracy
- Limitations:
  - Latency
  - Linear inseparability

Idealized Piecewise Linear Branch Prediction

Daniel A. Jiménez

This and subsequent slides are not part of CS2323 course.

They are included just for illustration.

Previous Neural Predictors

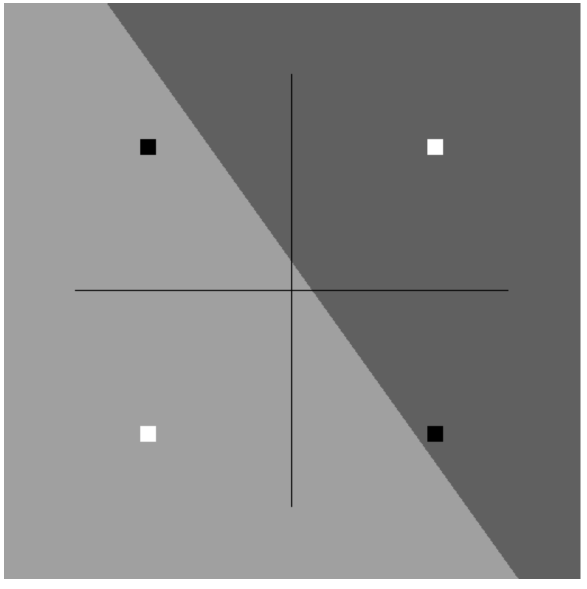
- The perceptron predictor uses only pattern history information
  - The same weights vector is used for every prediction of a static branch
  - The i^{th} history bit could come from any number of static branches
  - So the i^{th} correlating weight is aliased among many branches
- The newer path-based neural predictor uses path information
  - The i^{th} correlating weight is selected using the i^{th} branch address
  - This allows the predictor to be pipelined, mitigating latency
  - This strategy improves accuracy because of path information
  - But there is now even more aliasing since the i^{th} weight could be used to predict many different branches

Piecewise Linear Branch Prediction

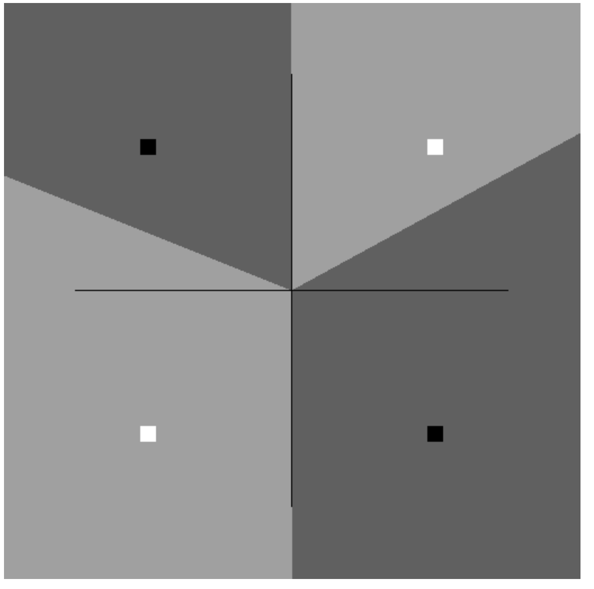
- Generalization of perceptron and path-based neural predictors

Why It's Better

- Forms a piecewise linear decision surface
  - Each piece determined by the path to the predicted branch
- Can solve more problems than perceptron



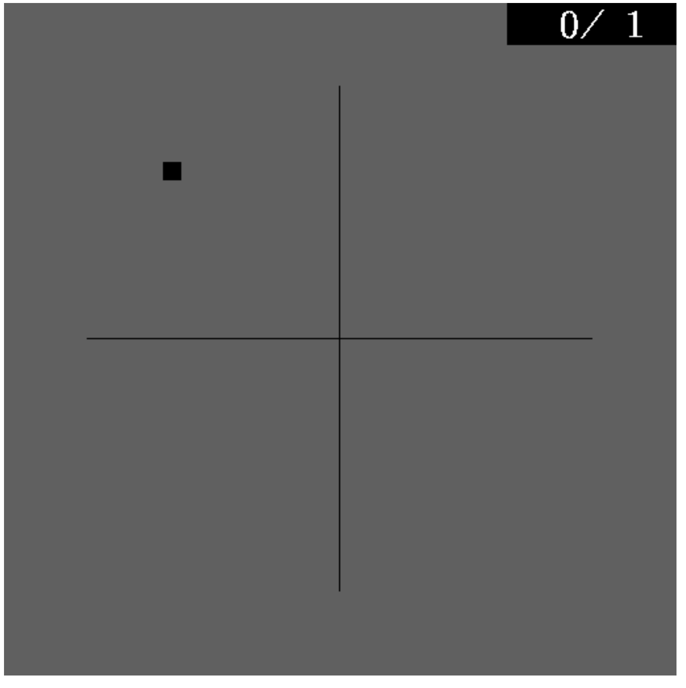
Perceptron decision surface for XOR doesn't classify all inputs correctly



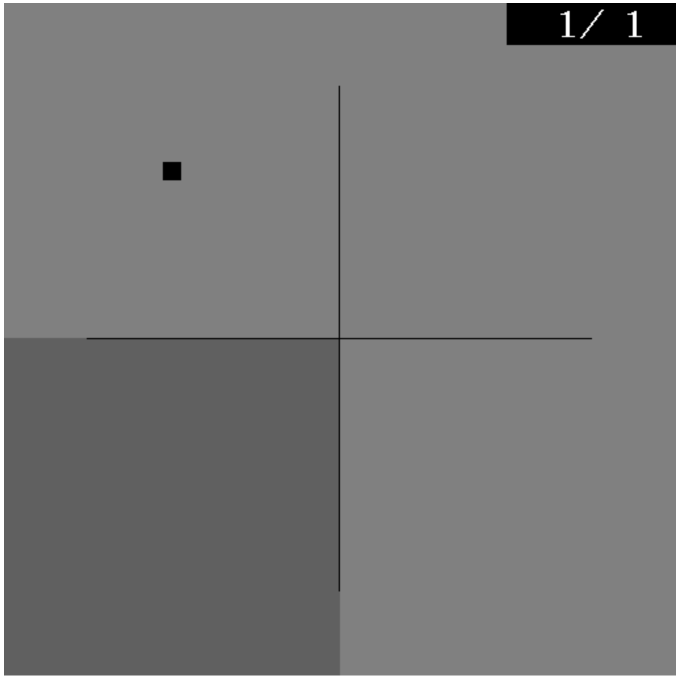
Piecewise linear decision surface for XOR classifies all inputs correctly

Learning XOR

- From a program that computes XOR using `if` statements



perceptron prediction



piecewise linear prediction