INF367 25H: Selected Topics in Artificial Intelligence

Diamonds and Rust in the AI Treasure Chest

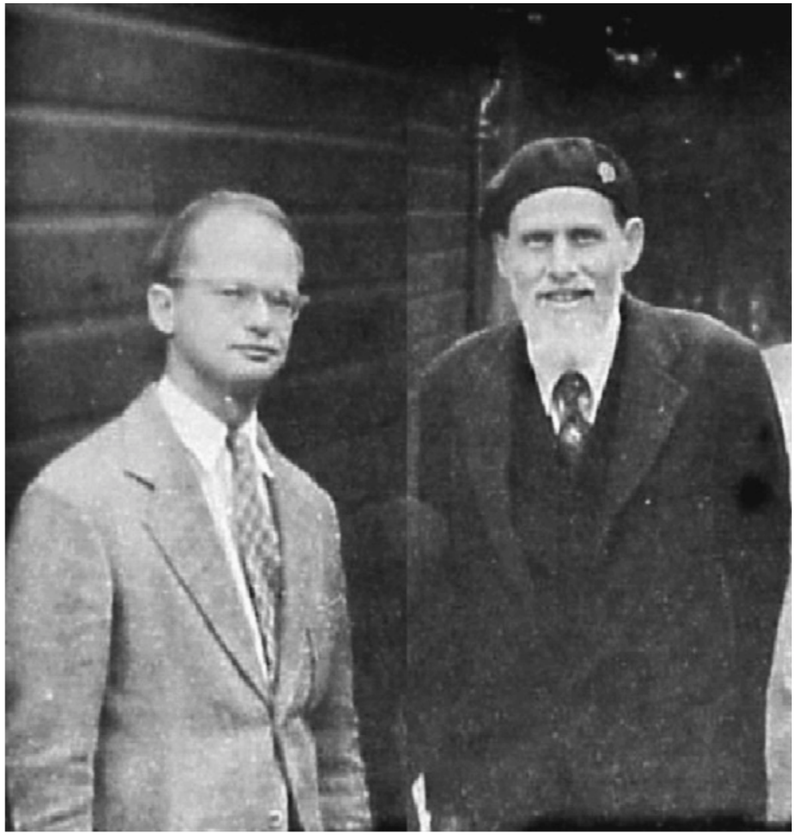
Dario Garigliotti - UiB - Lecture 3

Plan for today

- Recap from last lecture
- Primer on vector algebra
- Perceptron algorithm
- Activity
- Perceptron, again
- L vs h

Recap

First artificial neuron

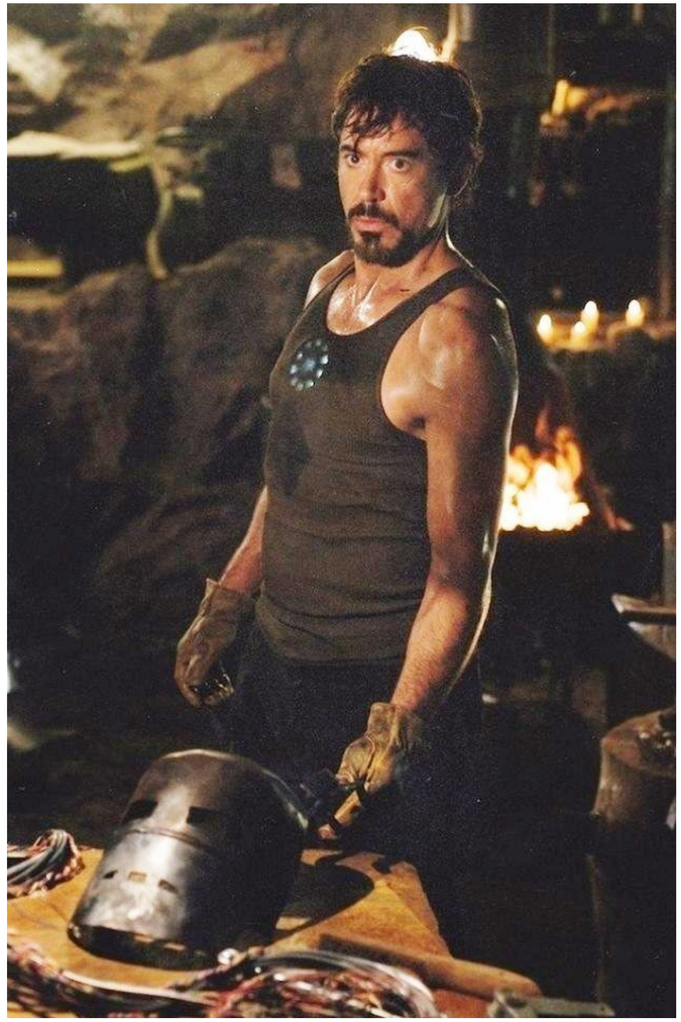


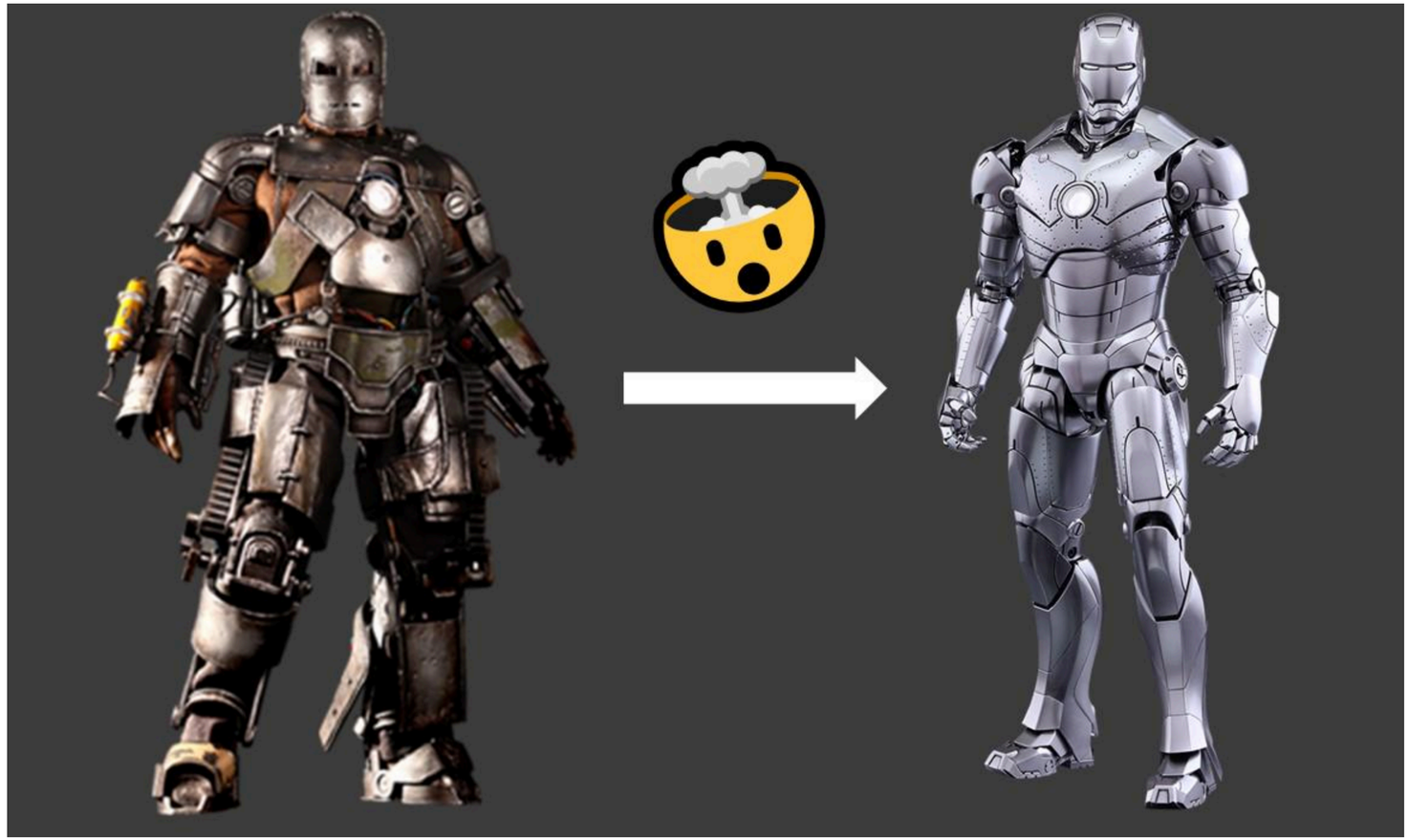
- MCP neuron, introduced by McCulloch and Pitts in 1943
- Context: “all computation reduced to logic”
  - *If brain is a computational device, how does it implement this logic?*
- Perform any computation by mixing basic units
- Unit of computation, without learning

Perceptron

- Combines
  - MCP: biologically-inspired unit of computation, extended by networks
  - Hebbian learning: learning due to strengthening neural connections as output of one neuron consistently fires into input of another one
- I.e. basic neuron model with learning algorithm
  - Learned weights “encode”/represent knowledge that is “remembered”

Perceptron - Mark I





Perceptron - Mark I

Illustration of the Mark 1 perceptron hardware. The photograph on the left shows how the inputs were obtained using a simple camera system in which an input scene, in this case a printed character, was illuminated by powerful lights, and an image focussed onto a 20×20 array of cadmium sulphide photocells, giving a primitive 400 pixel image. The perceptron also had a patch board, shown in the middle photograph, which allowed different configurations of input features to be tried. Often these were wired up at random to demonstrate the ability of the perceptron to learn without the need for precise wiring, in contrast to a modern digital computer. The photograph on the right shows one of the racks of adaptive weights. Each weight was implemented using a rotary variable resistor, also called a potentiometer, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.

Vector algebra

Vectors and vector operations

- Vector: magnitude and direction
  - Coordinate notation
  - Magnitude as Euclidean distance
- Operations
  - Product by scalar
  - Sum of vectors
  - Dot product

Perceptron algorithm

Perceptron algorithm

- Perceptron finds a hyperplane that separates the linearly separable classes
  - By learning the corresponding weight vector
- Formalization of the perceptron model…
  - using vector notation
  - subsuming the bias as *w0*
- Algorithm for training the model = learning the weights

Activity

Activity: Perceptron and friends

Two possible problems

1. If the two classes are not linearly separable, will the perceptron algorithm converge?
   - After all, it's possible for this Python thing here to terminate with a linear predictor:

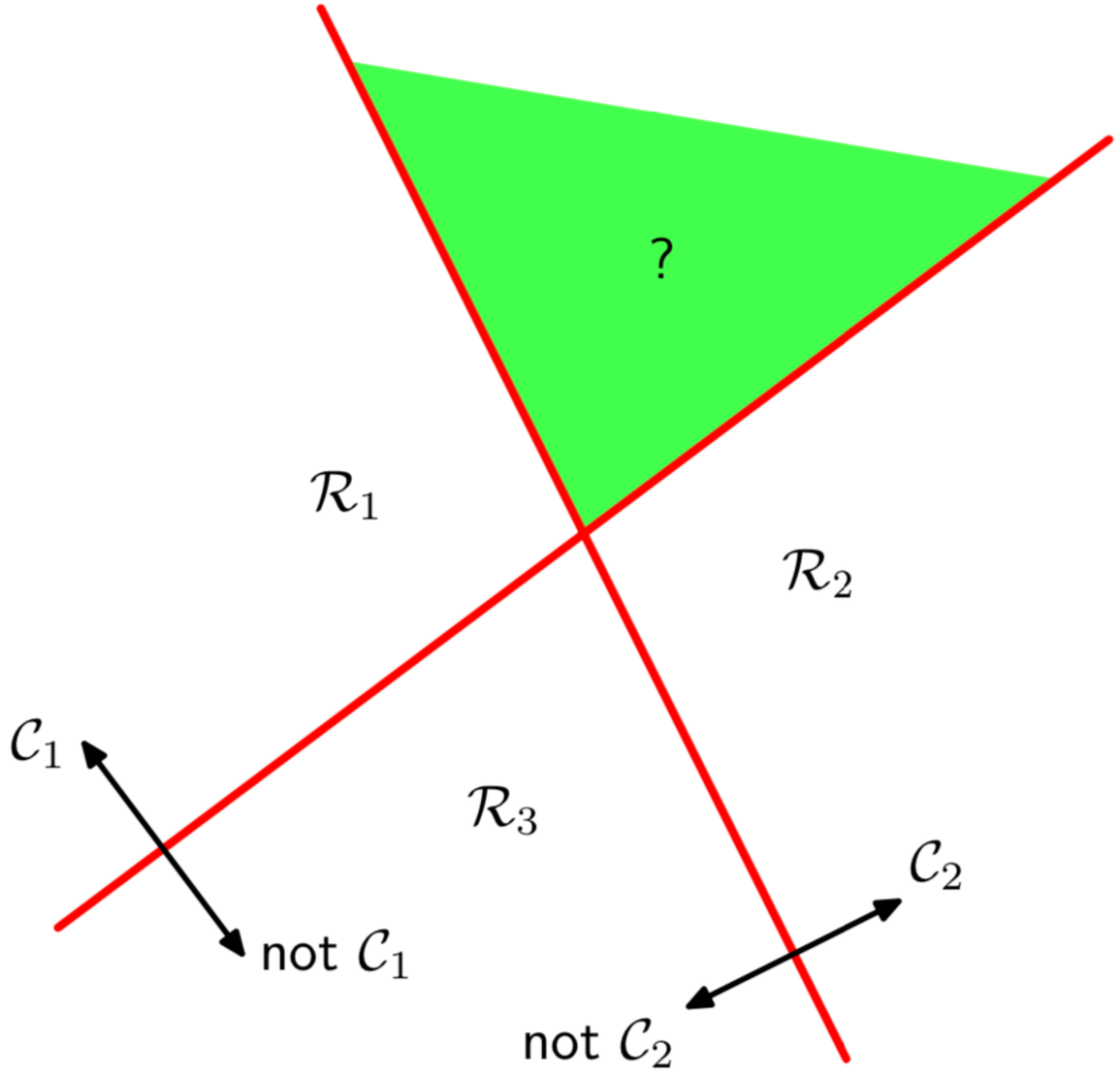
```
from sklearn.linear_model import Perceptron; clf = Perceptron(…);
clf.fit(…); clf.predict(…)
```

2. Consider a dataset with 3 possible output classes (e.g. {rainy, cloudy, sunny}).
   - What would it mean for these to be linearly separable?
   - If they are linearly separable, what kind of predictor could you build/learn to separate them?
   - What kind of issues could your efforts present?

2. Possible solution

One-vs-rest

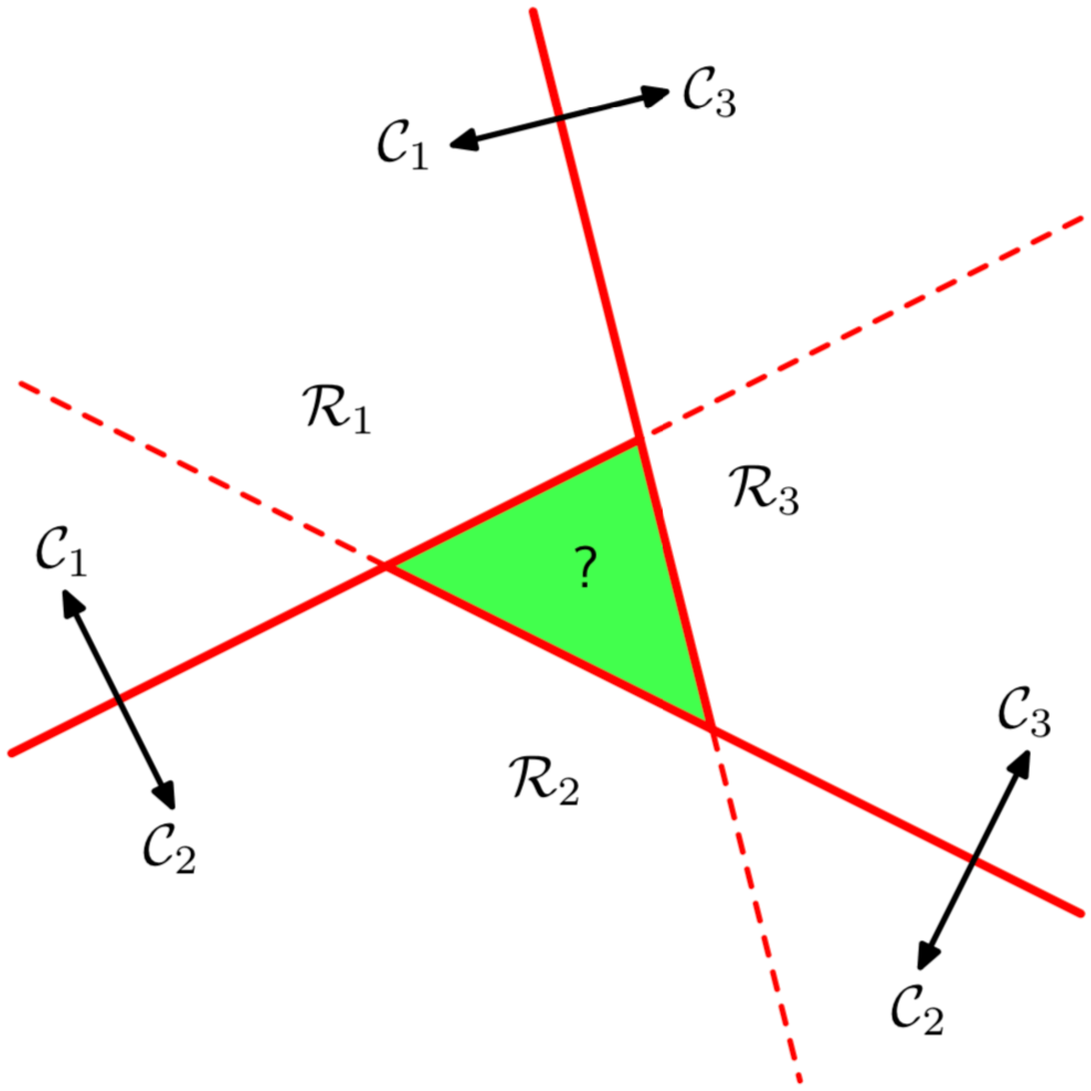
- For 3 classes,
  - use 3-1=2 classifiers
    - binary “one-vs-rest”



2. Possible solution

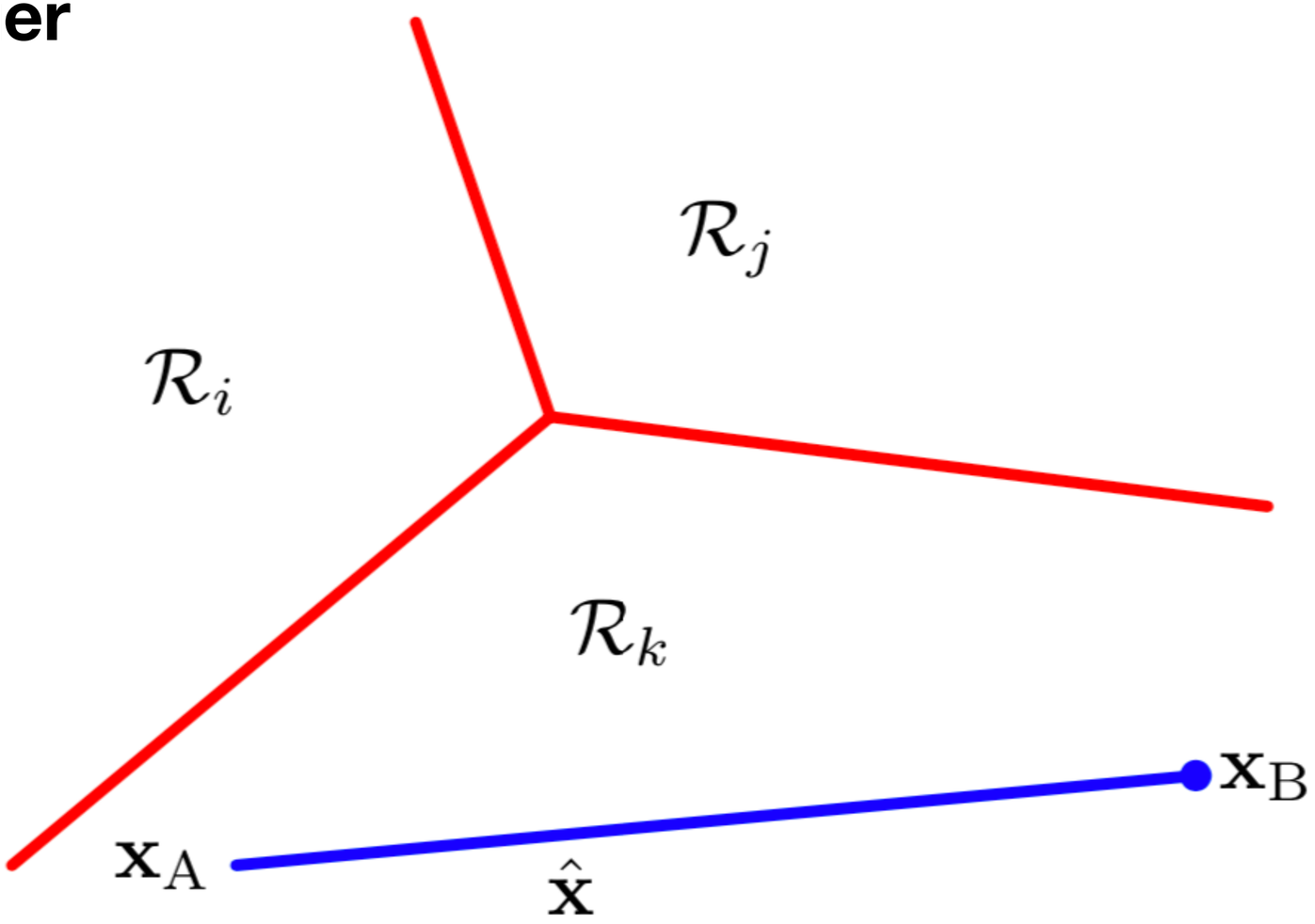
One-vs-one

- For 3 classes,
  - use 3(3-1)/2=3 classifiers
    - binary “one-vs-one”
  - Decision by majority



2. Possible solution

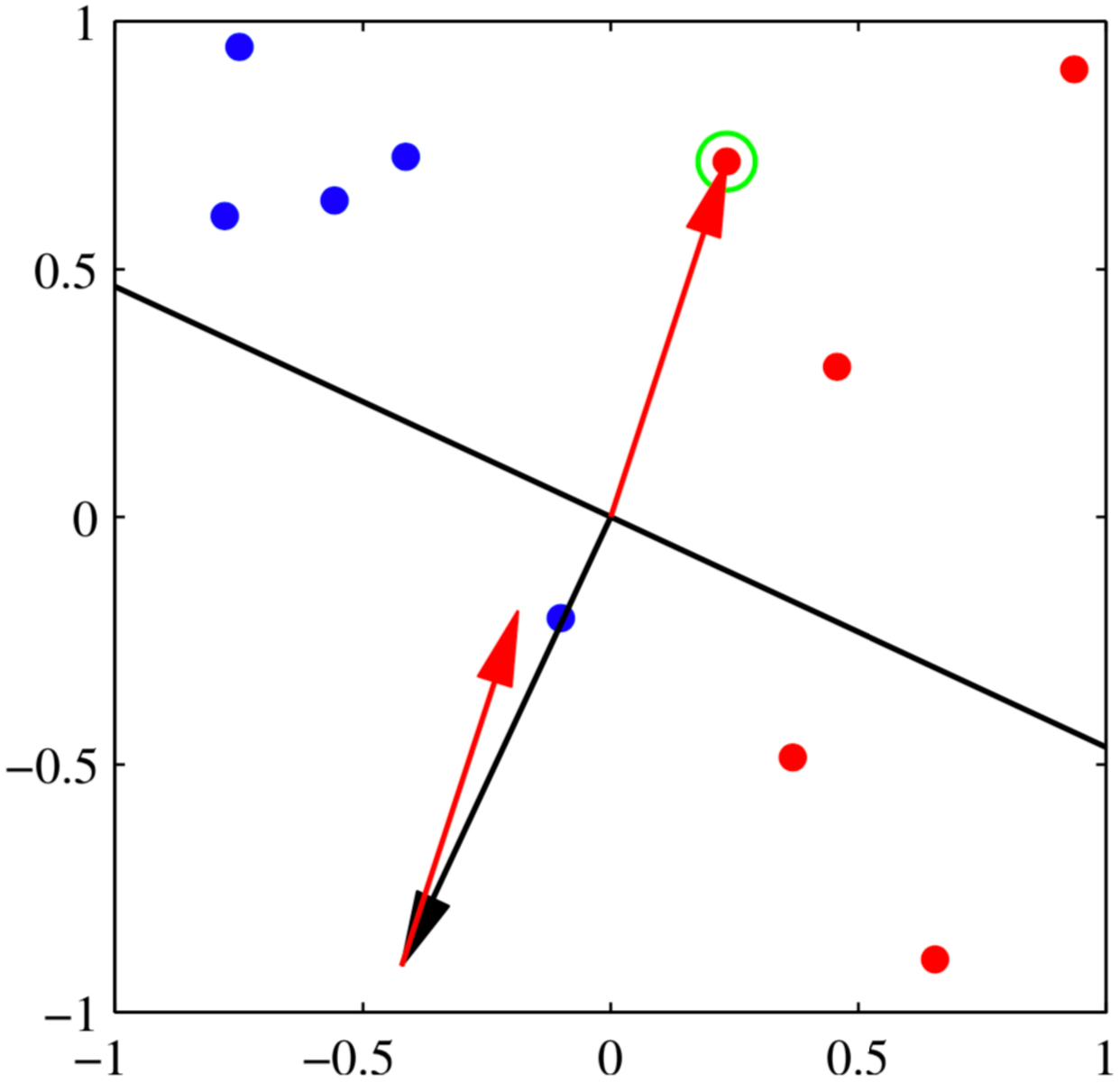
Single 3-class classifier



Perceptron algorithm, again

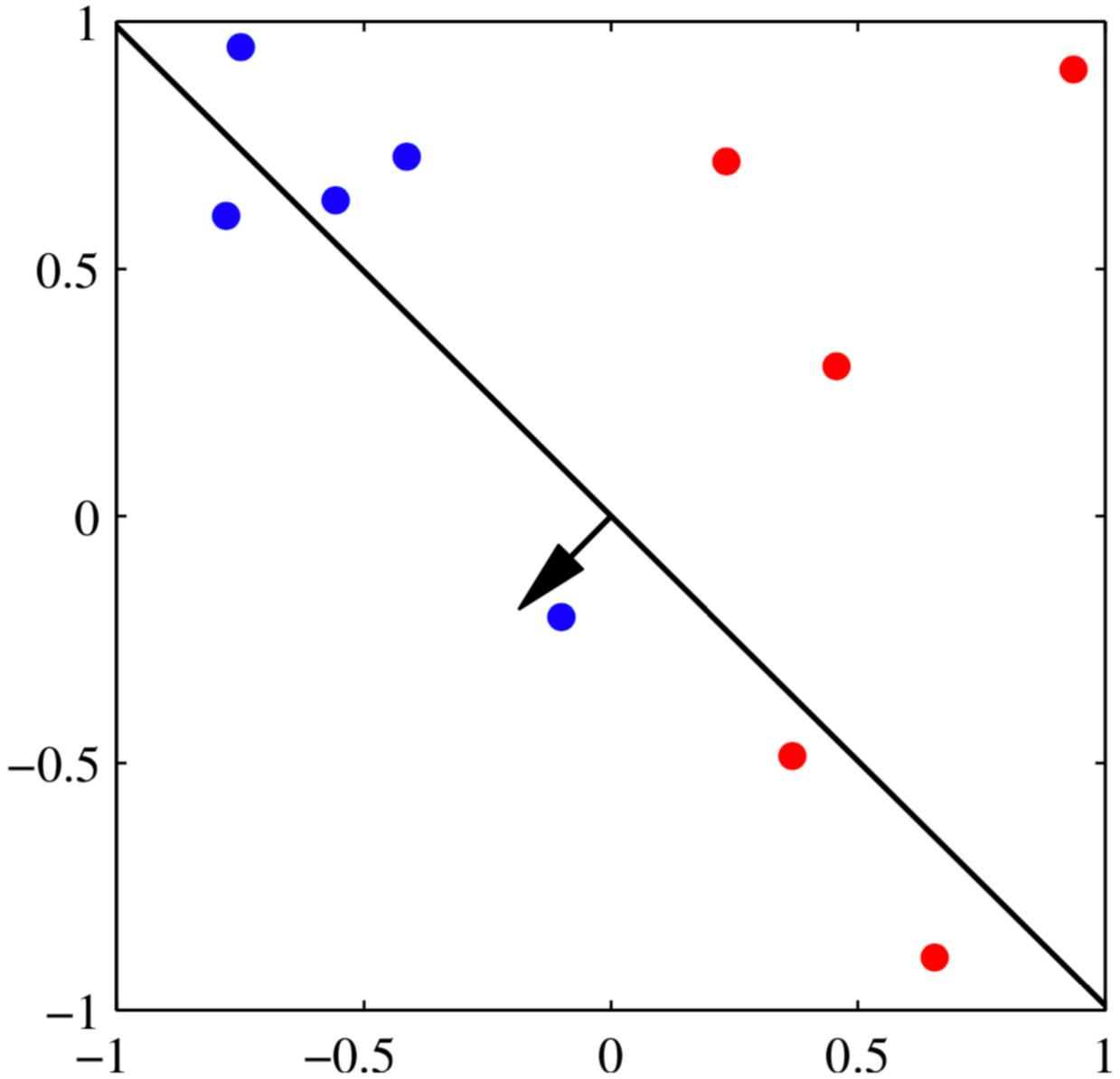
Perceptron

Example step 1 of 4



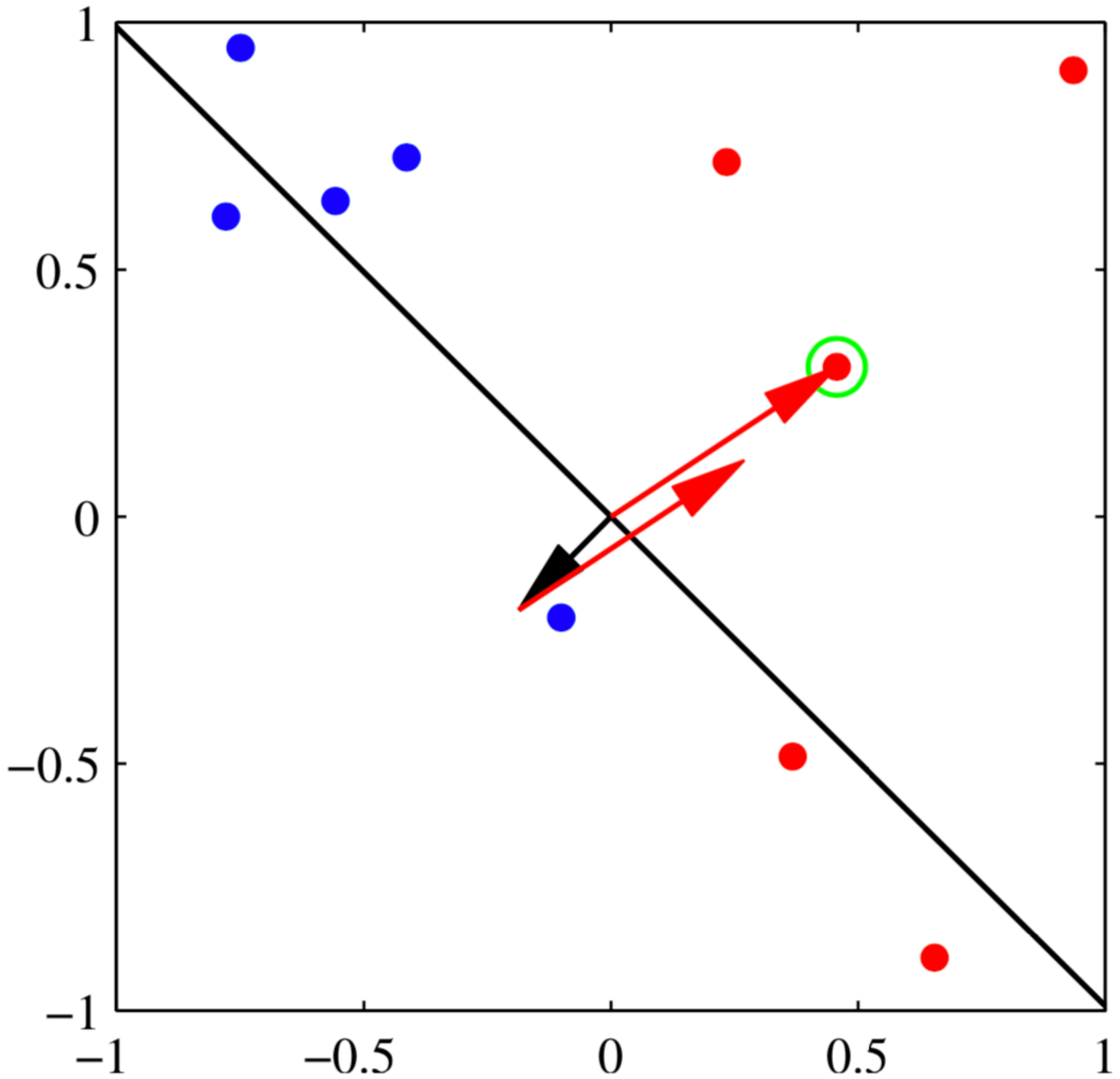
Perceptron

Example step 2 of 4



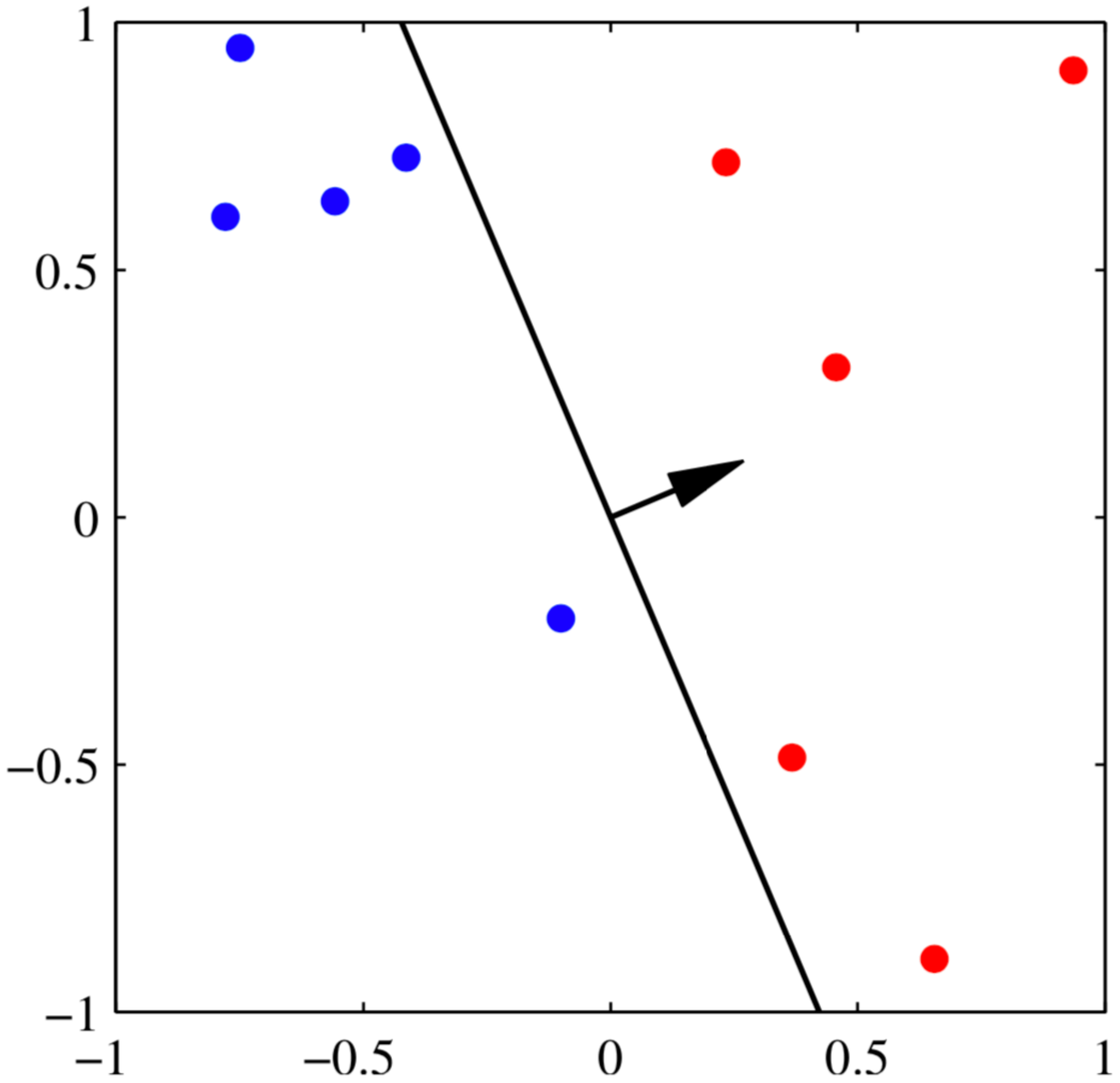
Perceptron

Example step 3 of 4



Perceptron

Example step 4 of 4



Learner vs Predictor

Learner vs Predictor/Model

- Some phenomenon *ph* in the real world *RW*
- Some data distribution \cal{Distrib} underlying *ph* according to the true function *f* that “maps input into output” in *RW*
- A dataset *D* ~ \cal{Distrib}, *D* = {(*x_i, y_i*) : *i* = 1..*n*}
- An approximator *h** in *H* = {*h* : *h* approximates *f*} = set of models or predictors
- In relation with *H*, a learner *L* learns *h*’s from *D: L(D)=h*
  - E.g. H1 = {decision trees…}, H2 = {decision forests…}, H3 = {neural nets…}