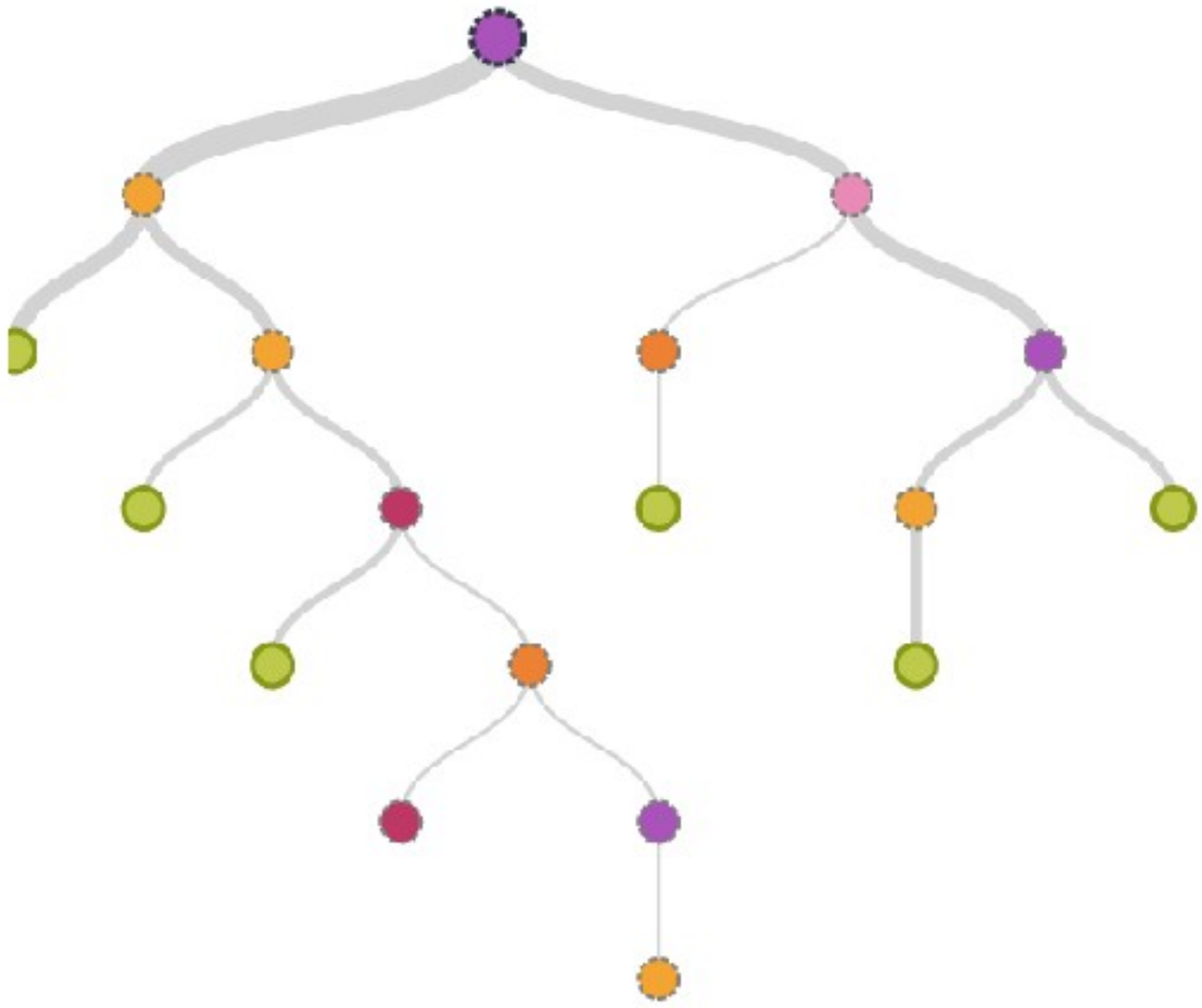
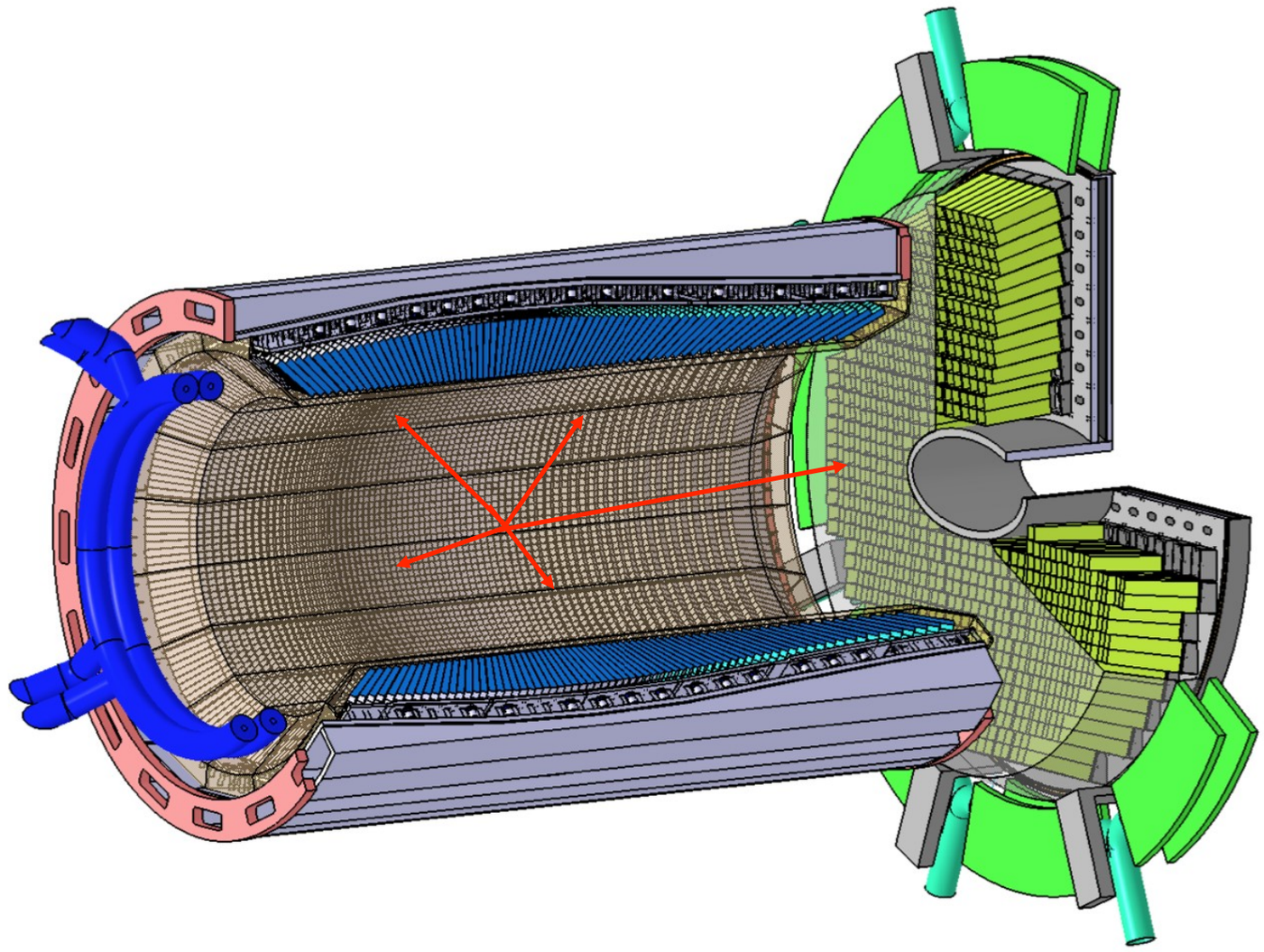
Data Analysis
using
Machine Learning

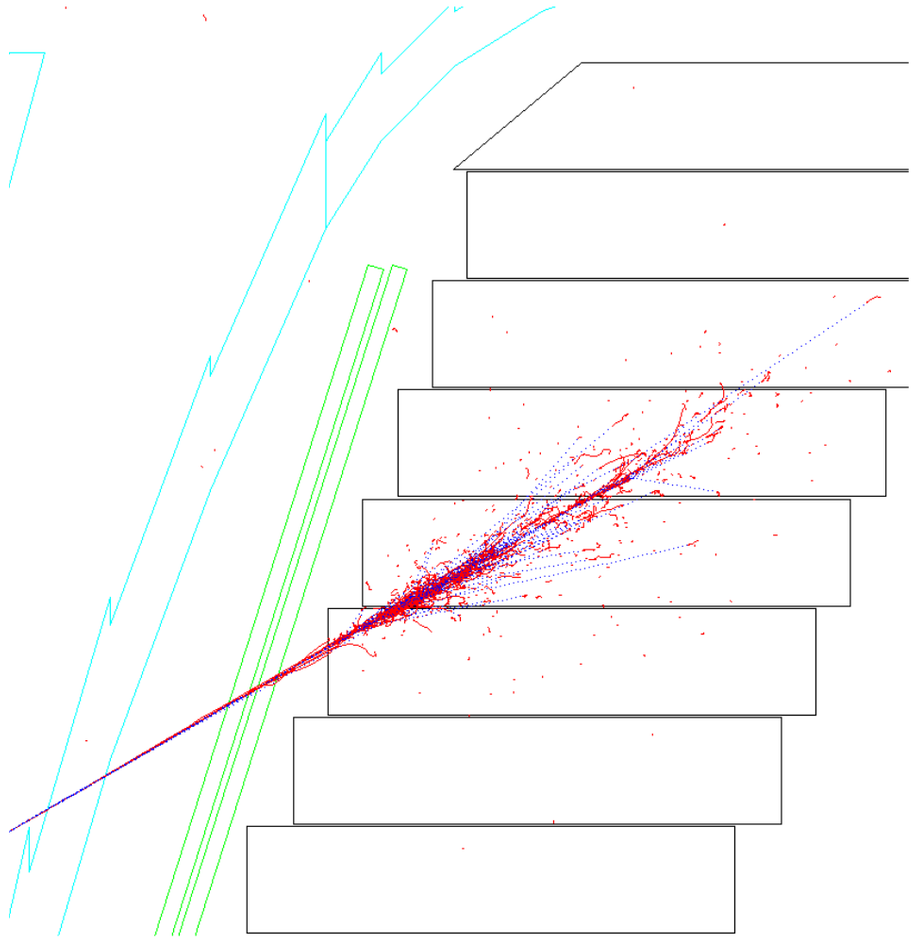


Myroslav Kavatsyuk, CMST

Assignment IV: Machine Learning

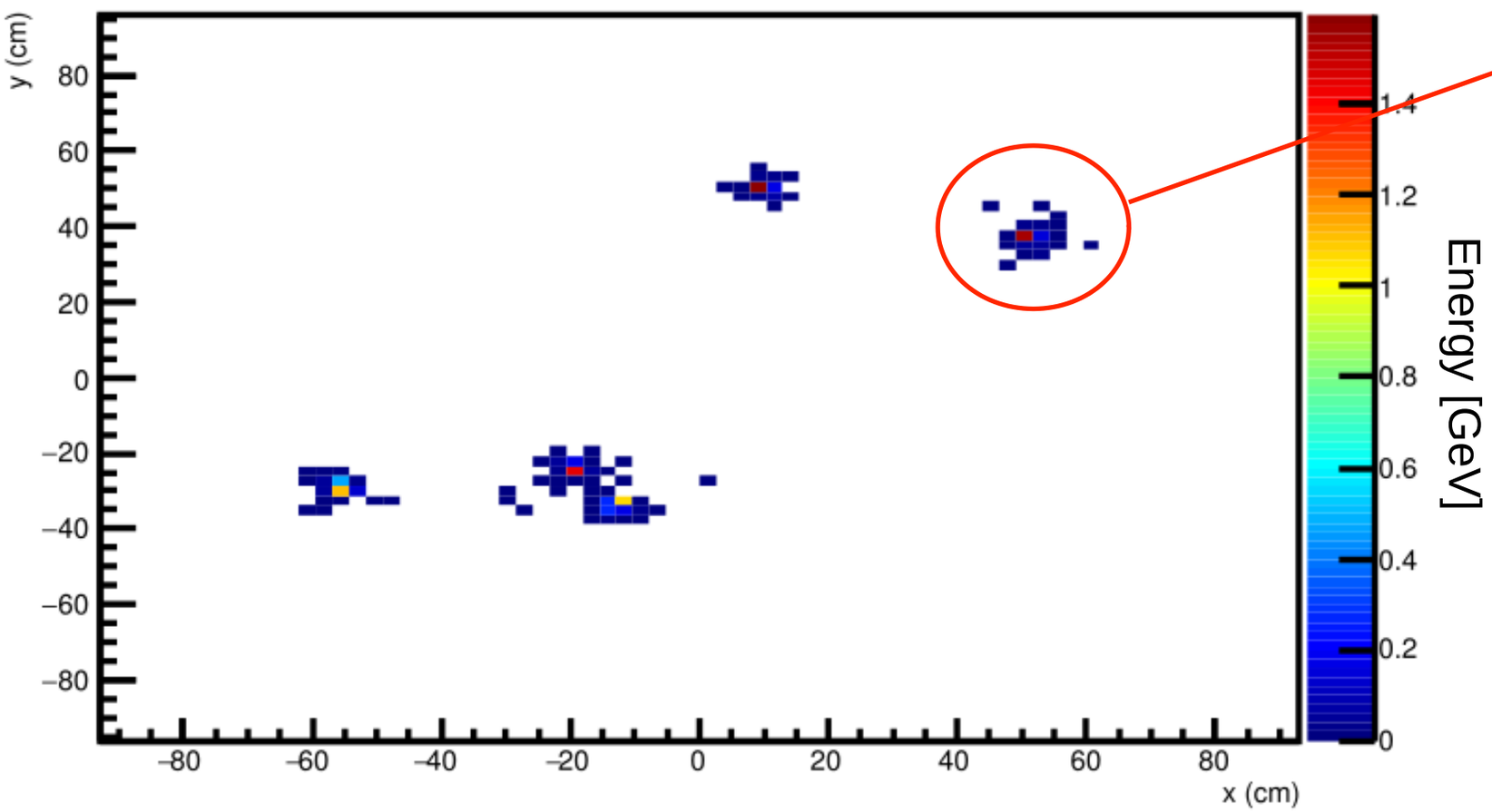
- **Monte Carlo simulated data of photons (S) and neutrons (B) hitting the PANDA detector**





Assignment IV: Machine Learning

- **Instances are clusters of firing crystals with pre-processed cluster-property parameters as features!**



Features:

- Theta, Phi, E-dep
- NrHits, NrBumps
- E1, E1E9, E9E25
- Z20, Z53, LatMom

Classes:

- Photon
- Neutron

Assignment IV: Machine Learning

- ***Challenge*: use ML to suppress neutron background while optimizing the photon detection efficiency**

- **Performance metric ("figure-of-merit"):**

$$FOM = \frac{S}{\sqrt{S+B}}$$

Number of left-over photon clusters after classification

Equivalent to the statistical significance accounting for background!!!

Number of left-over neutron clusters after classification

- **Use 50% of data for training and 50% for testing**

Assignment IV: Machine Learning

- **You are allowed to use part of my example code. Add a reference note and don't use it blindly!**
- **<u>Differences</u> between "fruit" and "emc" cases:**
  - **The #instances in "emc" data is much larger (~87.000 versus 179)! Some algorithms may get slow (f.e. kNN with larger "k"). Develop a strategy on how to deal with that!**
  - **The complexity of "emc" data is much larger, more features/per instance! How to effectively select the promising ones, without wasting CPU time!**
  - **The metrics differ:**
    - **"fruit"=<u>minimize error</u> of majority vote**
    - **"emc"=<u>maximize the FOM</u>!**

Assignment IV: Machine Learning

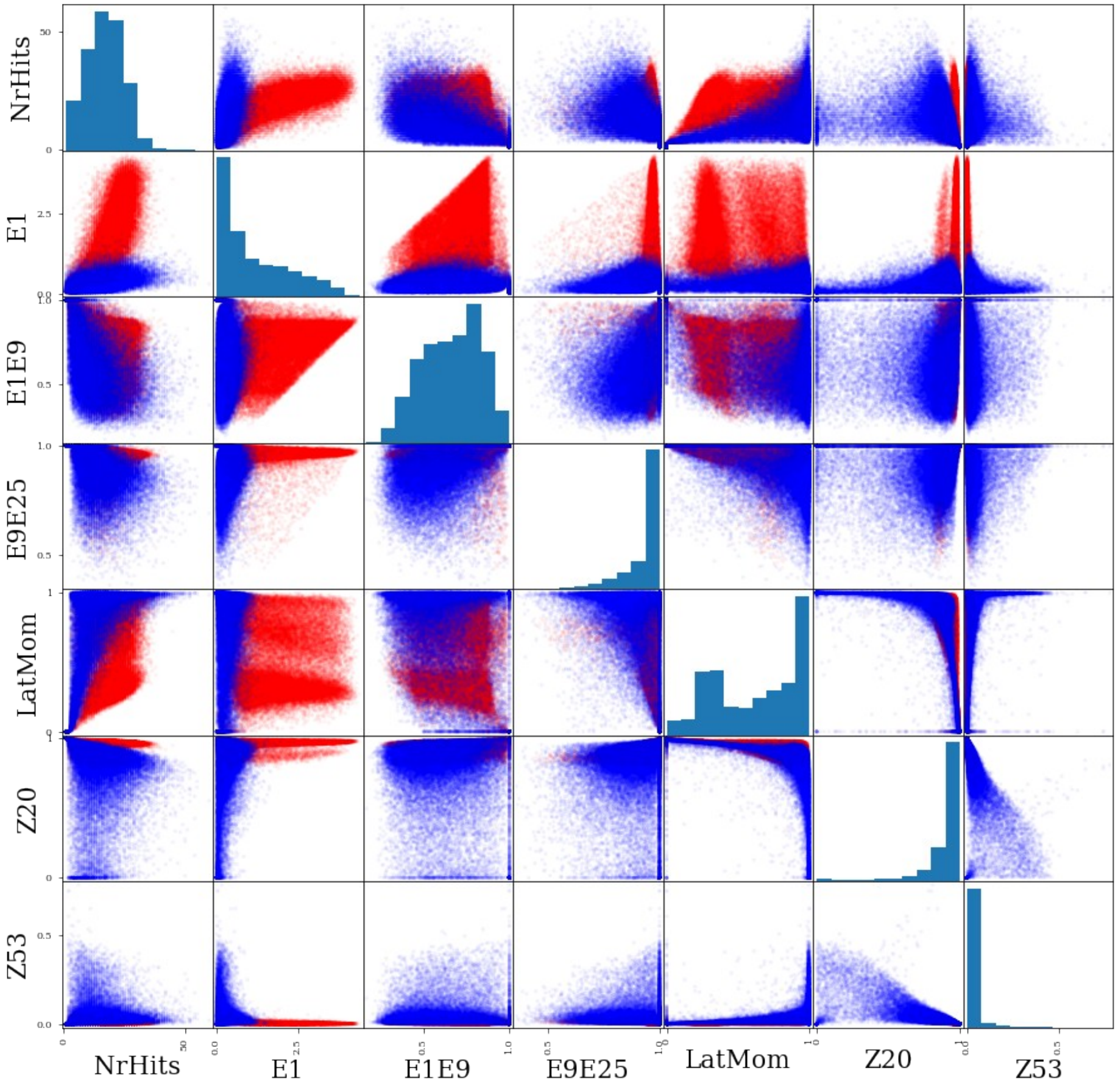
- **Few words on maximizing FOM:**
  - ***Think*: what is the ideal FOM you hope to expect? What is FOM for a worse-case scenario?**
  - **Optimize parameters of classifiers with respect to FOM!**
- **For example, for *k*NN, FOM will depend on…**

#nearest neighbors

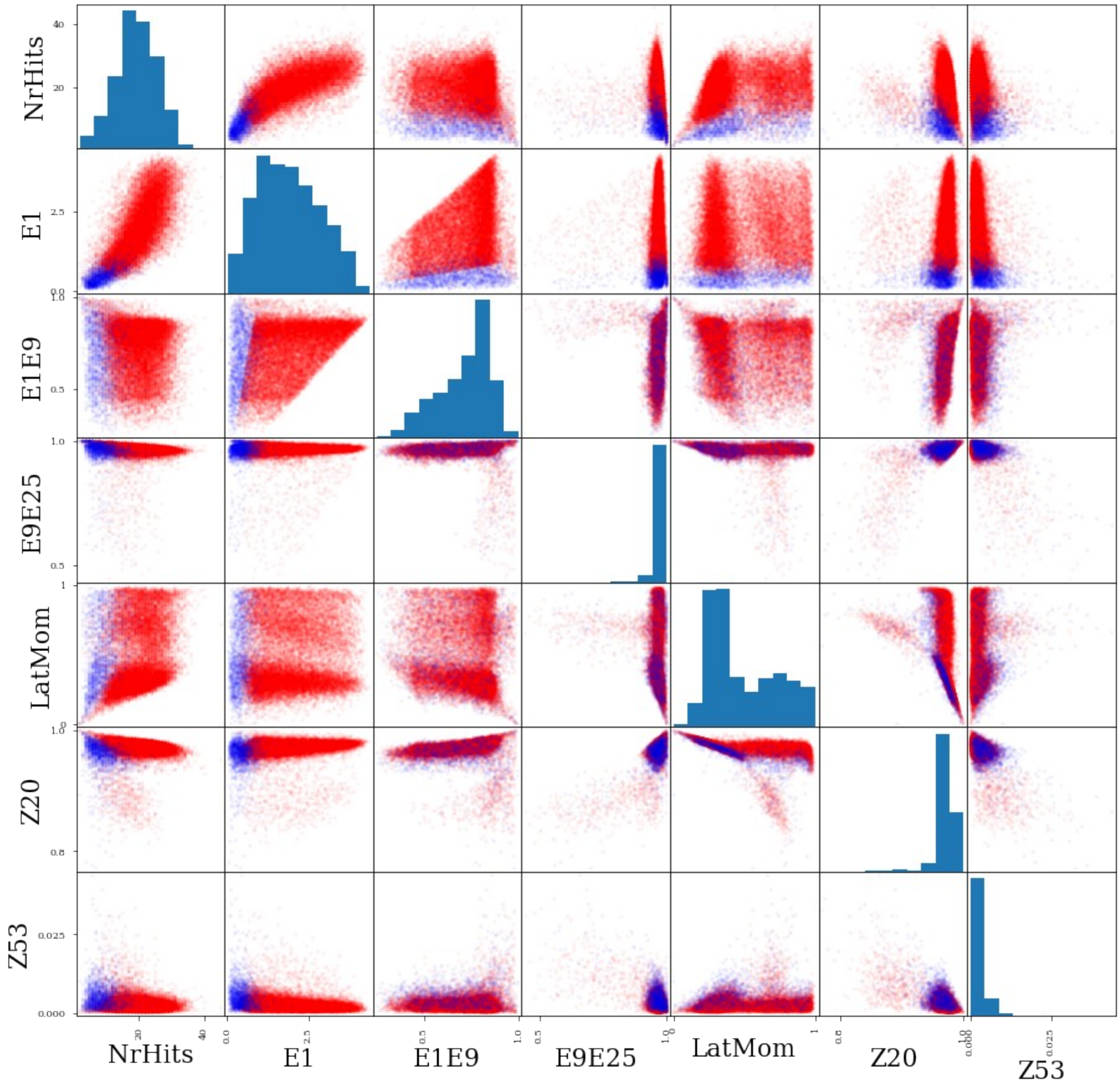
$$FOM(p,k) = \frac{S(p,k)}{\sqrt{S(p,k)+B(p,k)}}$$

Probability of instance being a photon cluster
(Tip, use *MLPClassifier.predict_proba()*)

Correlation plot (raw data)



Correlation plot (after kNN classification)



Input features: E1E9, E9E25, Z20, Z53, LatMom

Supervised

Data starts with labels AND measurements ("features")

Trying to find patterns in the measurement that are associated with labels, so that if there are new instances and features, we can predict the new label.

(e.g. a new fruit of a certain color, elongatedness, weight, acidity and sweetness -- what kind of fruit is it?)

PREDICTIVE data analysis

kNN, pros and cons

Pros:

- kNN = Multi-dimensional “density” estimator
- Easy algorithm, easy to understand
- Can provide a confidence (probability)
- In *principle*, very powerful

Cons:

- kNN requires high statistics for high dimensions
- Requires smart trees & data storage
- Not very well suited for category-type variables
- kNN requires optimising “k”

We've done k-Nearest Neighbors;

now let's try a different supervised learning algorithm.

Decision trees

Basically like a game of "Twenty Questions" with our data.

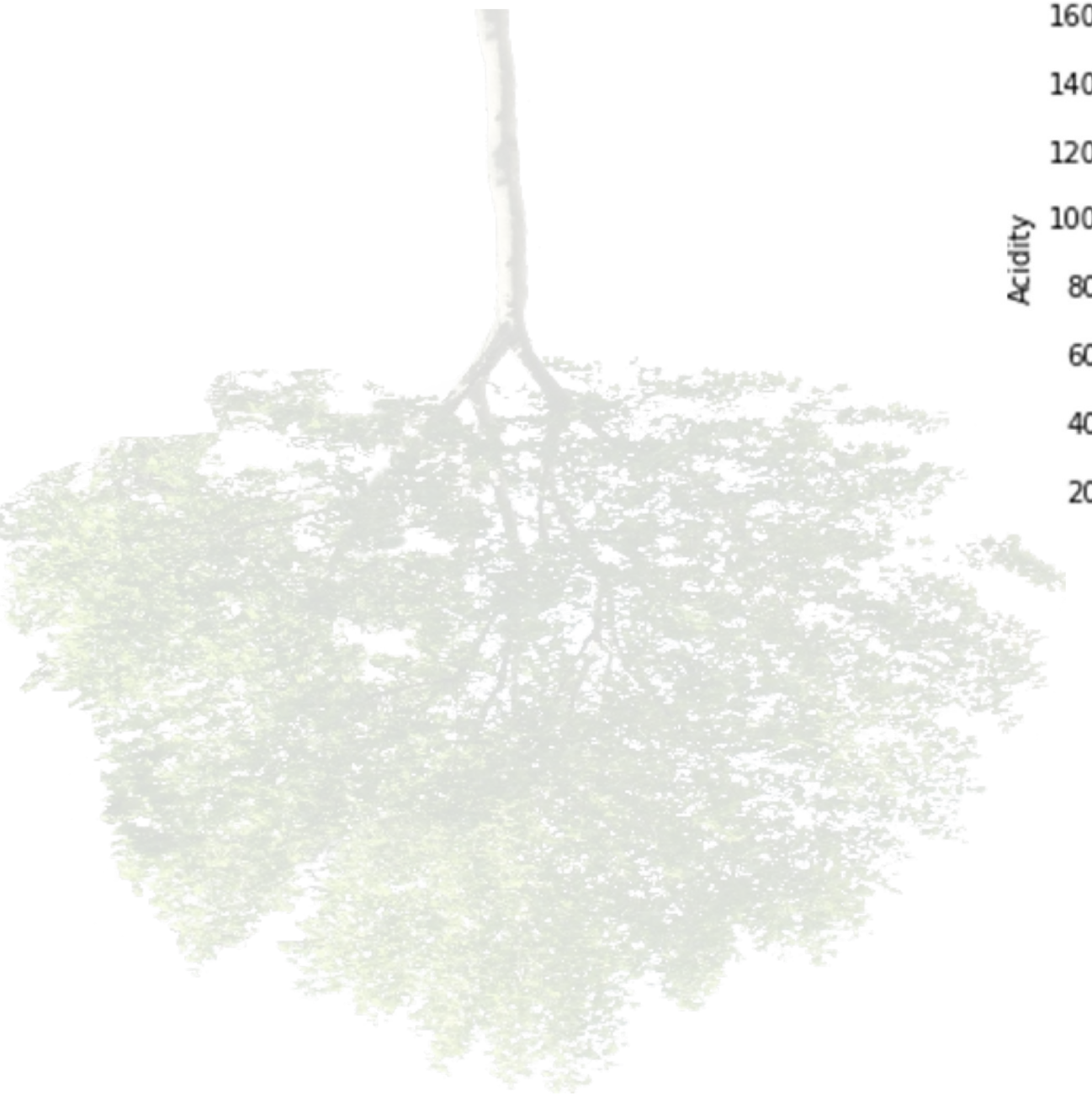
The algorithm tries to come up with good questions.

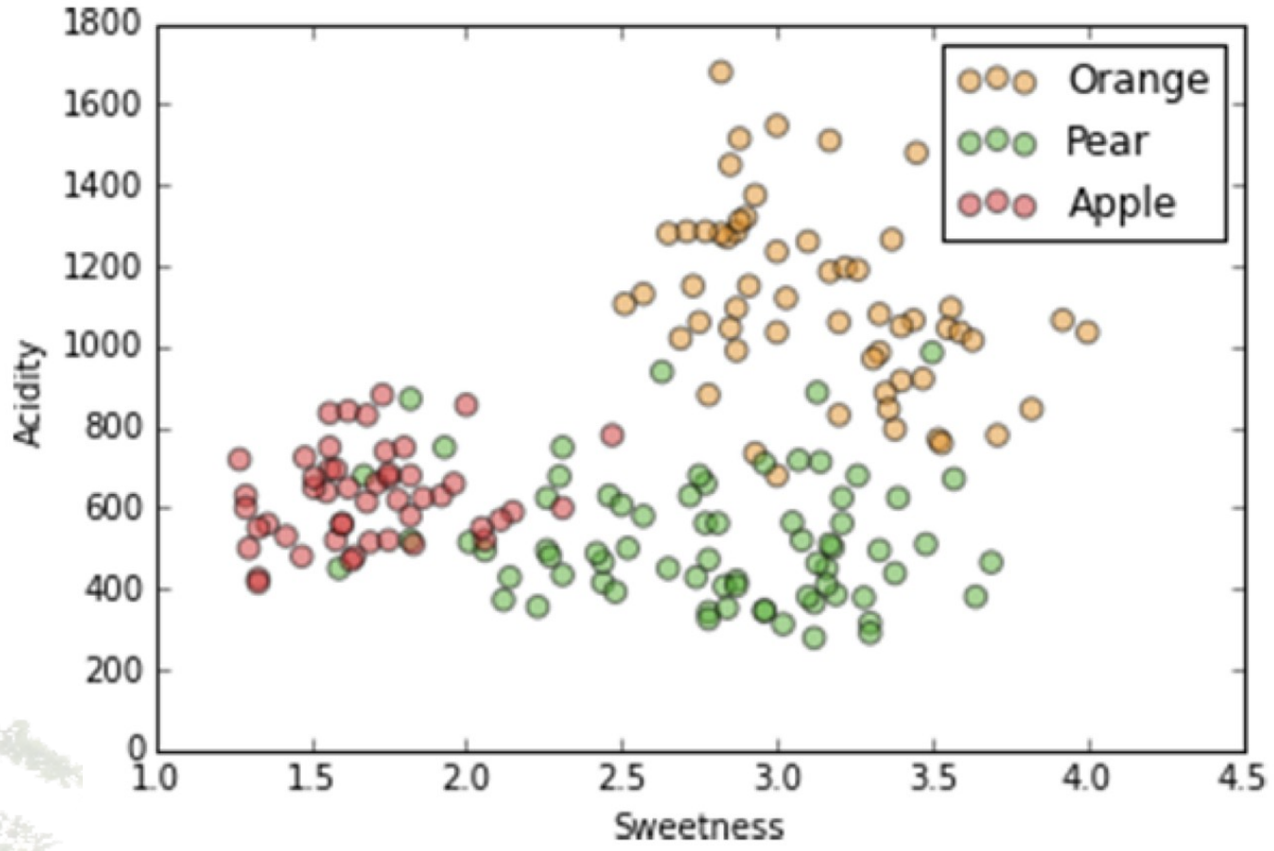
If I asked you to play twenty questions, thinking of a famous person, what would be a better first question:

* Is the person male or female?

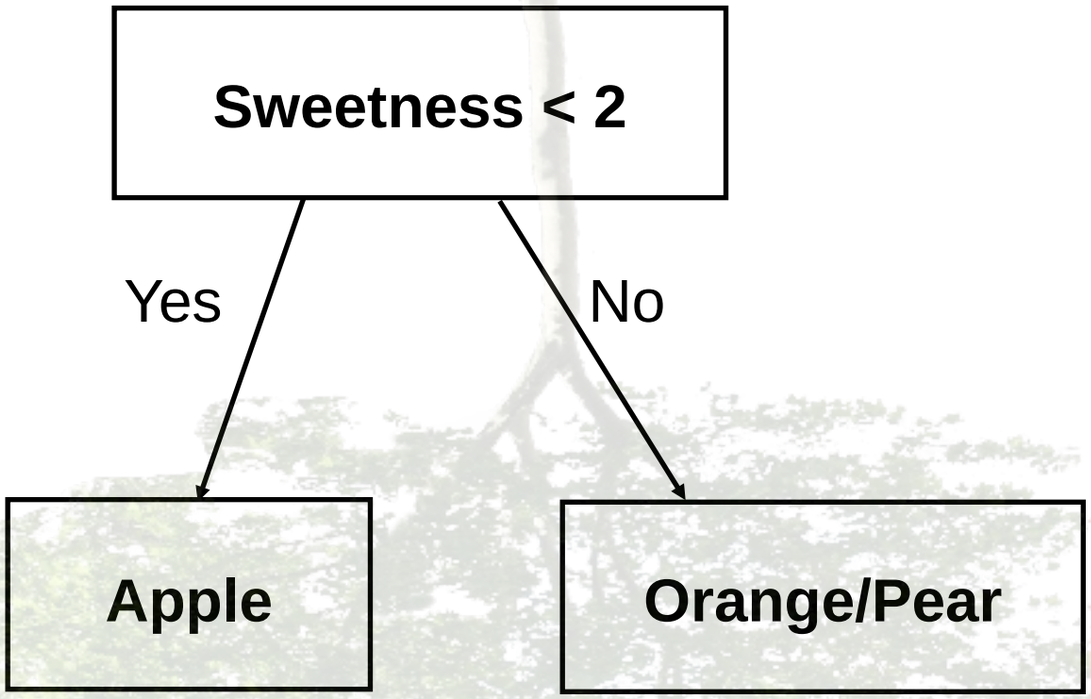
* Is the person ...

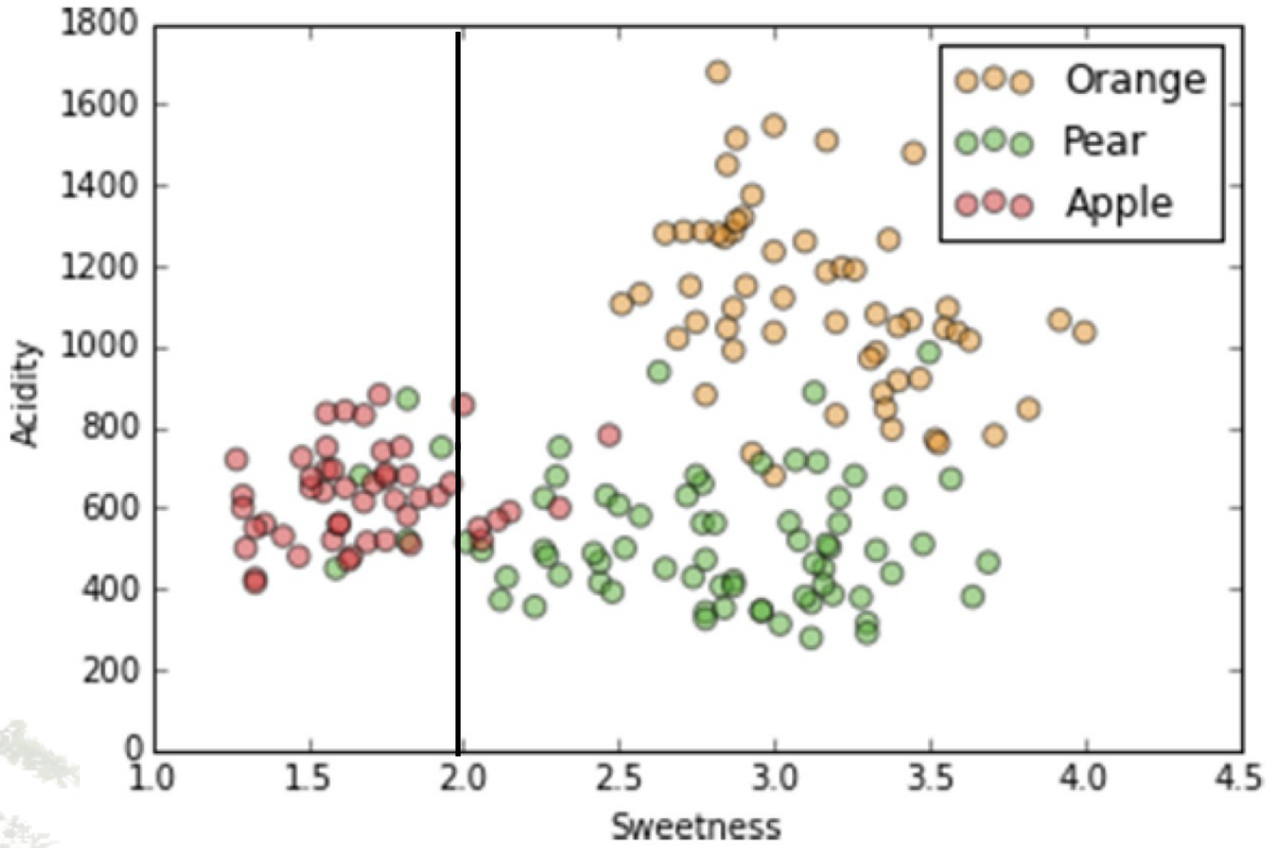
Example Decision Tree



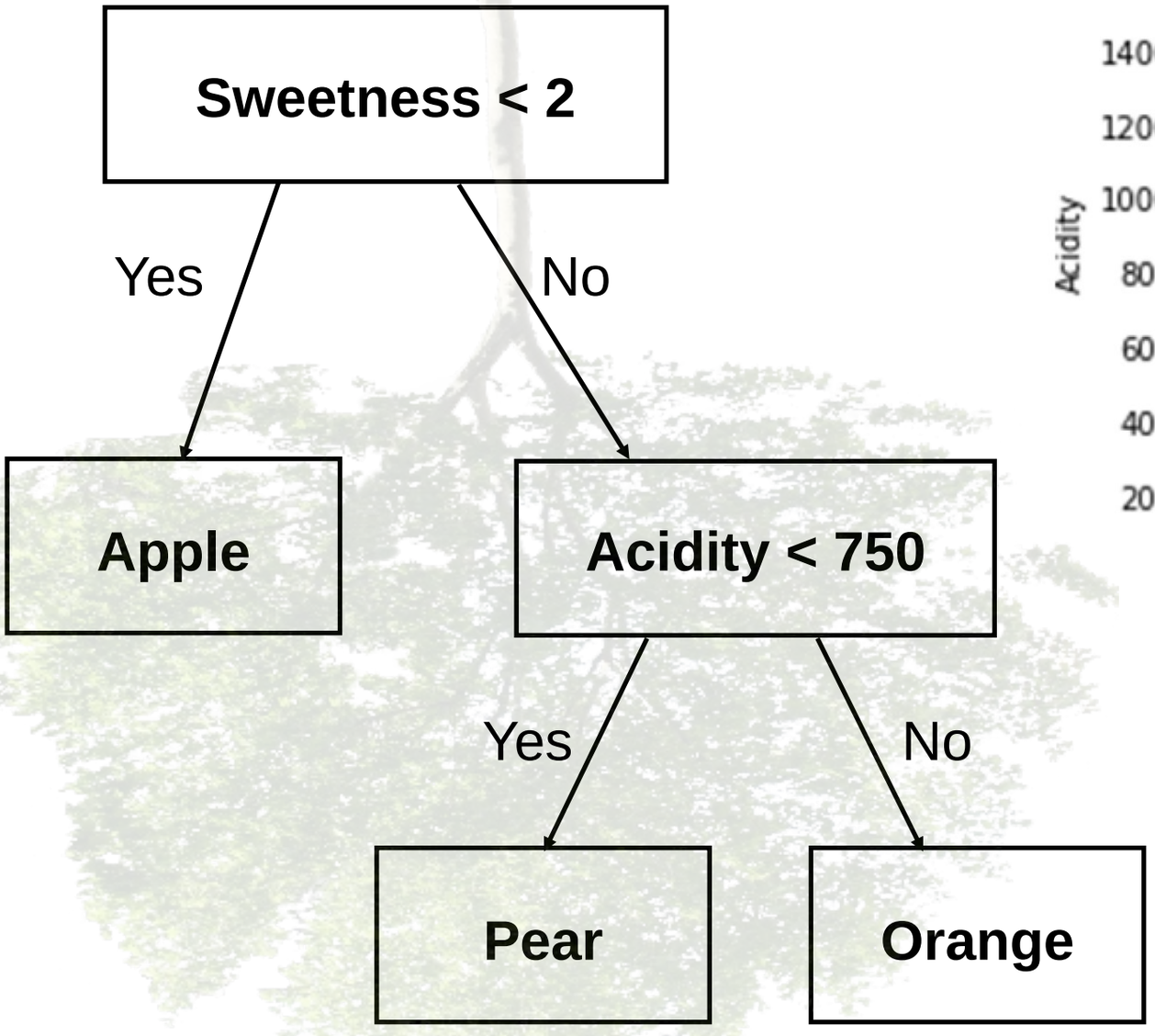


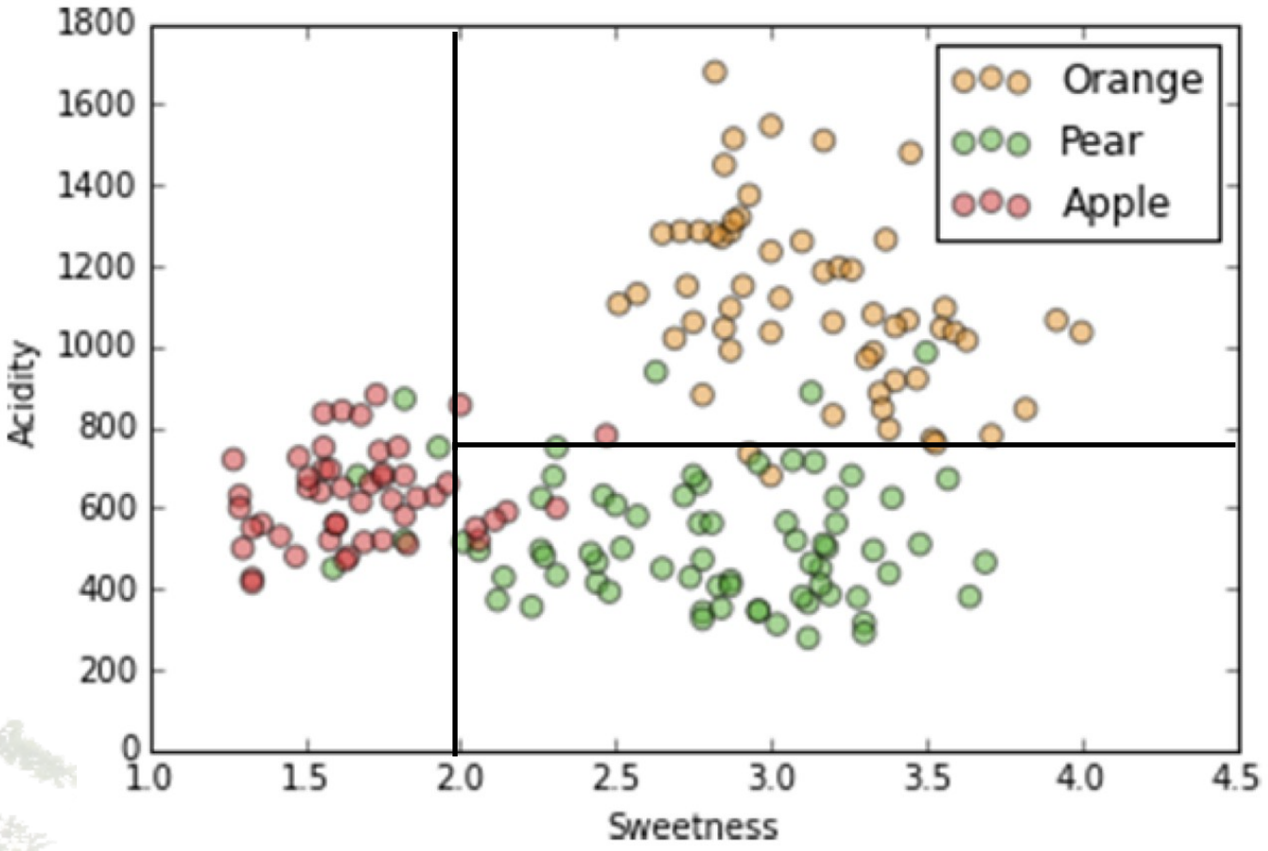
Example Decision Tree





Example Decision Tree





How to find the “right” question?

When to stop creating further nodes?

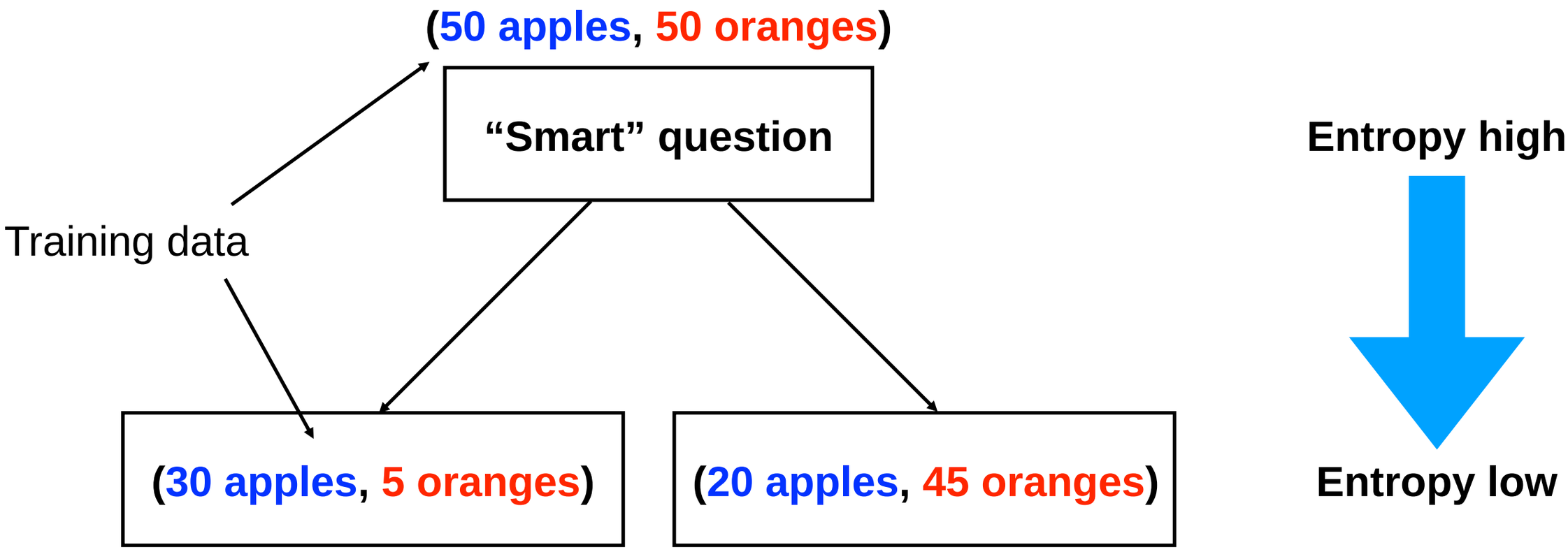
Overfitting/Underfitting?

Buzz word: "*information gain*"

High entropy

Low entropy

Buzz word: “*information gain*”



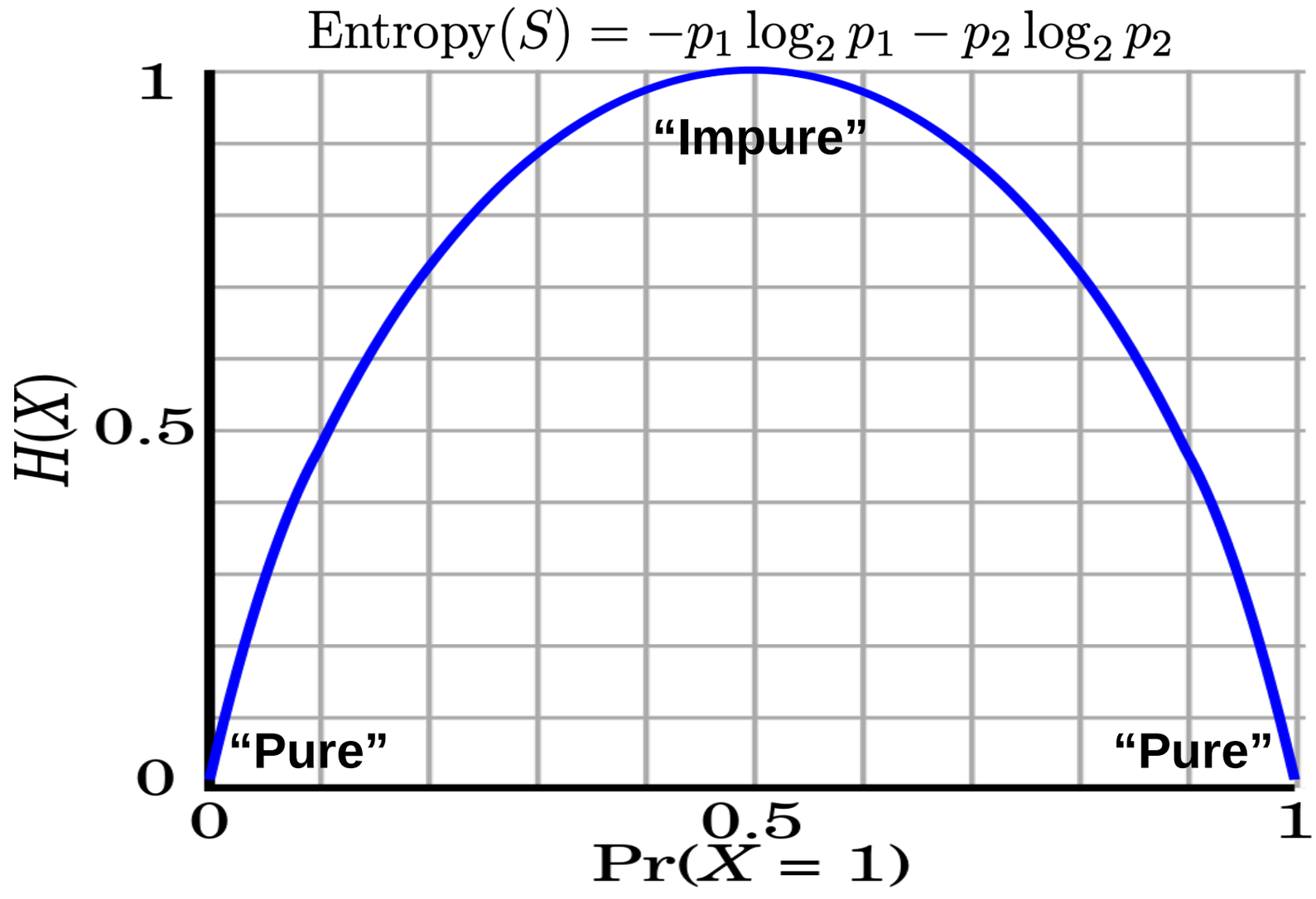
Information gain = Entropy_before - Entropy_after

Maximize!

Buzz word: “*information gain*”

(50 apples, 50 oranges)

Entropy = - p(apples) * $\log_2$ p(apples) - p(oranges) * $\log_2$ p(oranges)
= -0.5 * (-1) - 0.5 (-1) = 1 (maximum entropy)



Buzz word: “*information gain*”

(50 apples, 50 oranges)

“Smart” question

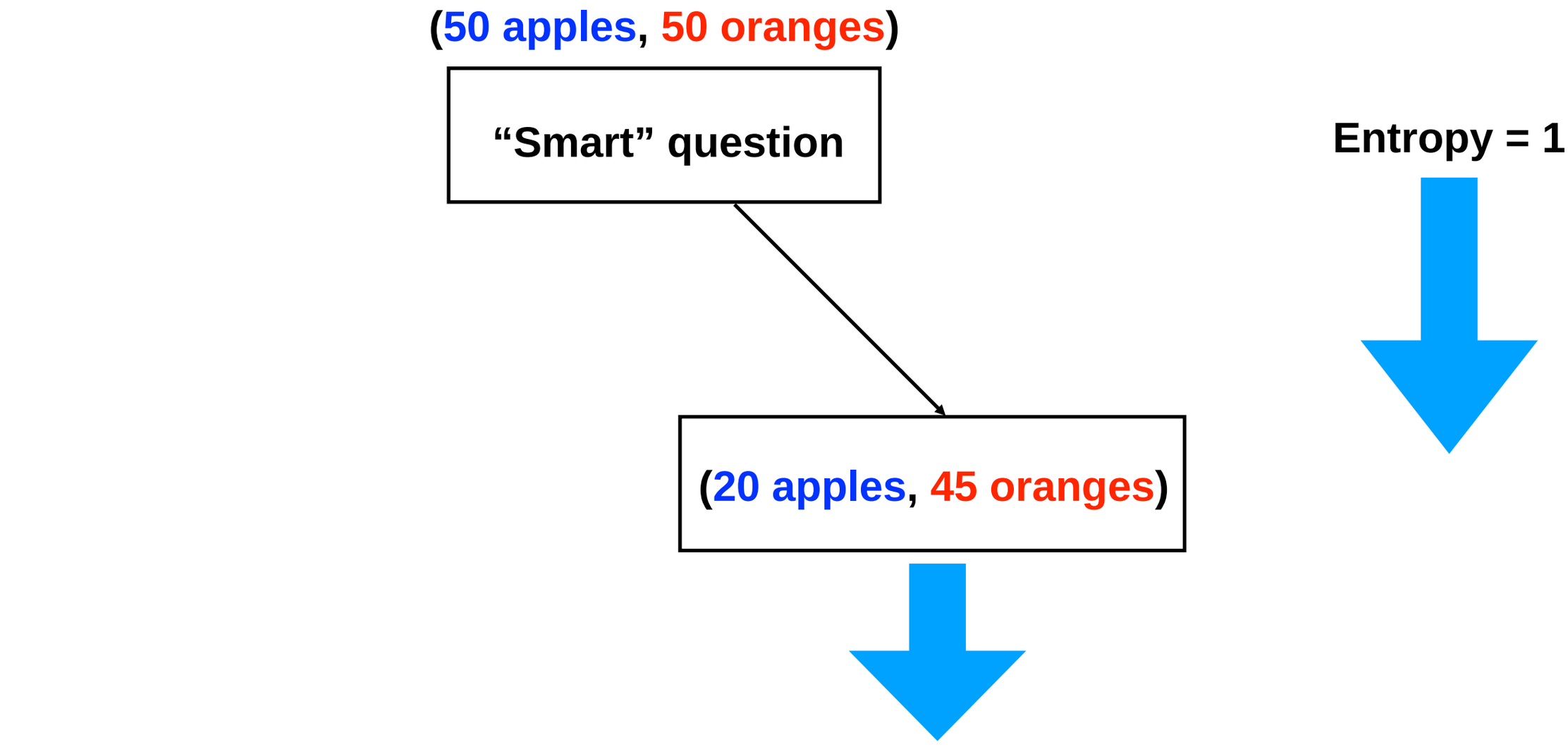
Entropy = 1

(30 apples, 5 oranges)

Entropy = - p(apples) * $\log_2$ p(apples) - p(oranges) * $\log_2$ p(oranges)

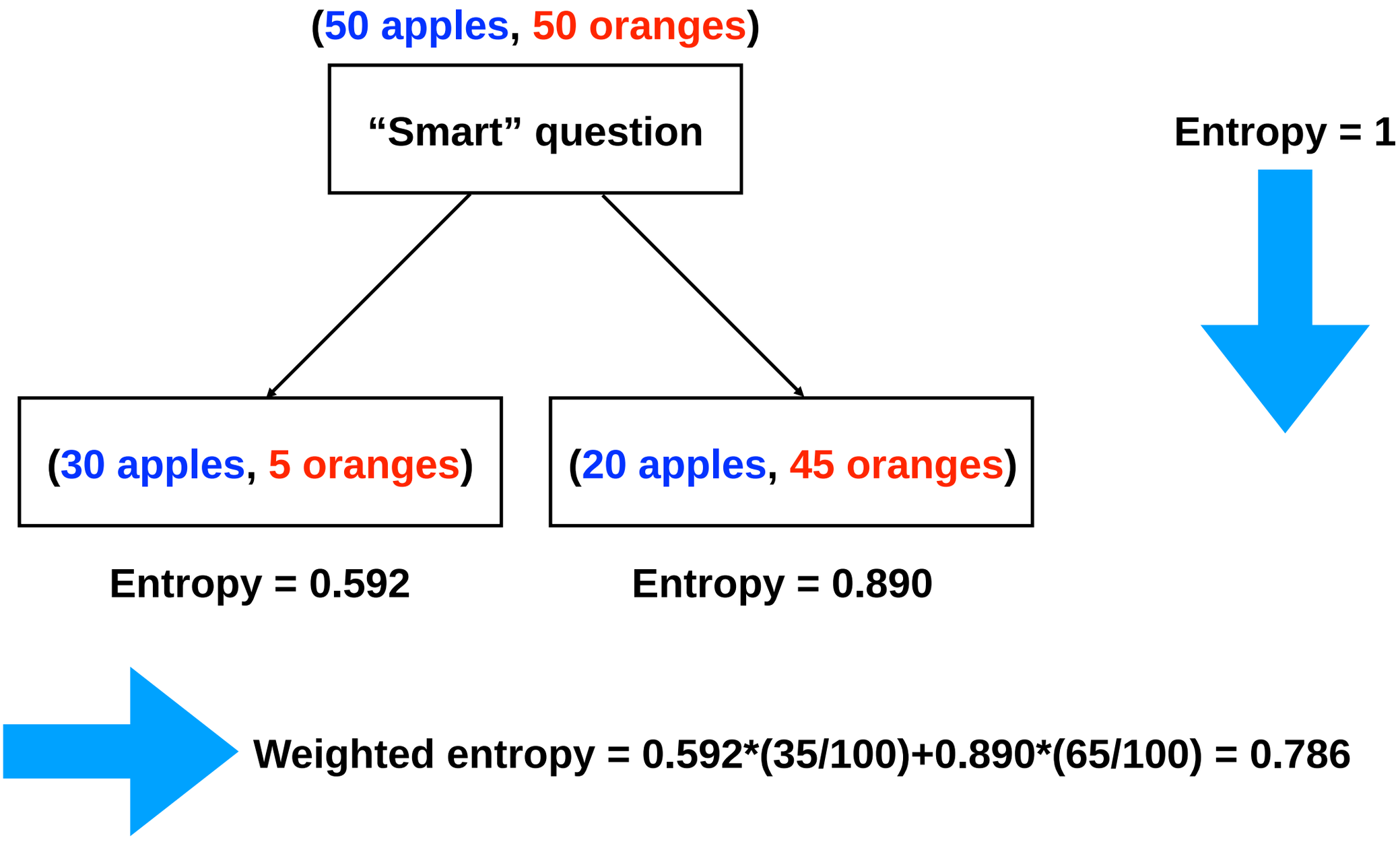
= -30/35 * $\log_2$ (30/35) - 5/35 * $\log_2$ (5/35) = 0.592

Buzz word: “*information gain*”

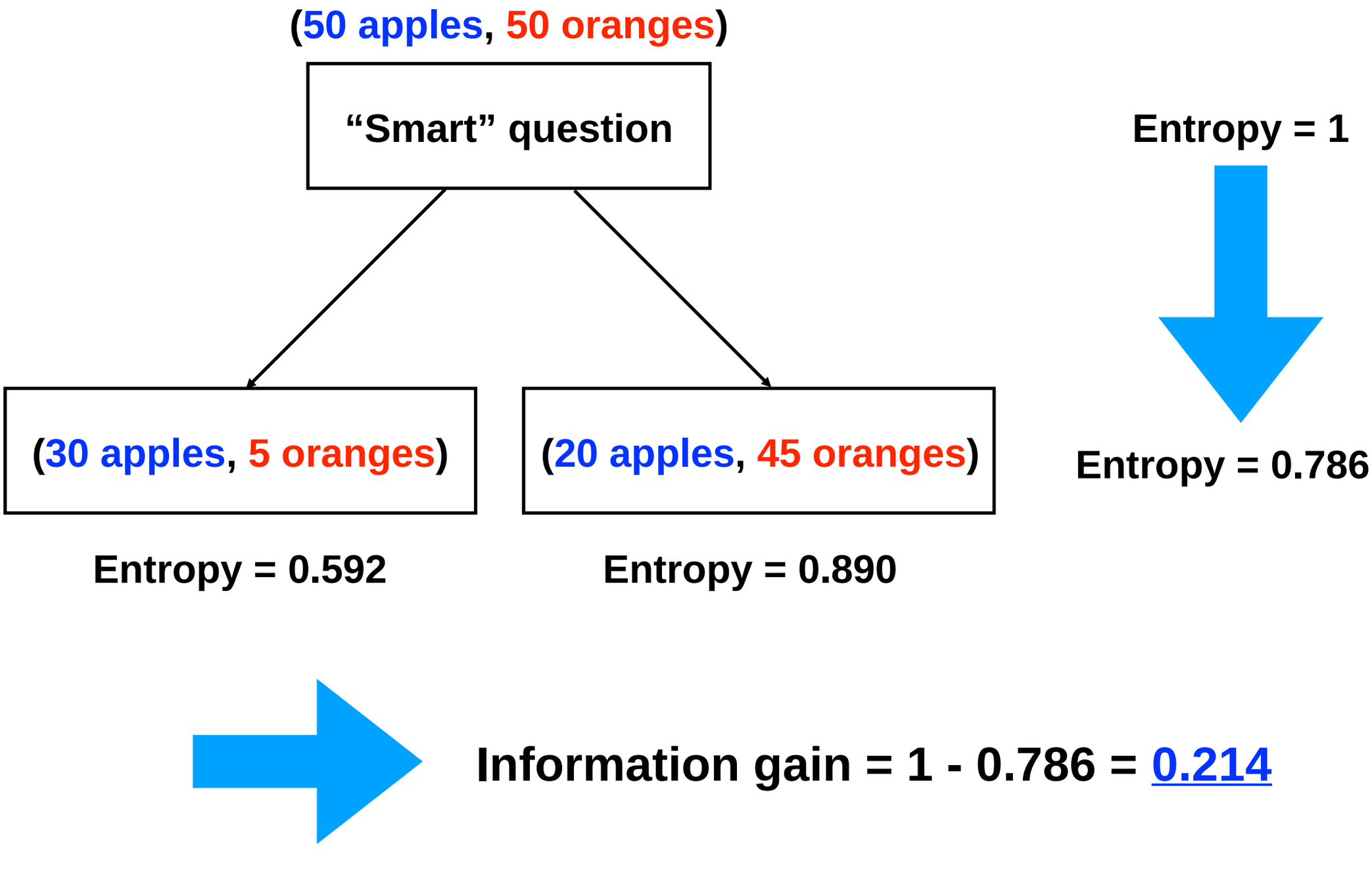


Entropy = - p(apples) * $\log_2$ p(apples) - p(oranges) * $\log_2$ p(oranges)
= -20/65 * $\log_2$ (20/65) - 45/65 * $\log_2$ (45/65) = 0.890

Buzz word: “*information gain*”

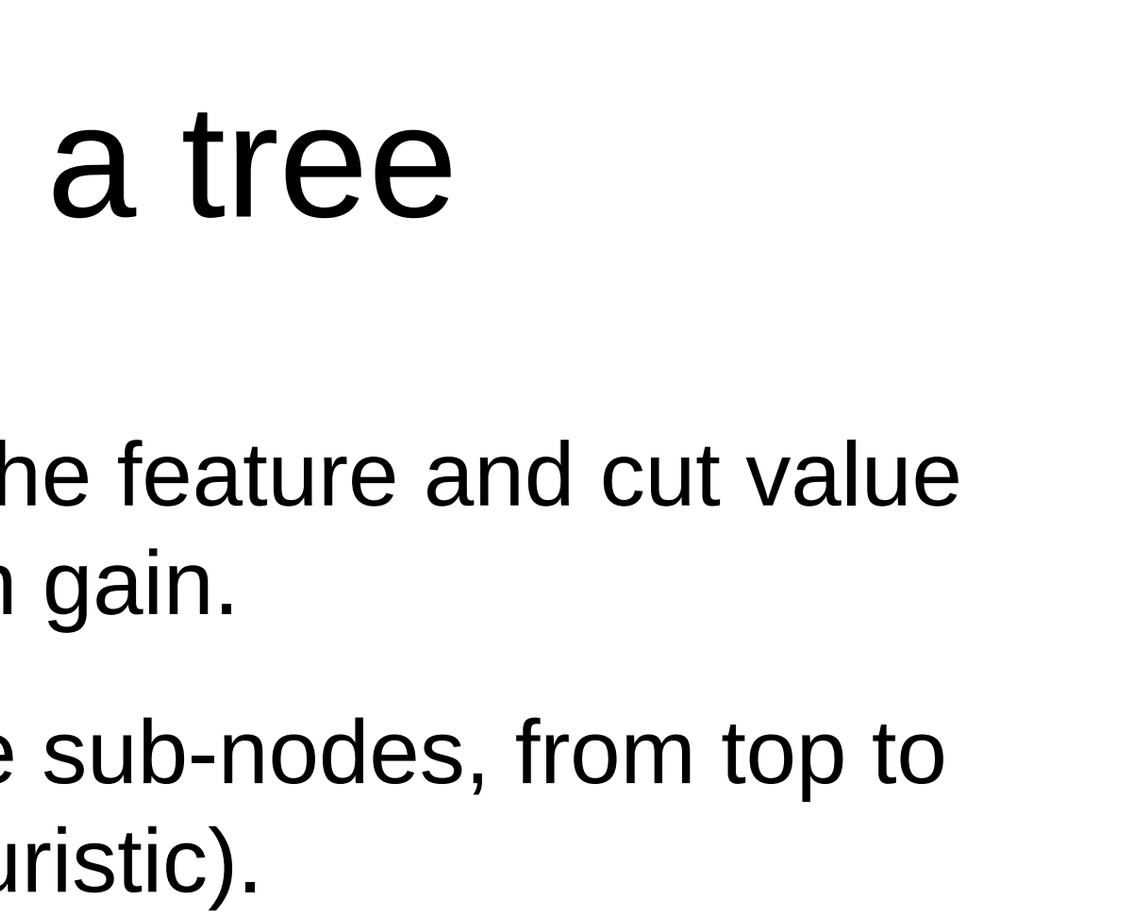


Buzz word: “*information gain*”



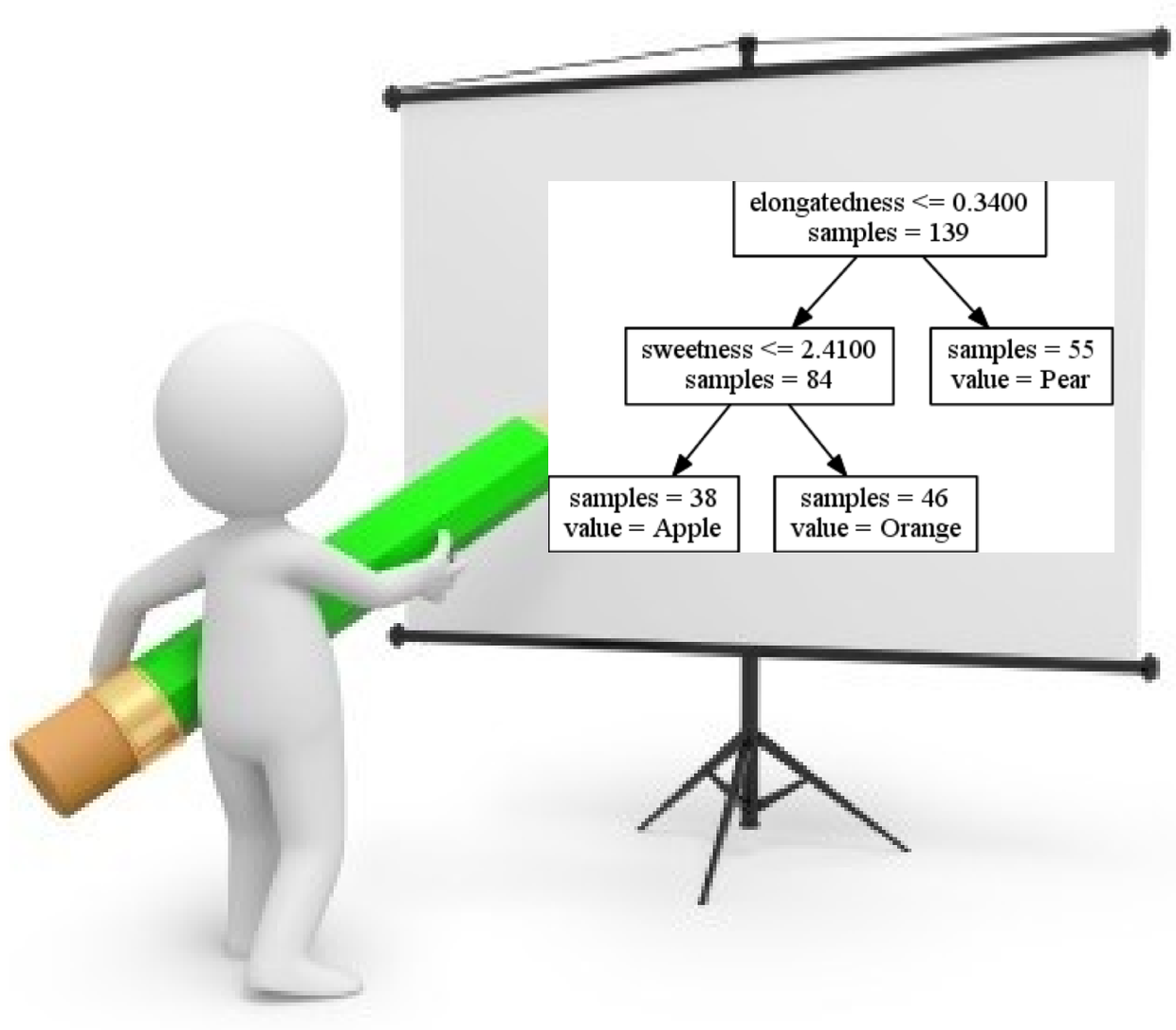
Building a tree

- Start with top node, find the feature and cut value giving highest information gain.
- Proceed similarly with the sub-nodes, from top to bottom (a.k.a. greedy heuristic).
- Stop growing tree when a minimum information gain is reached.
- End point is called a leaf.



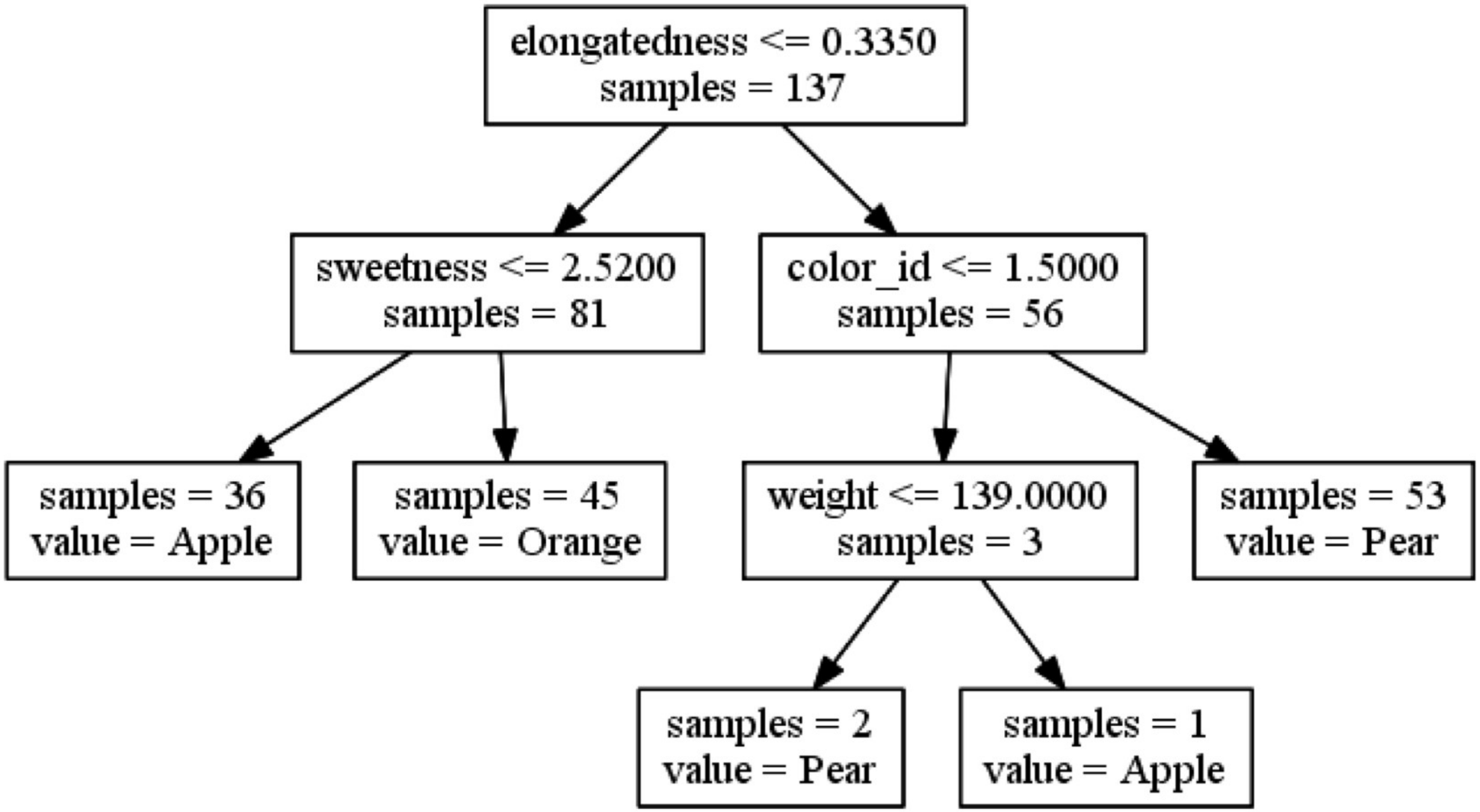
Tuneable, basically chosen such to avoid under/overfitting;
Different strategies and alternatives possible….

Demo…


elongatedness <= 0.3400
samples = 139
sweetness <= 2.4100
samples = 84
samples = 55
value = Pear
samples = 38
value = Apple
samples = 46
value = Orange

REPLICATE 1:

Decision tree built on a randomly chosen 70% training set

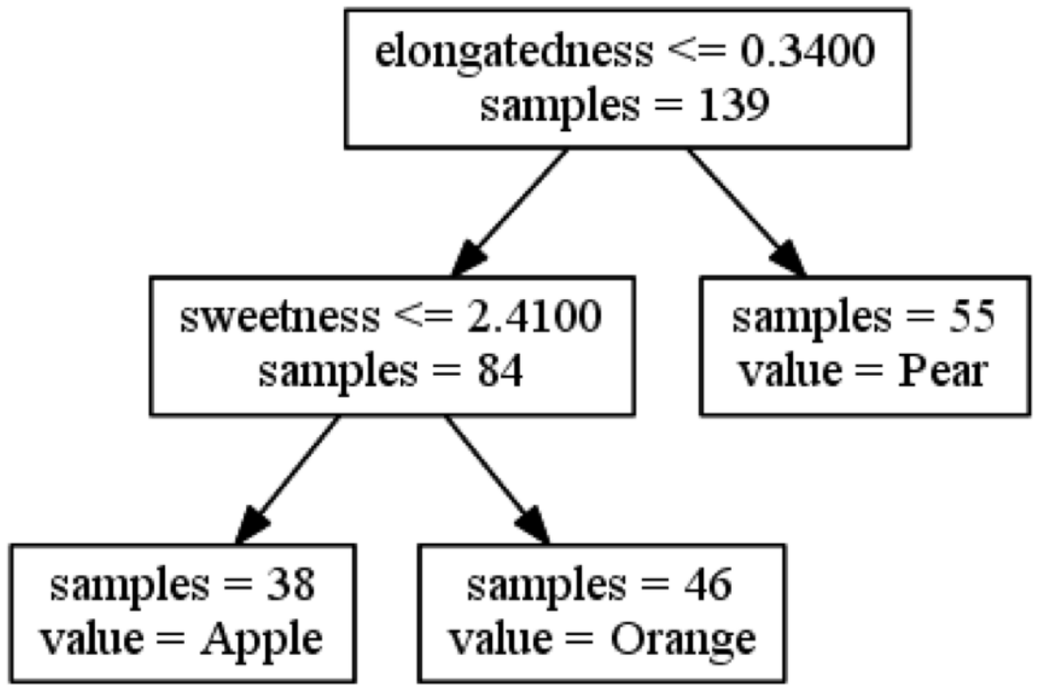


Note: Not all features used (acidity missing)

The last branch is only splitting 3 of 179 instances. Is it overfitting?

REPLICATE 2:

Decision tree built on another randomly chosen 70% training set



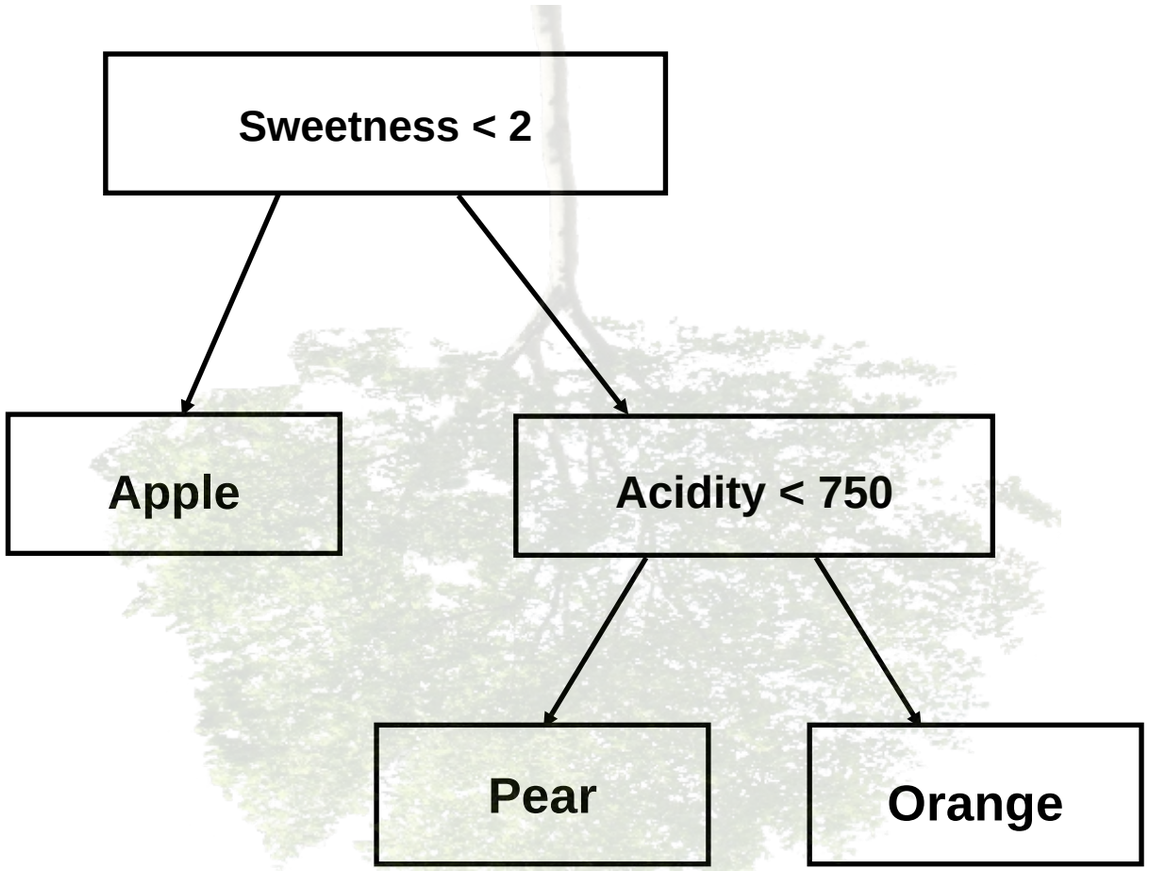
This time, only two features were used; this probably means the 70% randomly chosen for the training set did not have enough variation in the other features to justify a split.

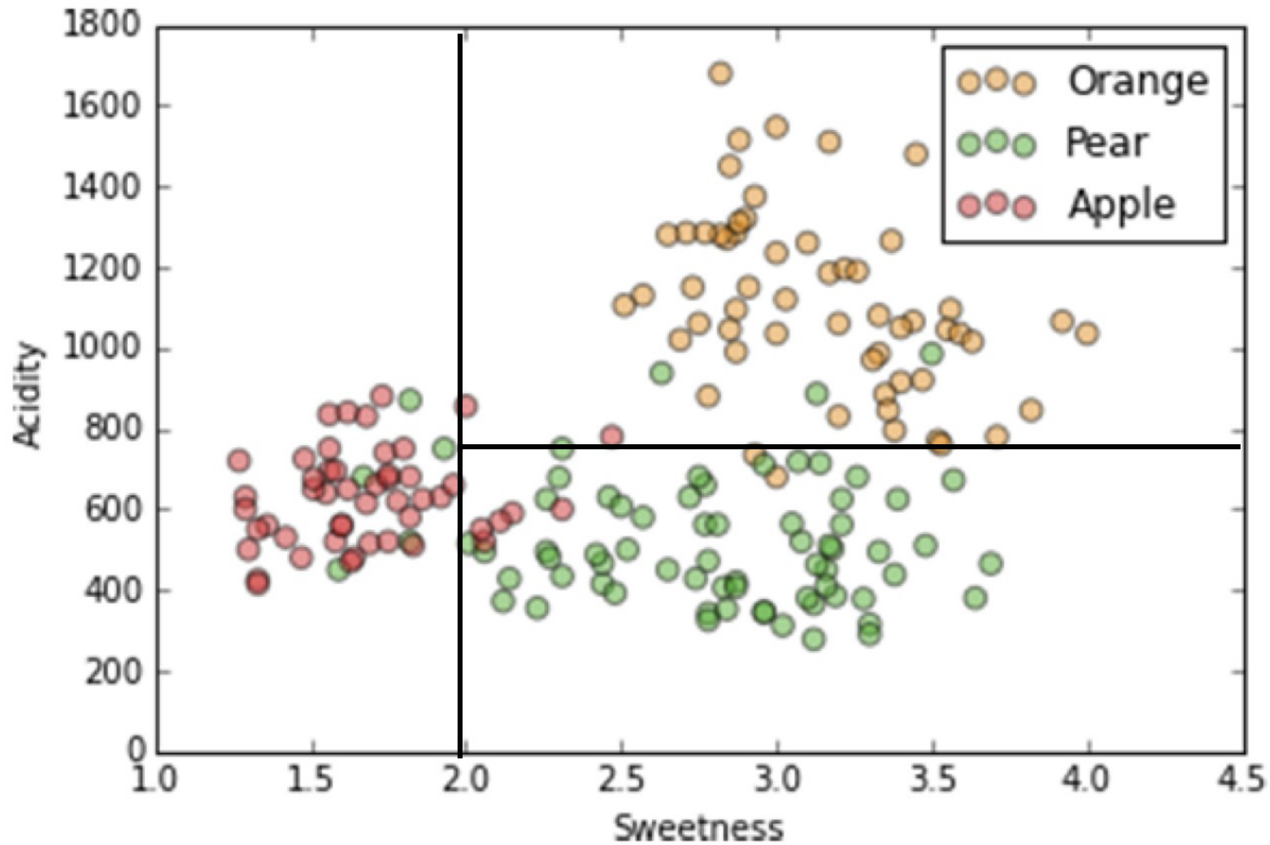
We got different models with different training sets. That tells us the fit is probably off, but it doesn't tell us if either of our models is better than the other, i.e. if the more complex one is overfitting or the simpler one is underfitting.

Decision Trees

Pros:

- Easy to interpret, visualise, and understand
- Allows mixture of categorical and numerical features
- Sorts out itself which features are relevant
- Fast classification, trickles down the tree

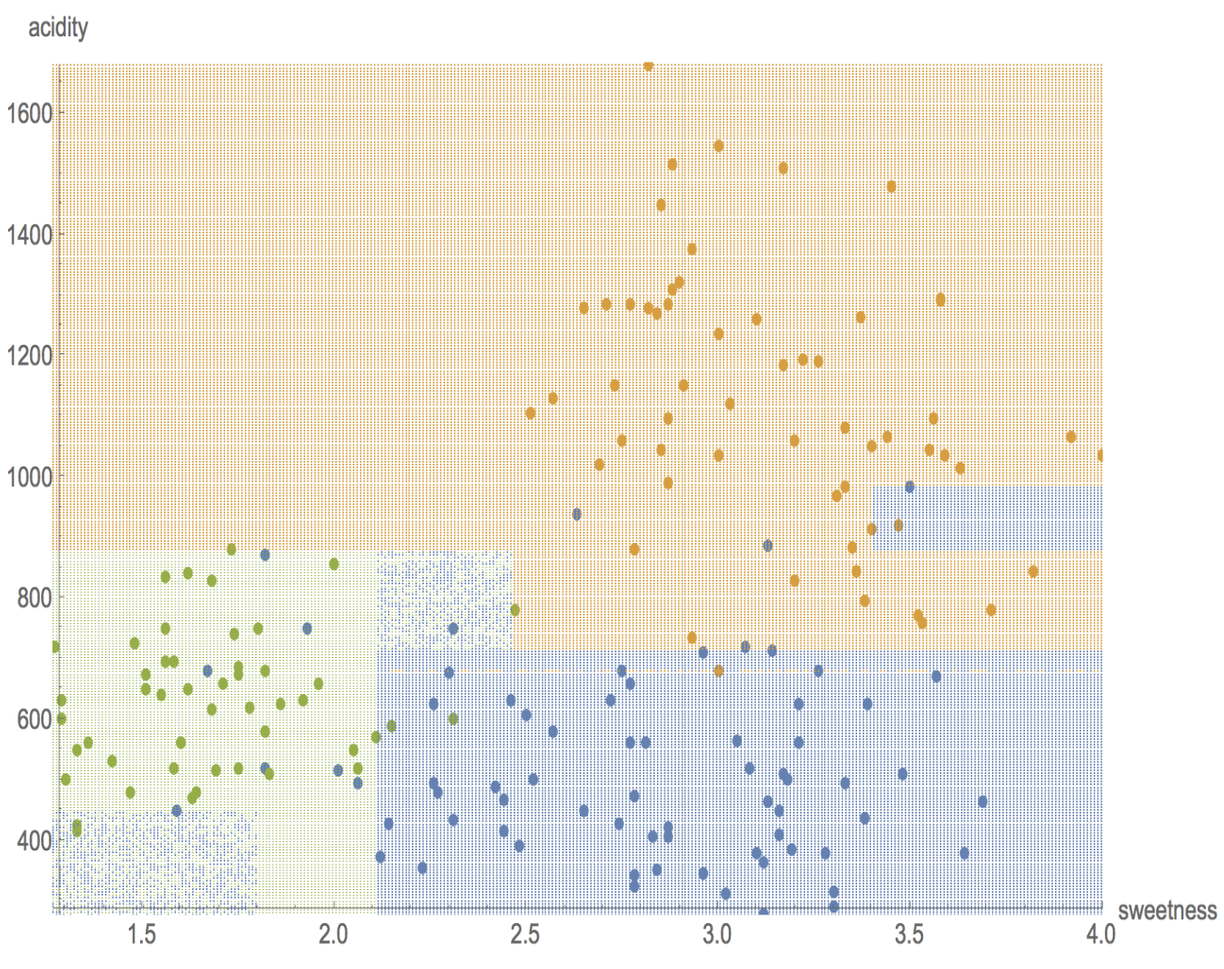


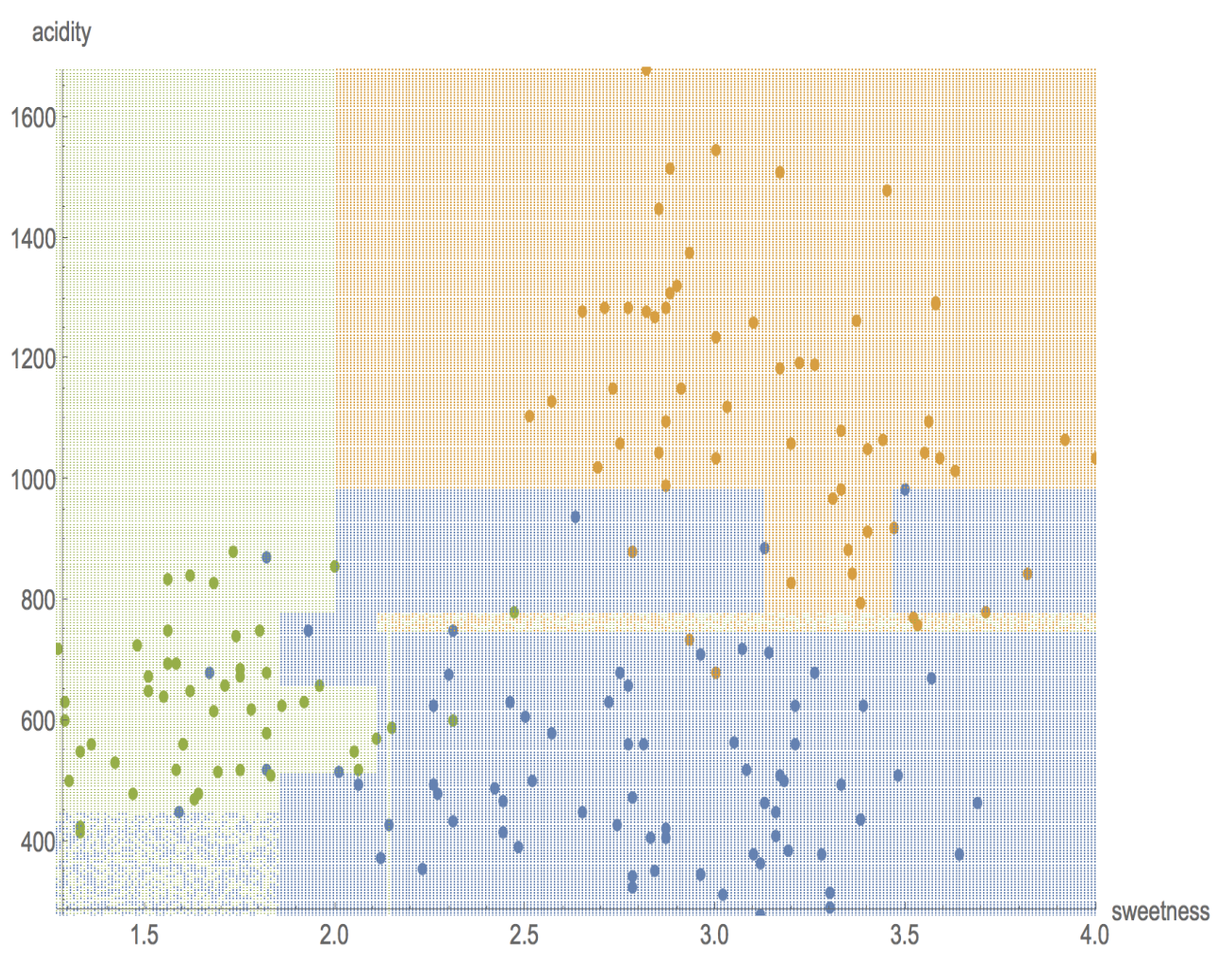


Decision Trees

Cons:

- Not stable: strong dependence on training set (variance)
- Very sensitive to overfitting or underfitting
- Tree can become rather complex
- No guarantee to *globally* optimise the tree (greedy)

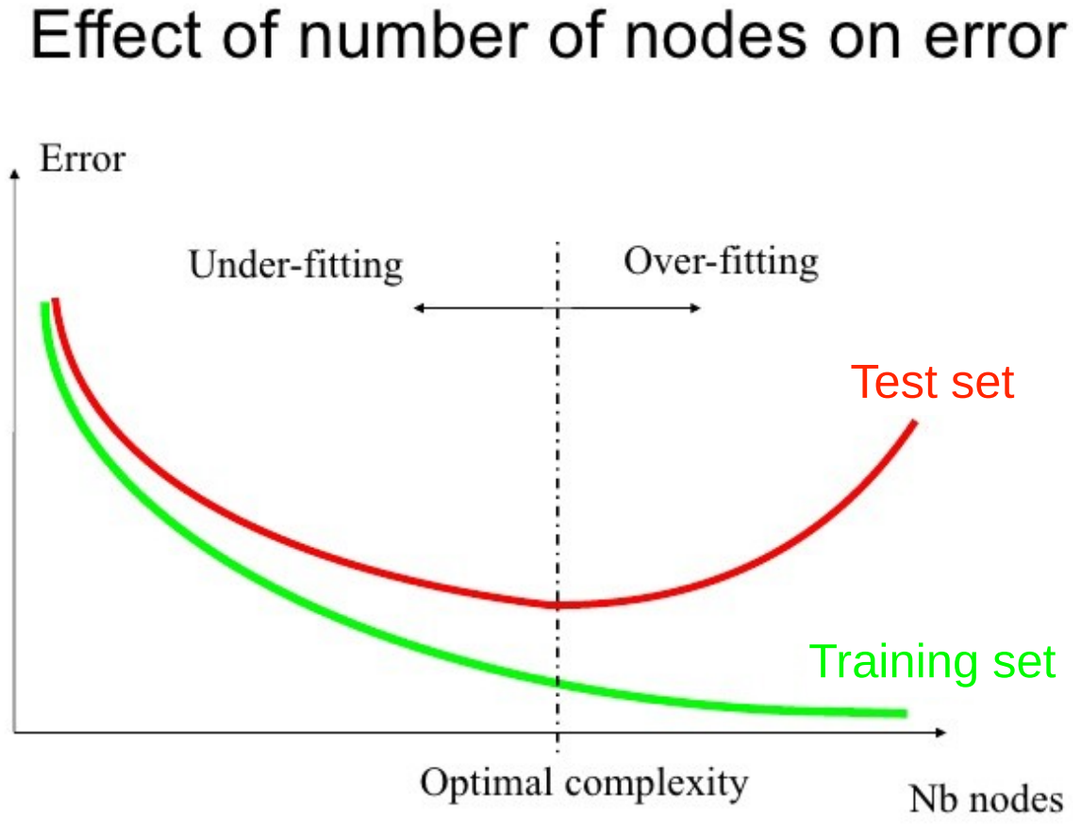




Decision Trees

Tackle the “cons”?

- Pruning
- Ensemble methods: “*Random Forest*”



Decision Trees

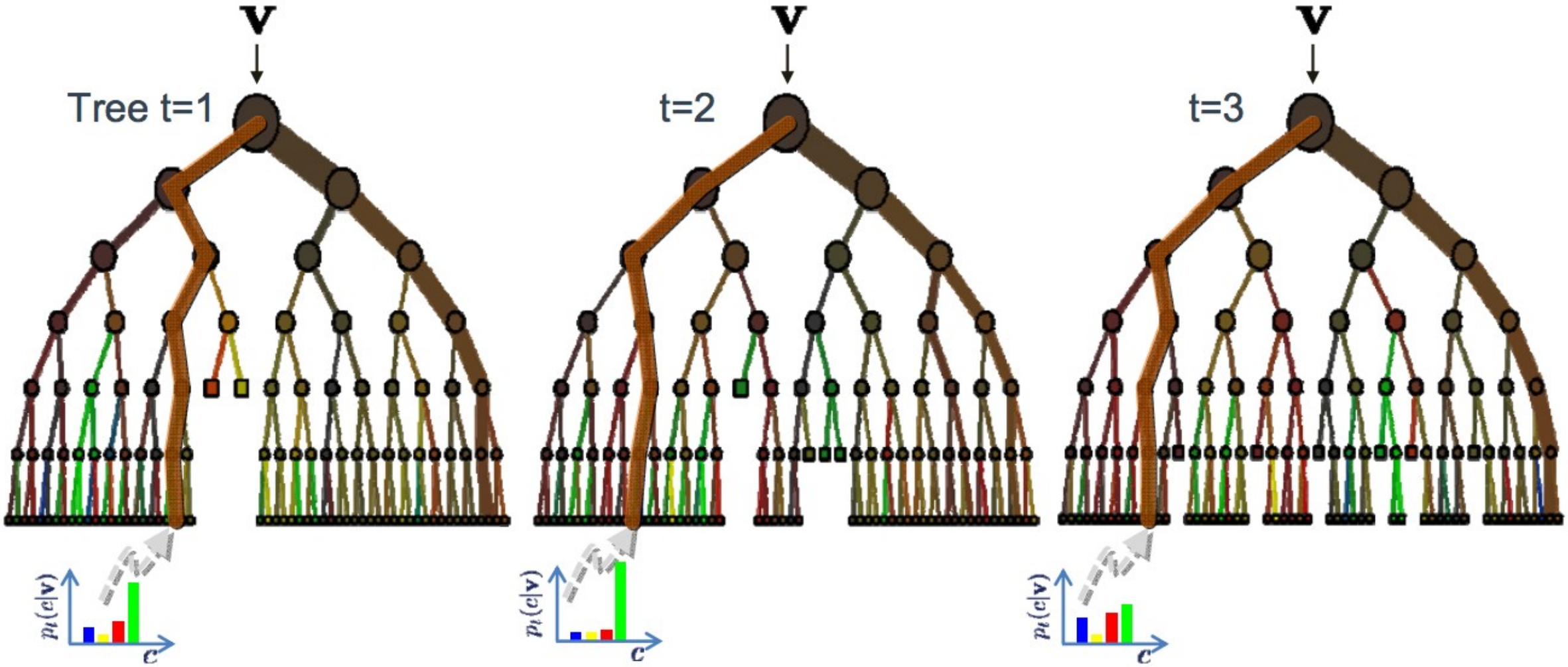
Tackle the “cons”?

- Pruning
- Ensemble methods: “*Random Forest*”

Random Forest - the algorithm

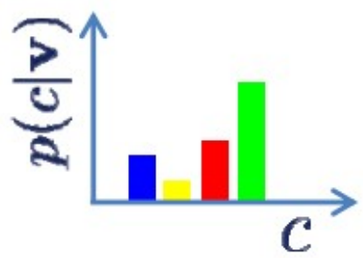
- Create an ensemble of unpruned trees, can be many!!
- Each tree is trained with a randomly selected part of the total training sample (randomising 1).
- For each node of a tree, select randomly a subset of features; pick the best variable and splitting criteria; split it into two daughter nodes (randomising 2).
- Classify an instance by averaging over the output of all the trees.
- Note: the cross validation is done by the algorithm itself!

Random Forest

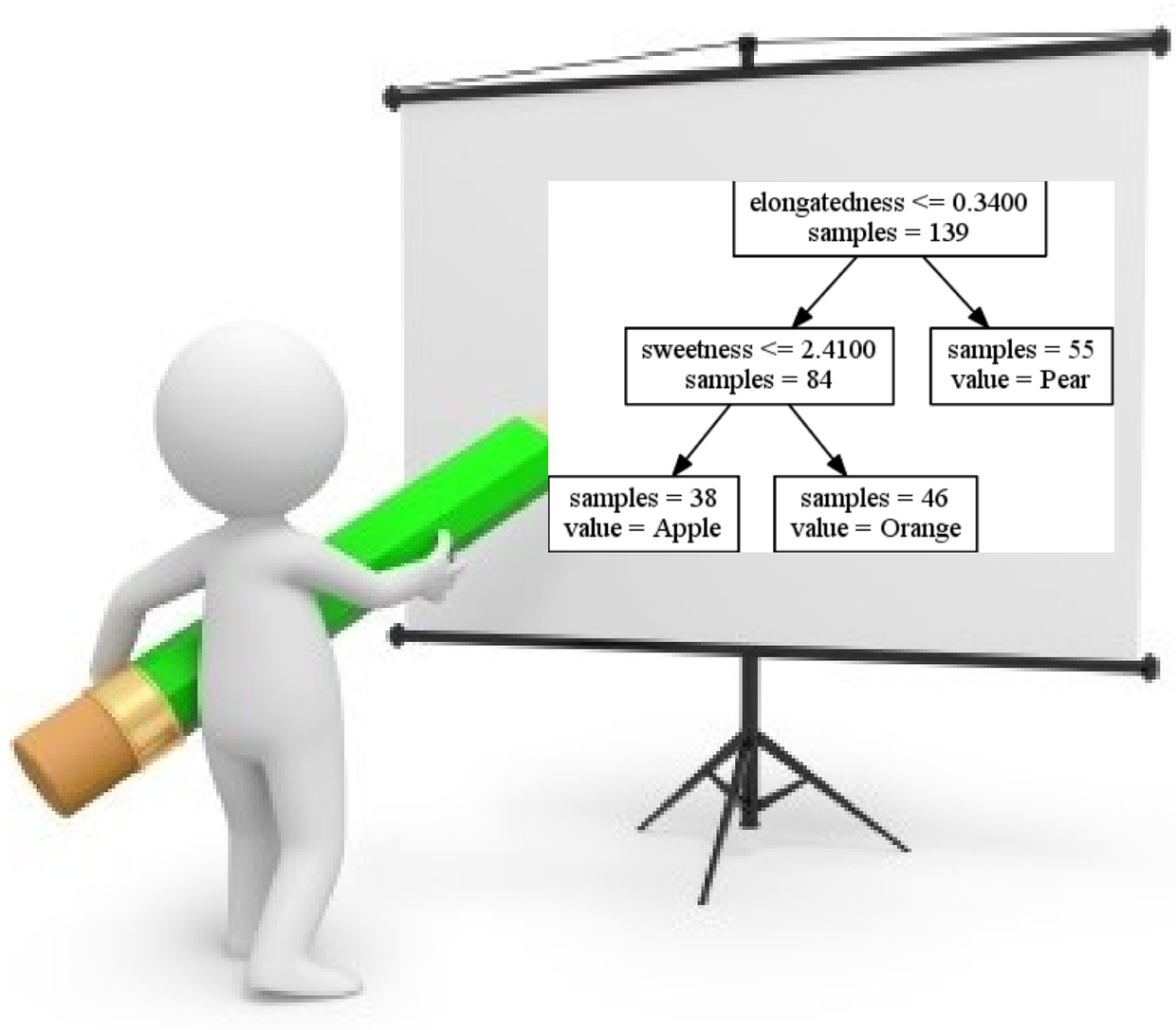


The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$

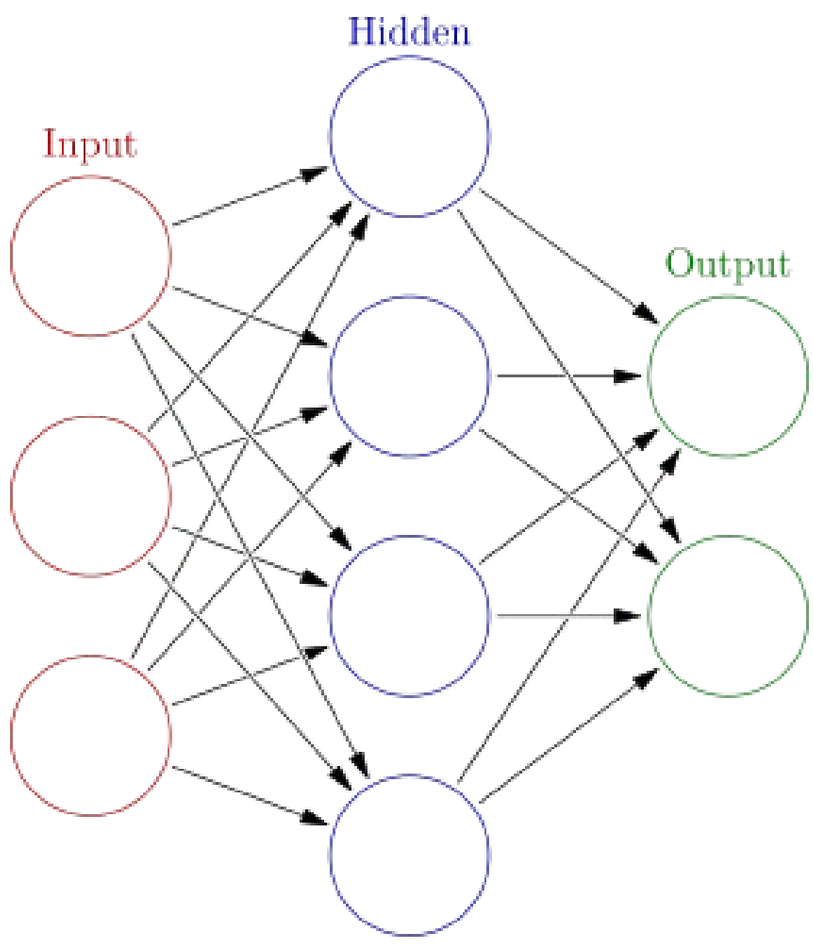


Demo…


elongatedness <= 0.3400
samples = 139
sweetness <= 2.4100
samples = 84
samples = 55
value = Pear
samples = 38
value = Apple
samples = 46
value = Orange

Artificial Neural Networks

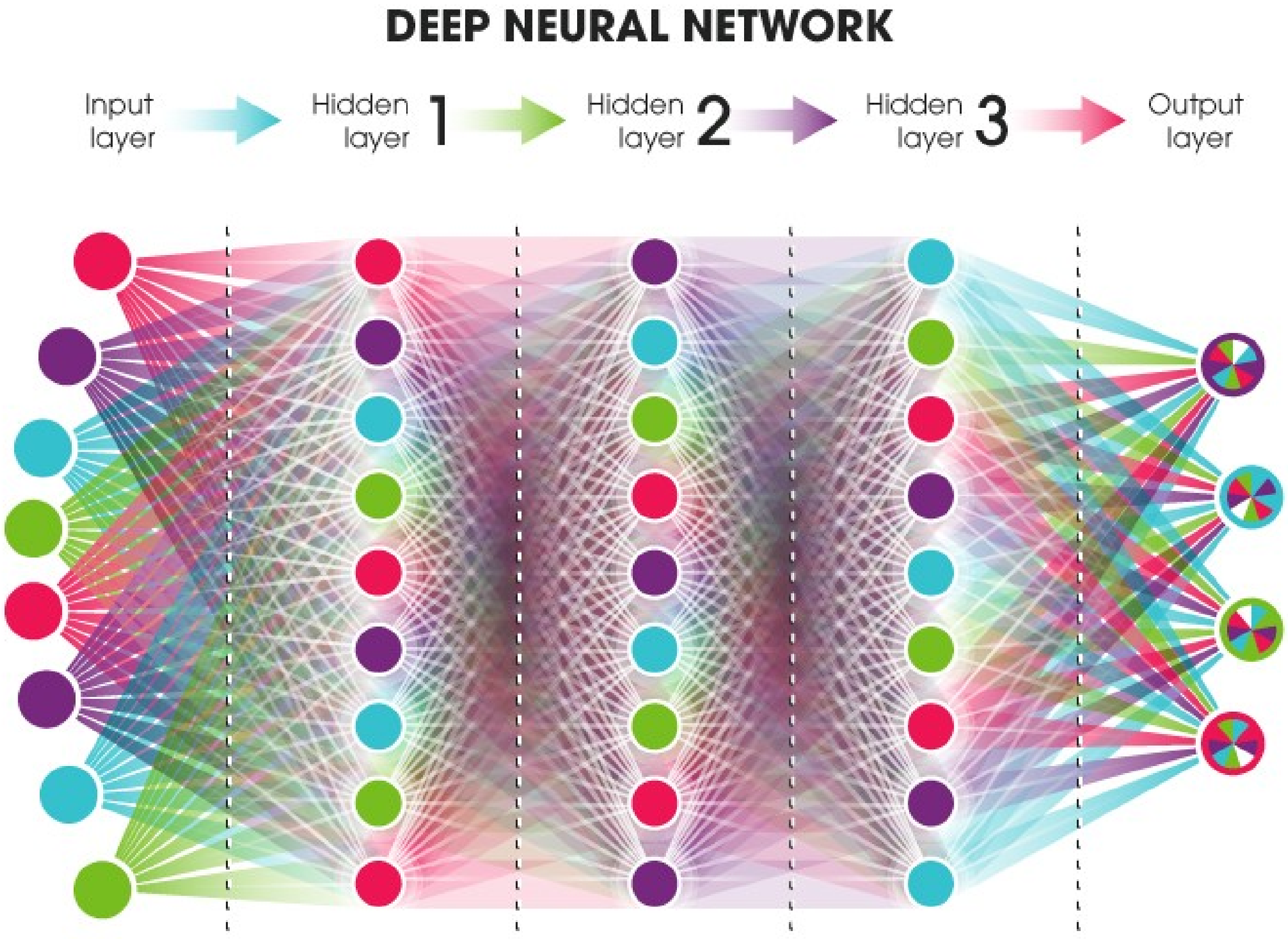
- Supervised ML, finds non-linear boundaries (similar to kNN and DT)
- Popular because of analog with human brain.
- One of the oldest (40's) "ML" algorithm on the market.
- Revived again recently because of big data challenges, leading to buzzword "*deep learning*".



Multi-Layer Perceptron (MLP)

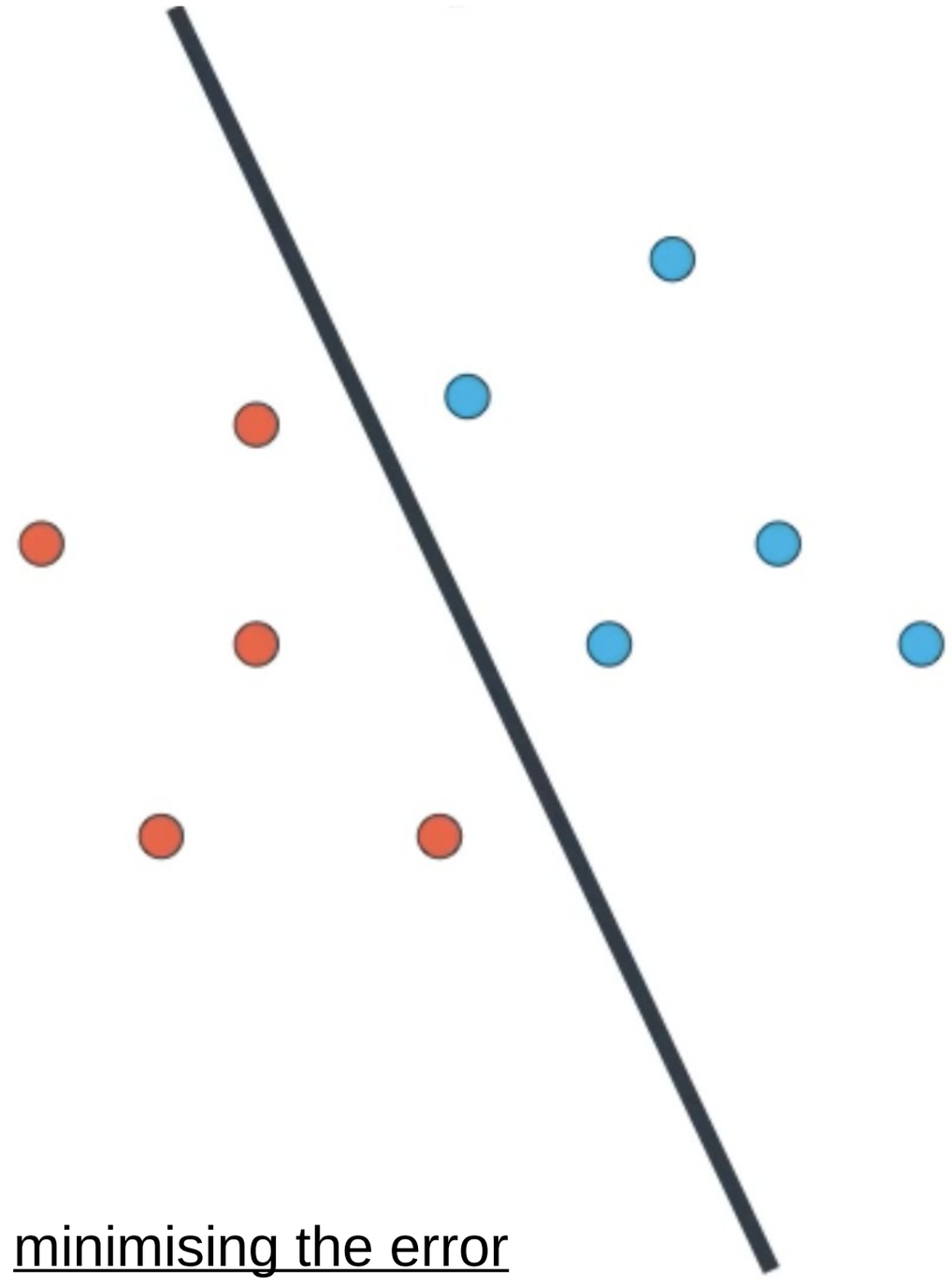
Note: following slides are taken from a talk of Luis Serrano

Neural networks can be intimidating!



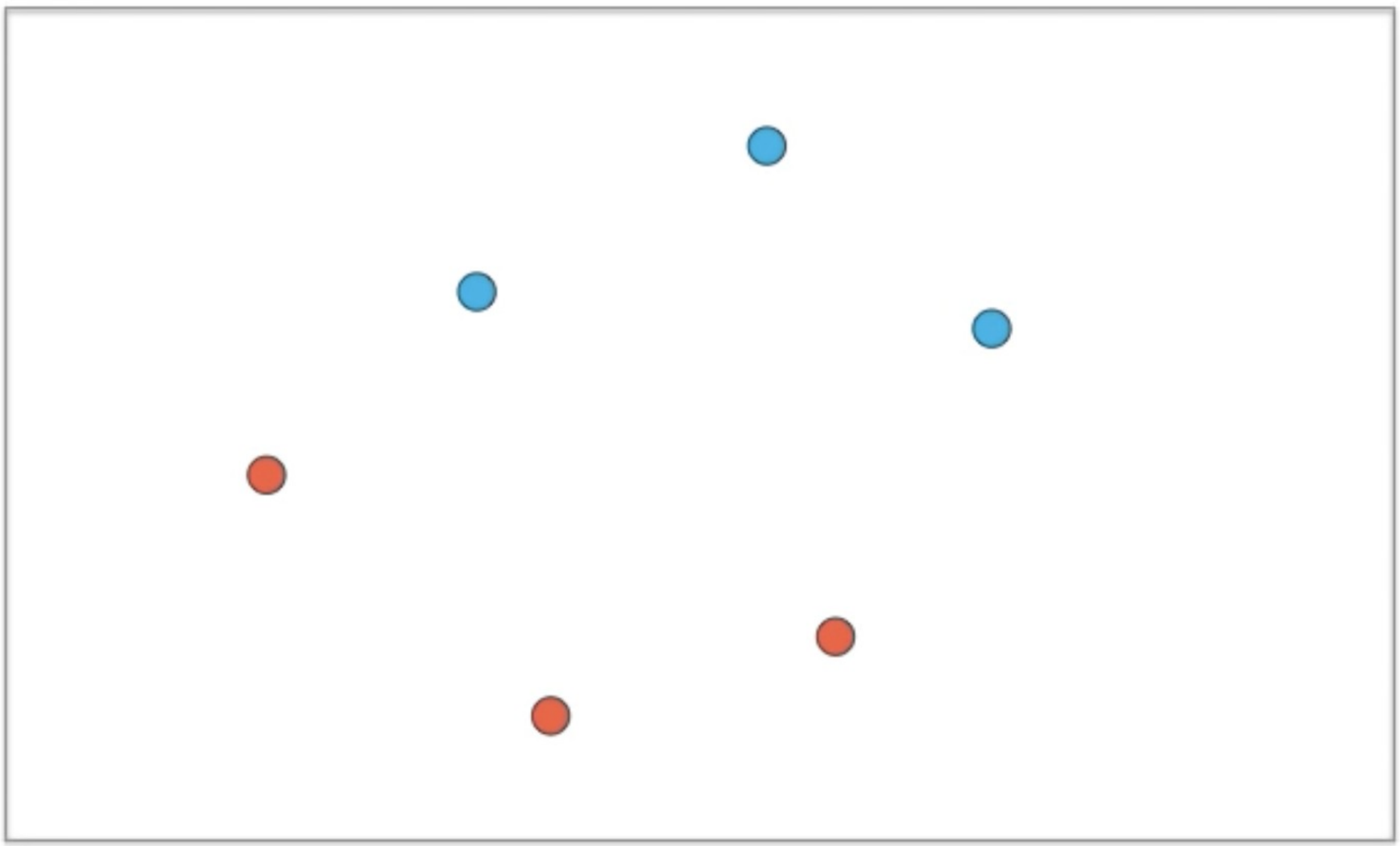
neuralnetworksanddeeplearning.com - Michael Nielsen, Yoshua Bengio, Ian Goodfellow, and Aaron Courville, 2016.

What is ML all about?

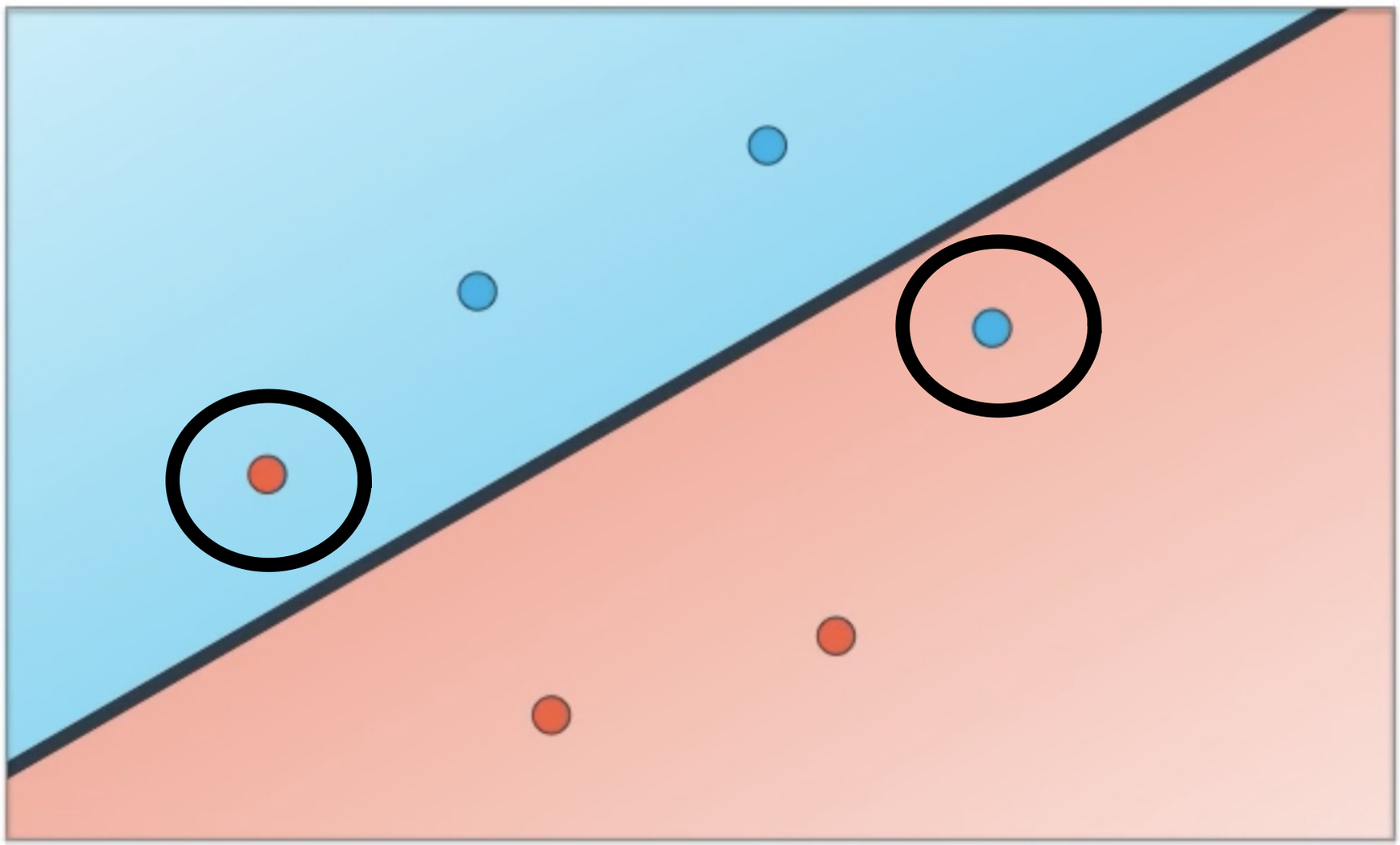


ML: split the data by minimising the error

Split the data!

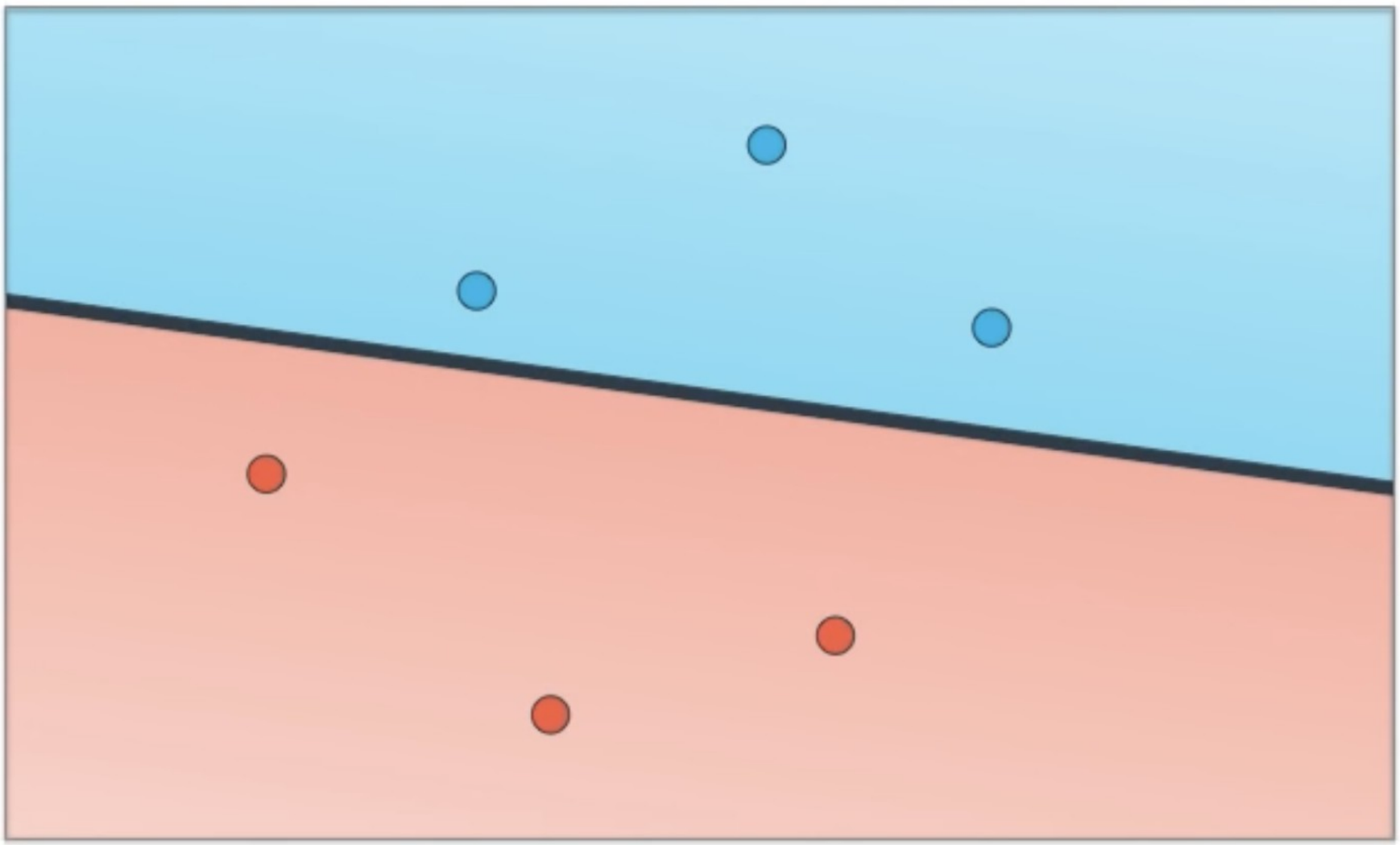


Split the data!



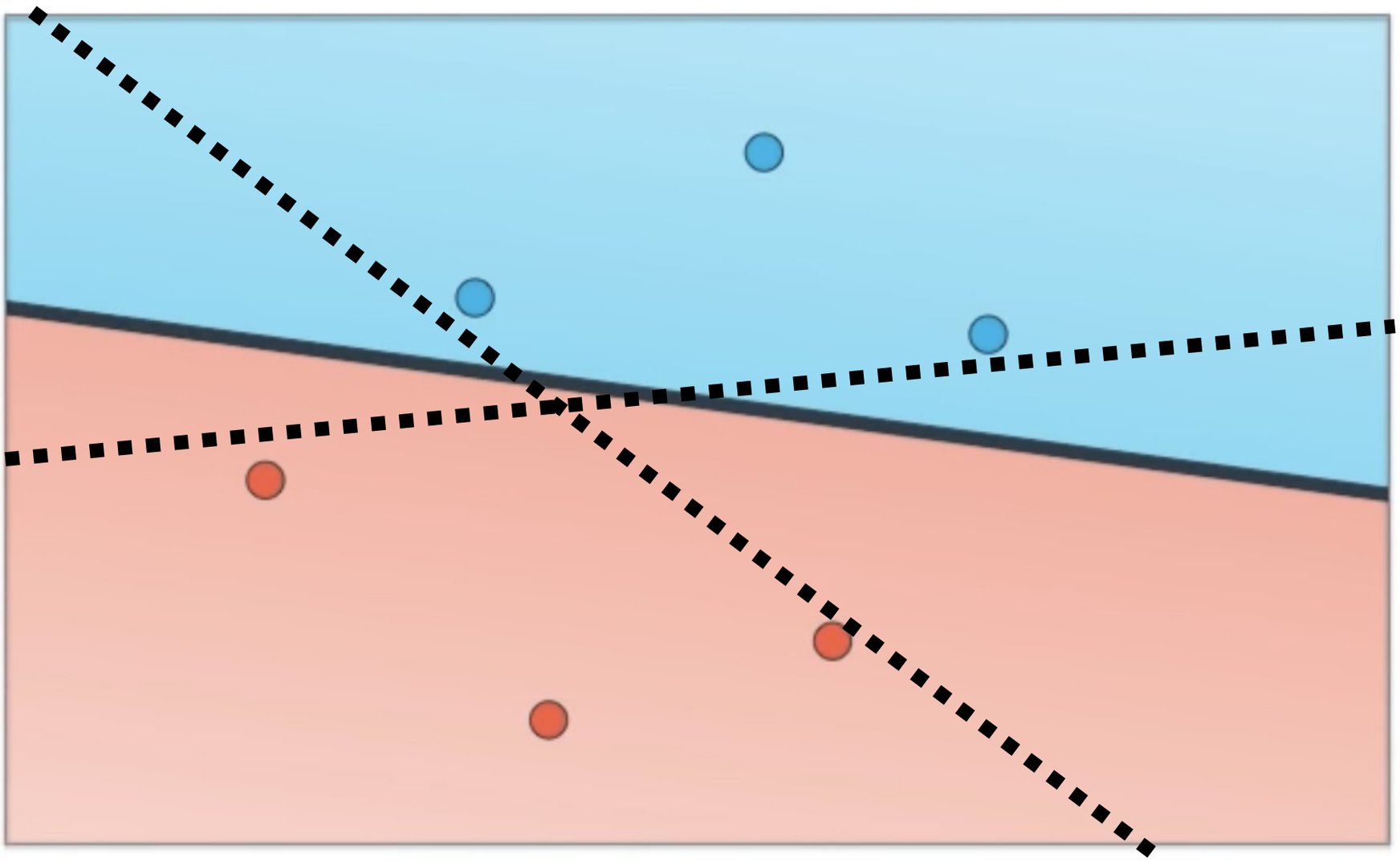
2 errors ;(

Split the data!



0 errors ;)

Split the data!

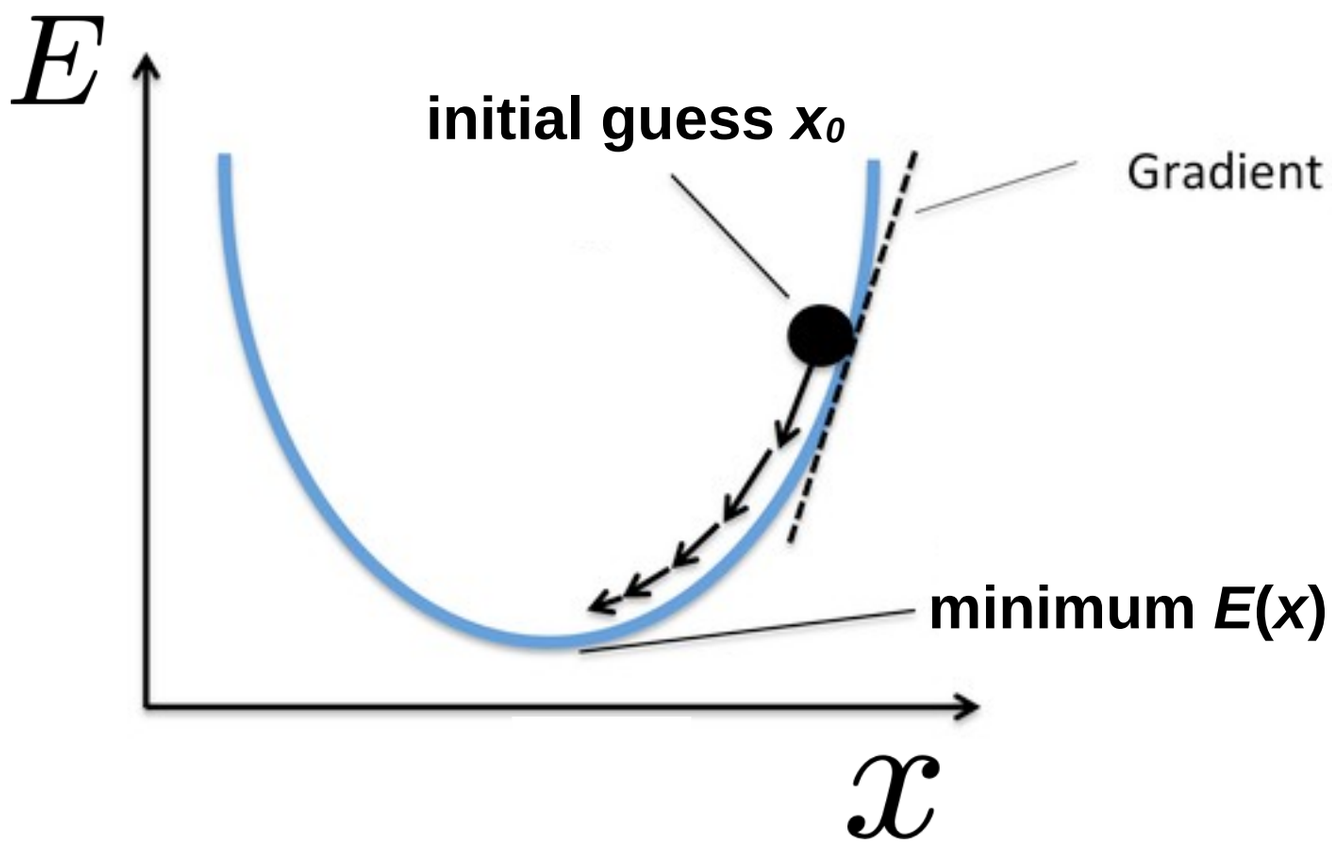


0 errors ;)

The number of errors is a discrete function.

Better to have a <u>continuous error function</u> in order to calculate derivatives etc.
(to evaluate gradient descent)

Gradient descent



$$x_{i+1} = x_i - \eta \nabla E(x)$$

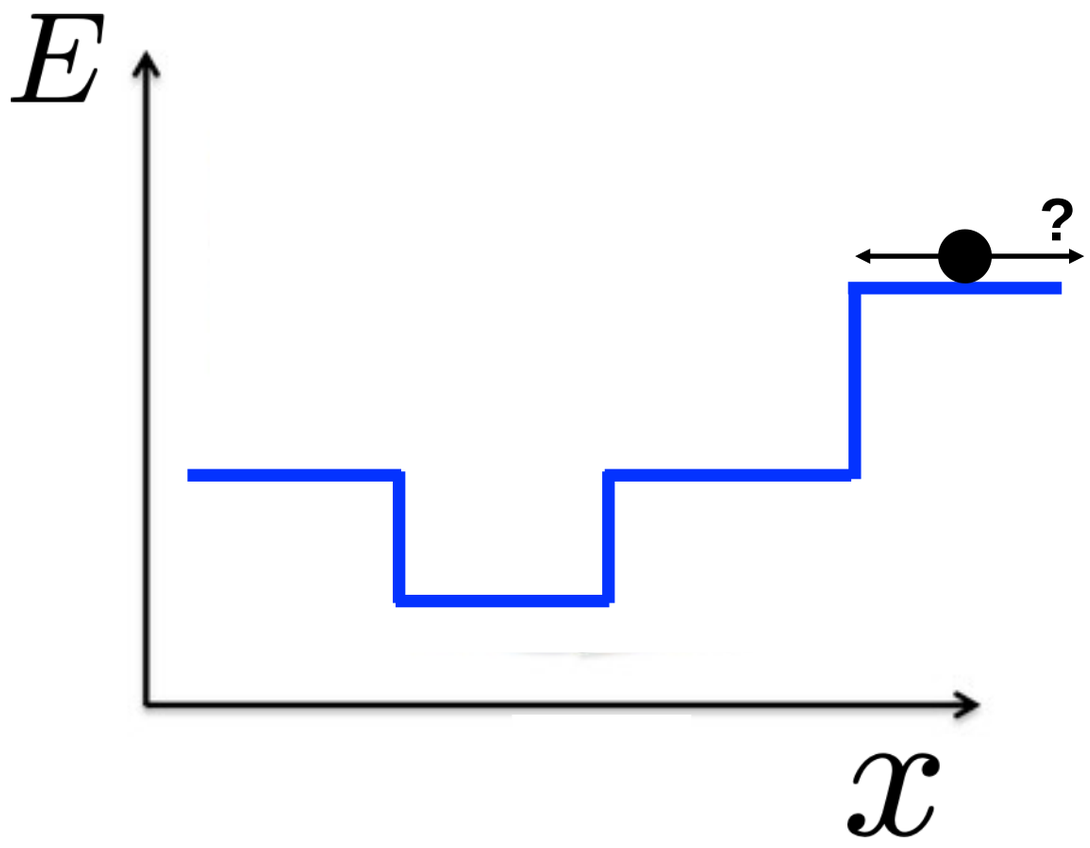
vector representing the "split"

learning rate

"gradient"

error function

Gradient descent



$$x_{i+1} = x_i - \eta \nabla E(x)$$

error function

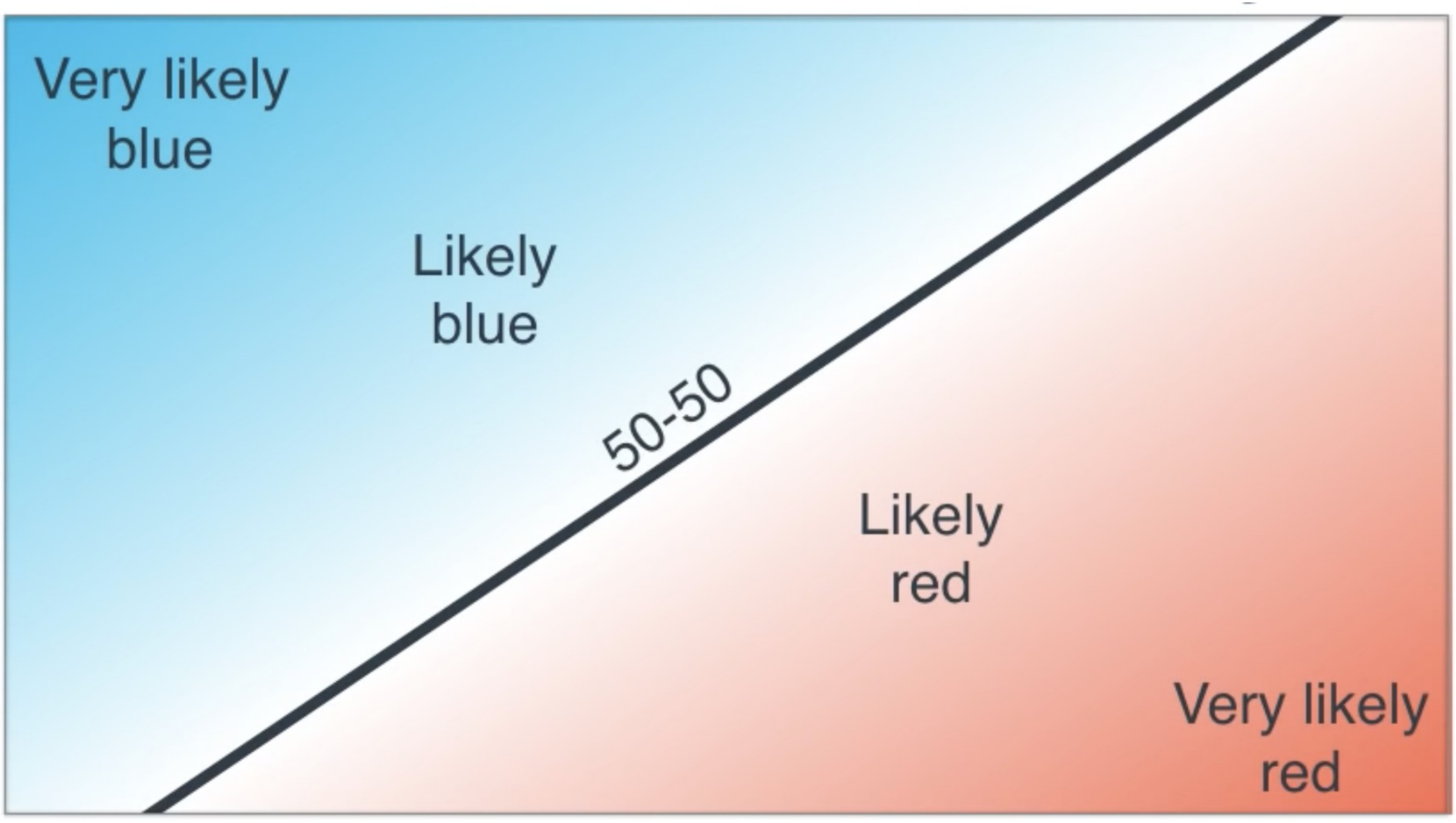
vector representing the “line/(hyper)plane”

learning rate

“gradient”

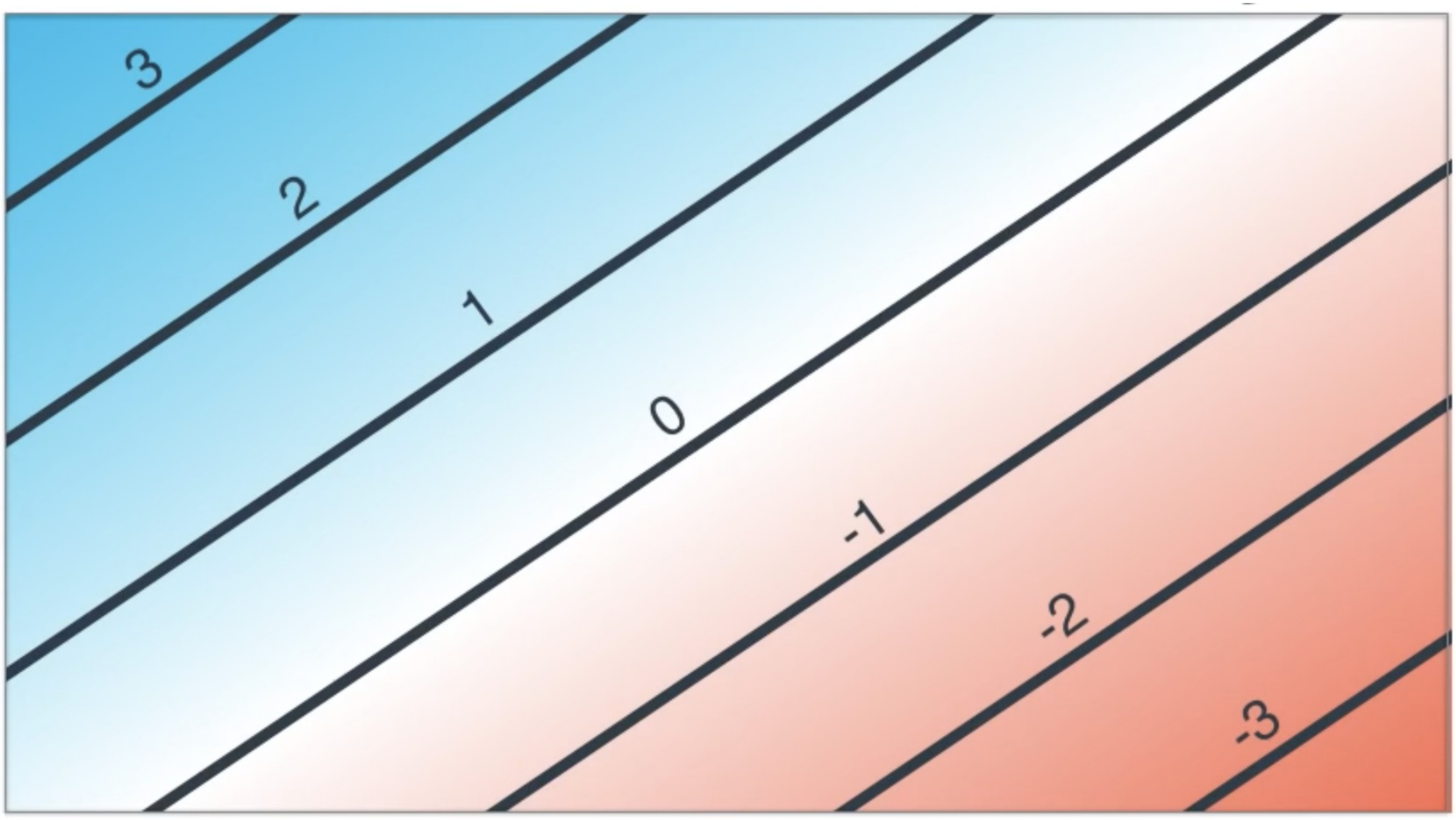
ML: Logistic Regression

Create likelihoods!

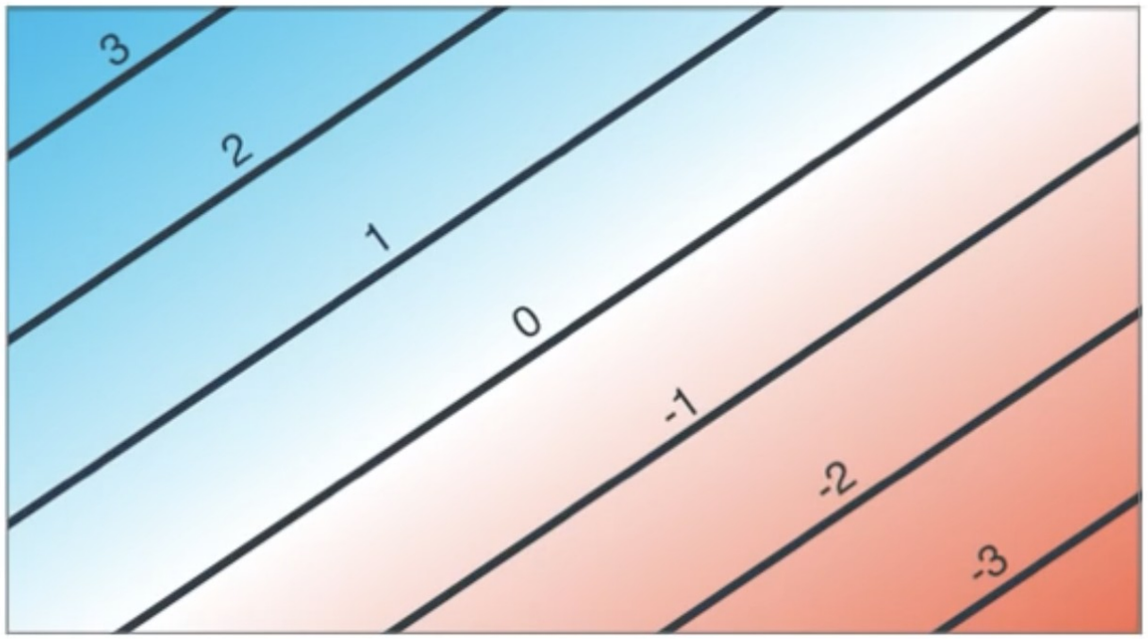


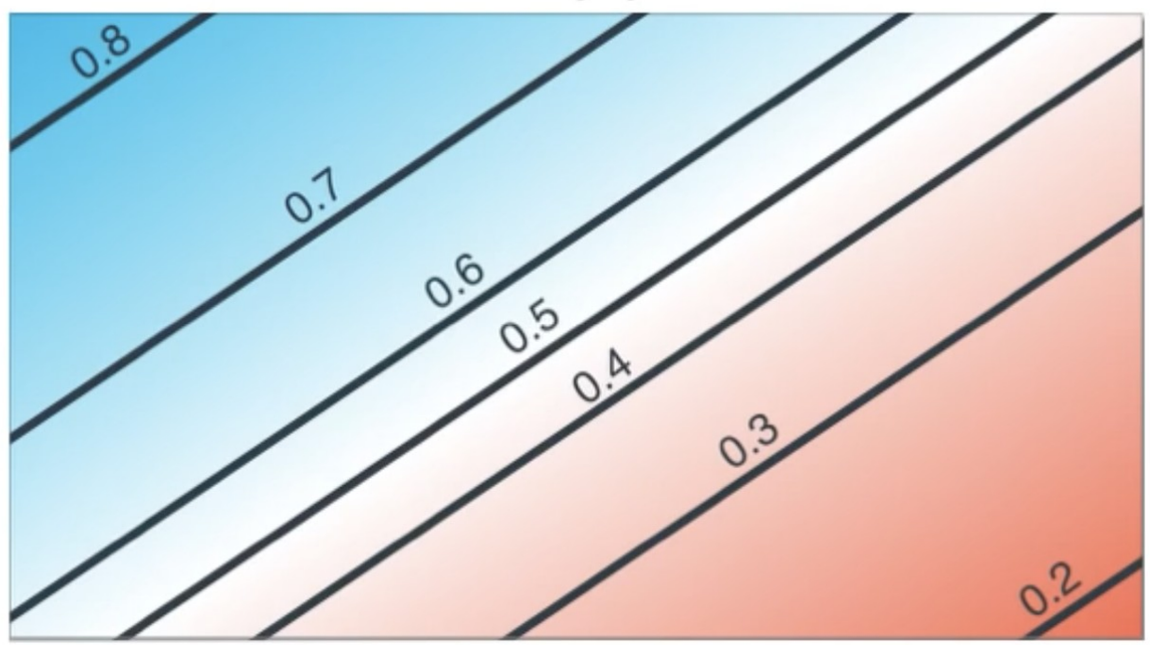
Logistic Regression

“Blueness"

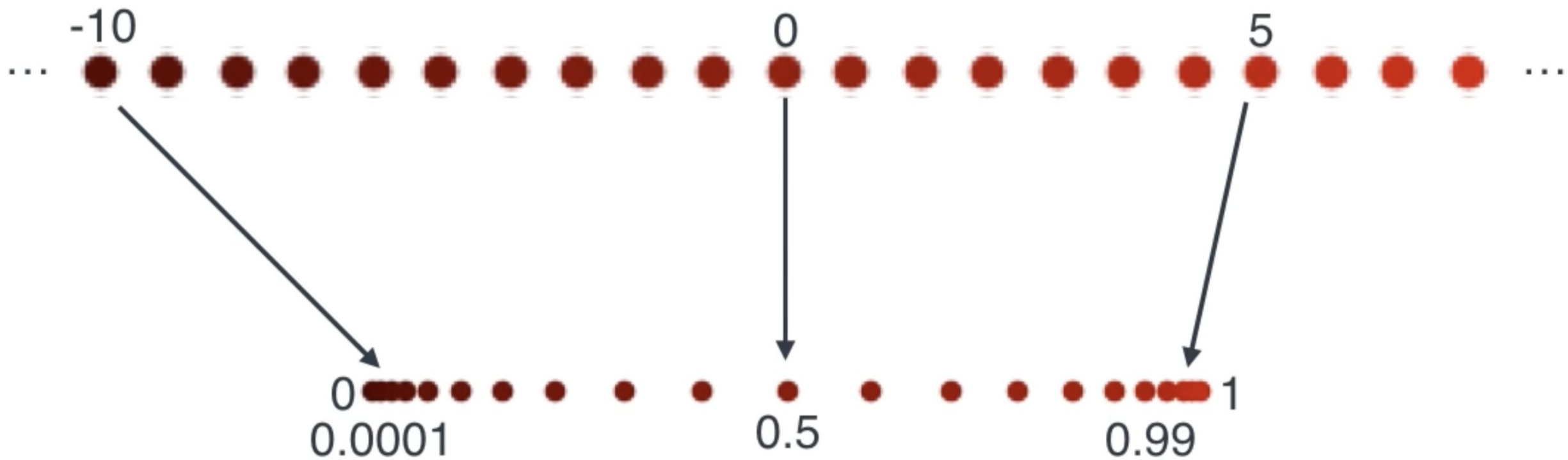


Probability

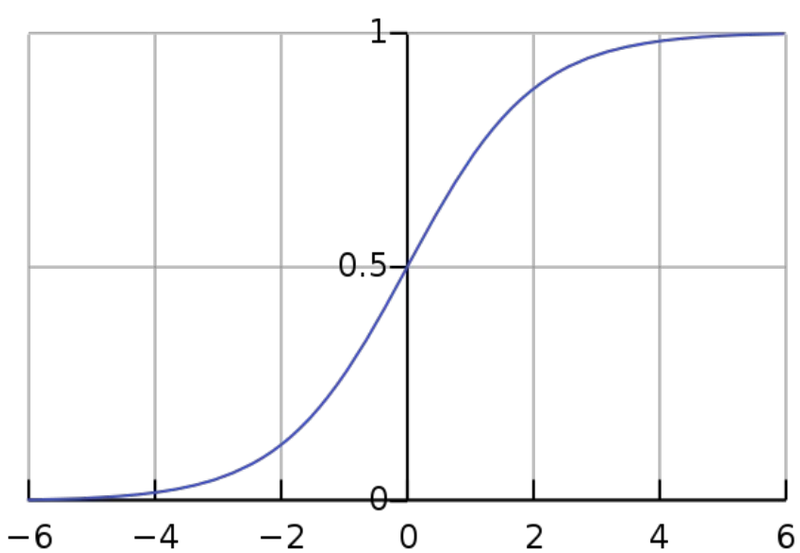




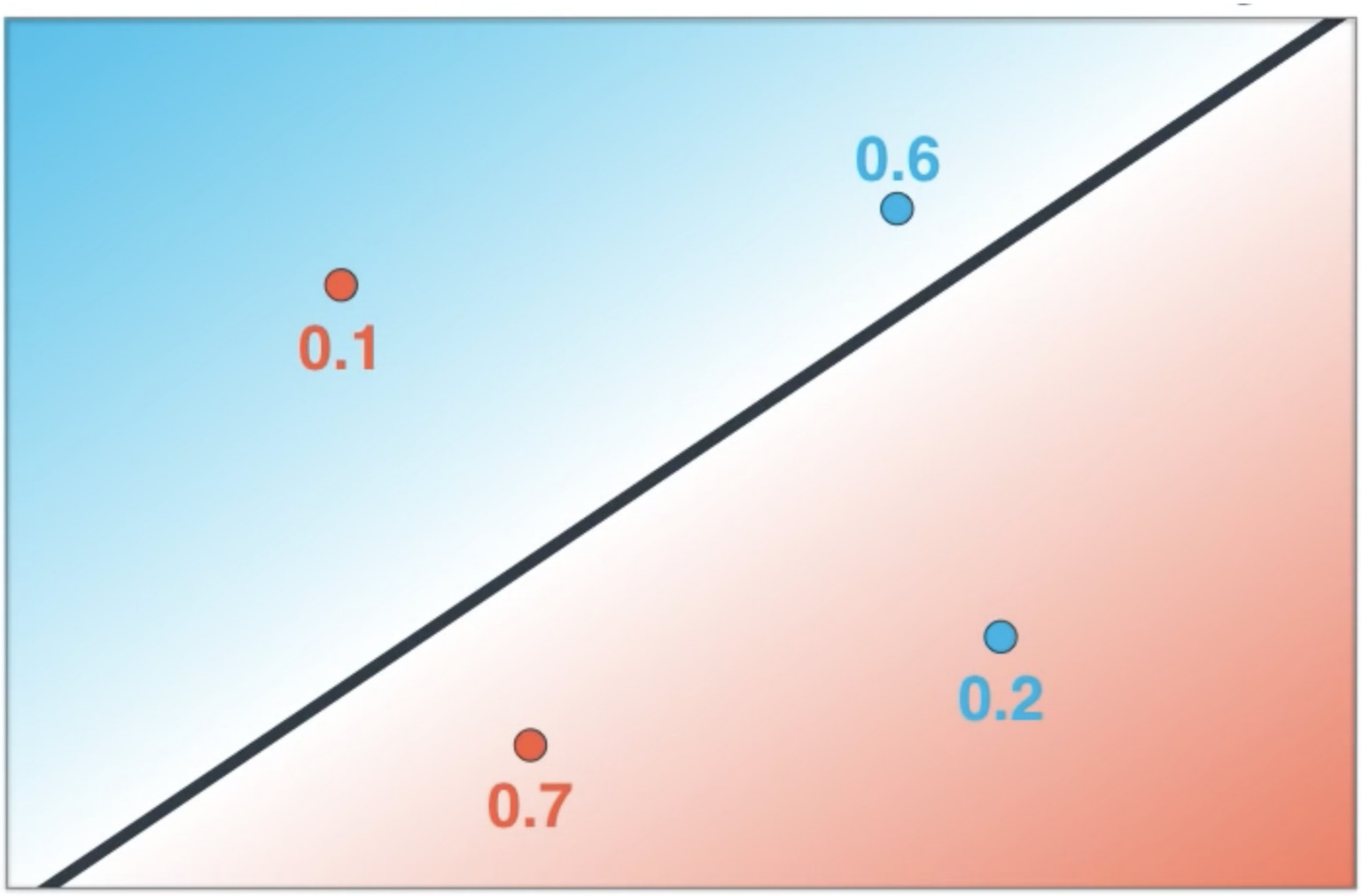
Activation function



Sigmoid function: $$f(x) = \frac{1}{1 + e^{-x}}$$

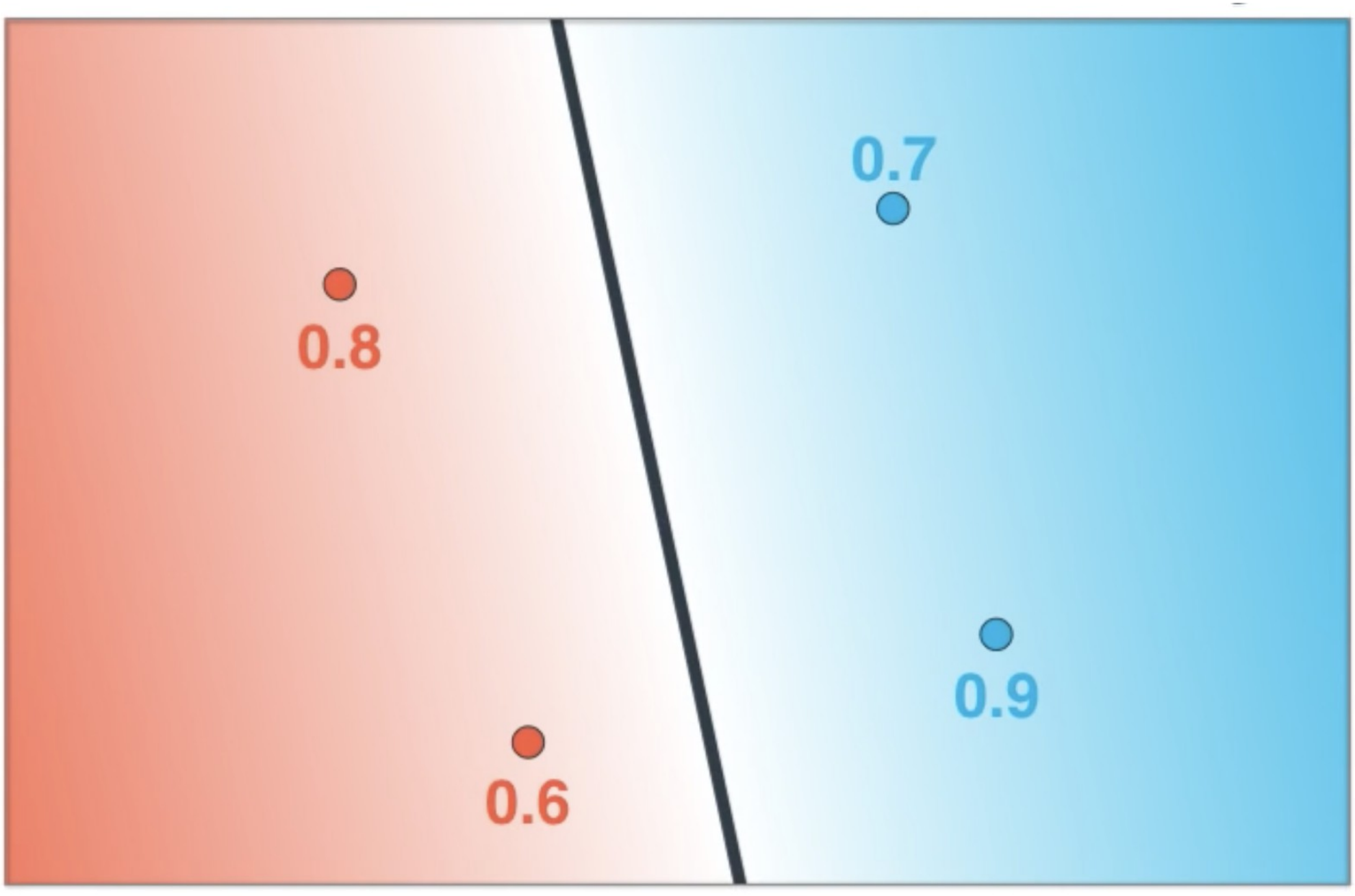


Probability



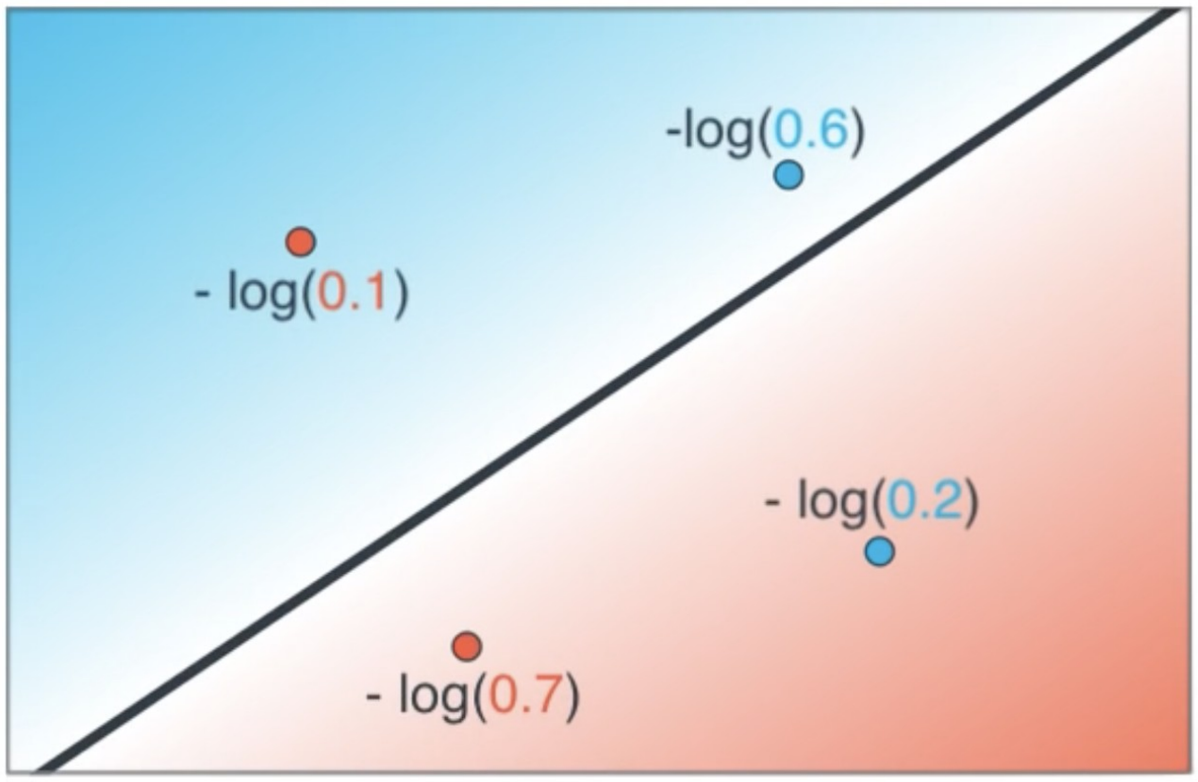
$$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$$

Probability



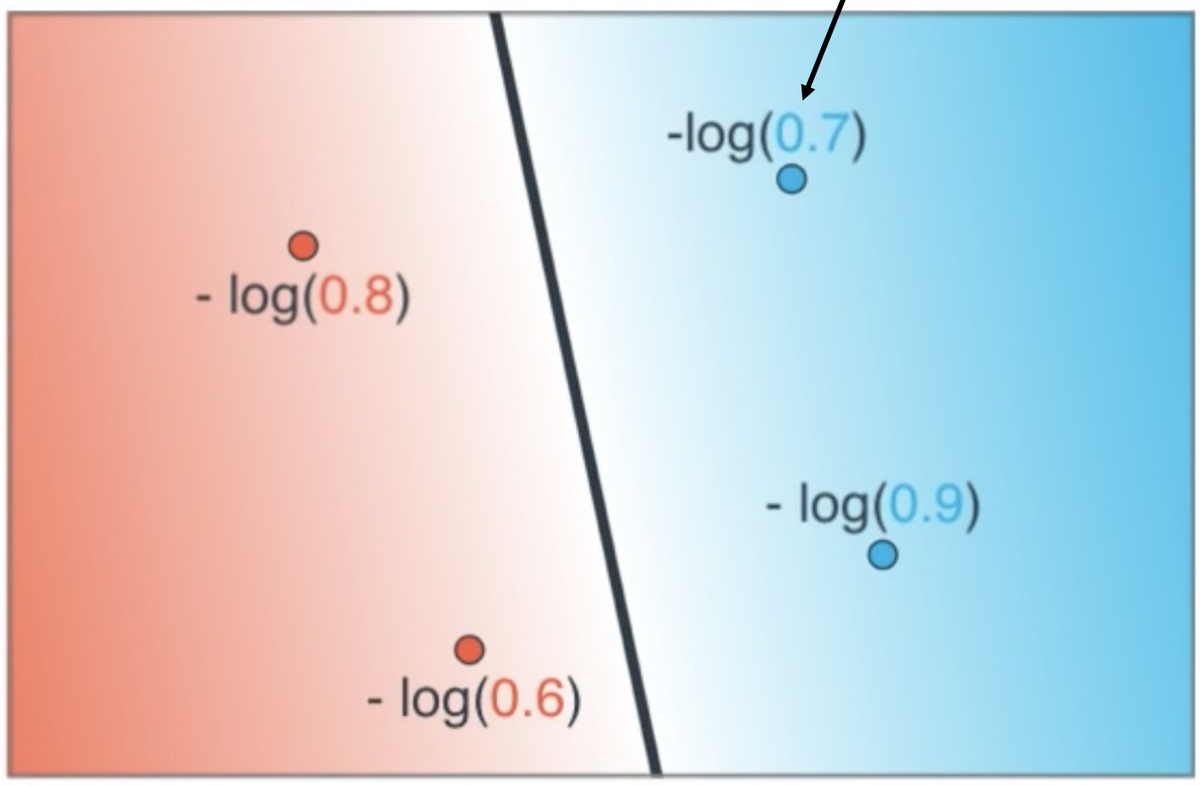
$$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$$

Log-likelihood error function



0.6*0.2*0.1*0.7 = 0.0084

-log(0.6) - log(0.2) - log(0.1) - log(0.7) = 4.8



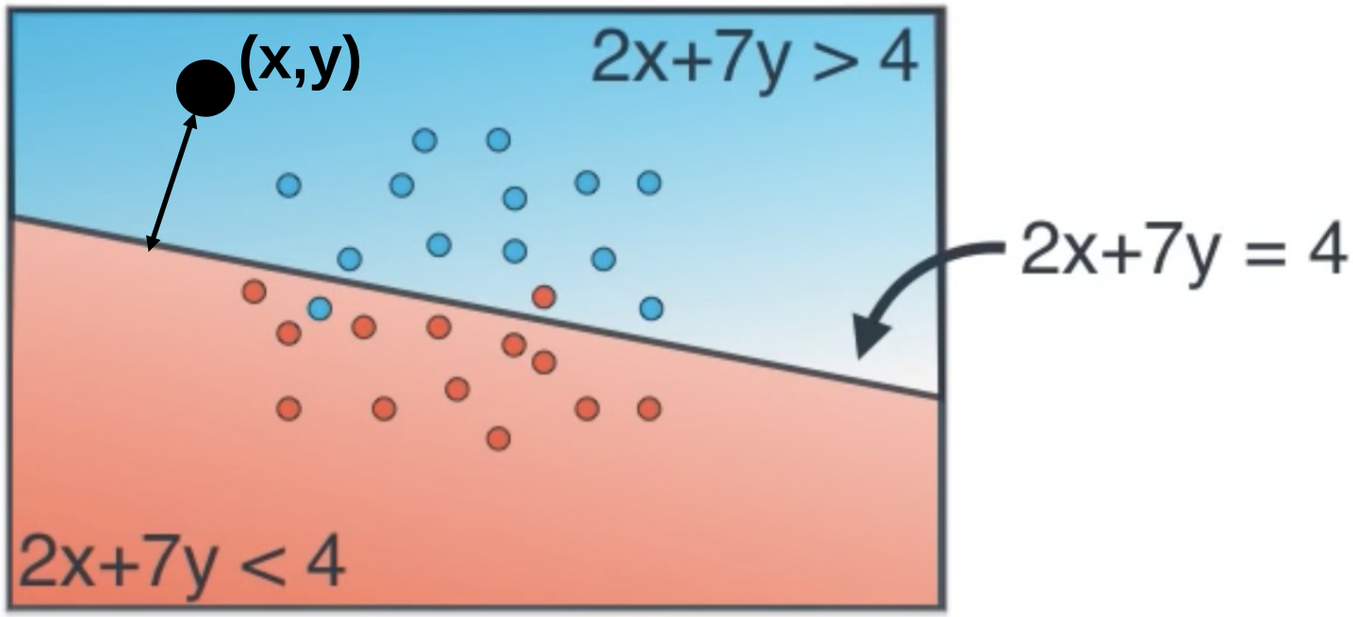
0.7*0.9*0.8*0.6 = 0.3024

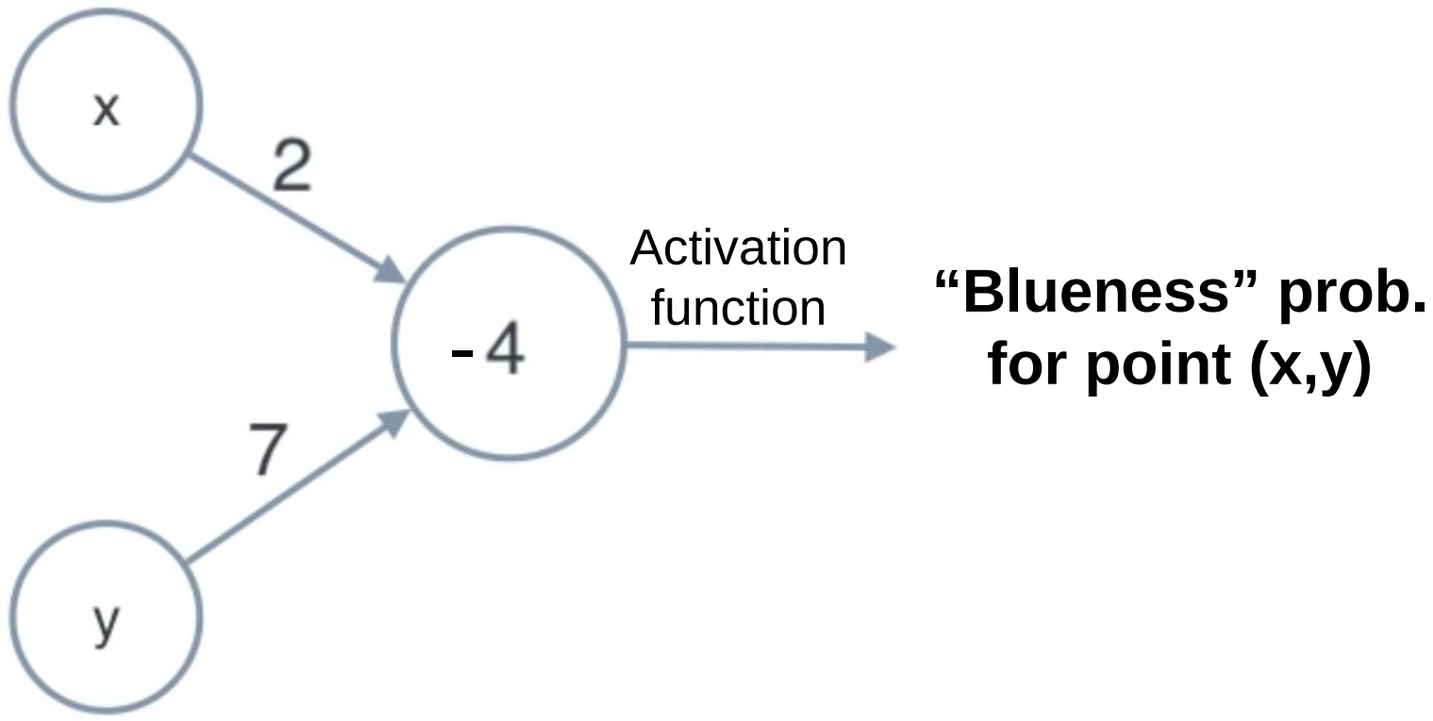
-log(0.7) - log(0.9) - log(0.8) - log(0.6) = 1.2

Find the line that minimises the total *continuous* error function

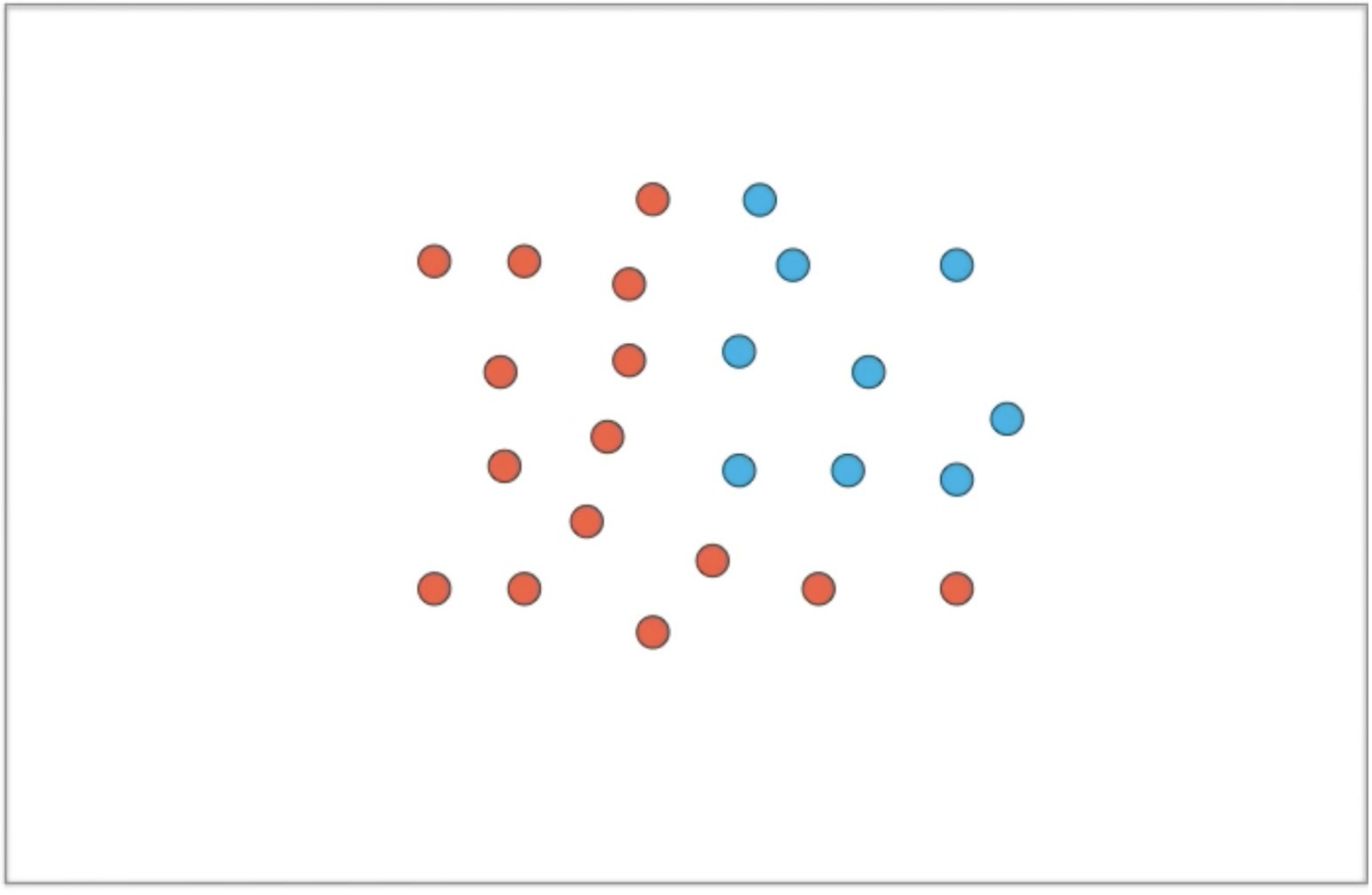
…this method is called *logistic regression*

A Neuron

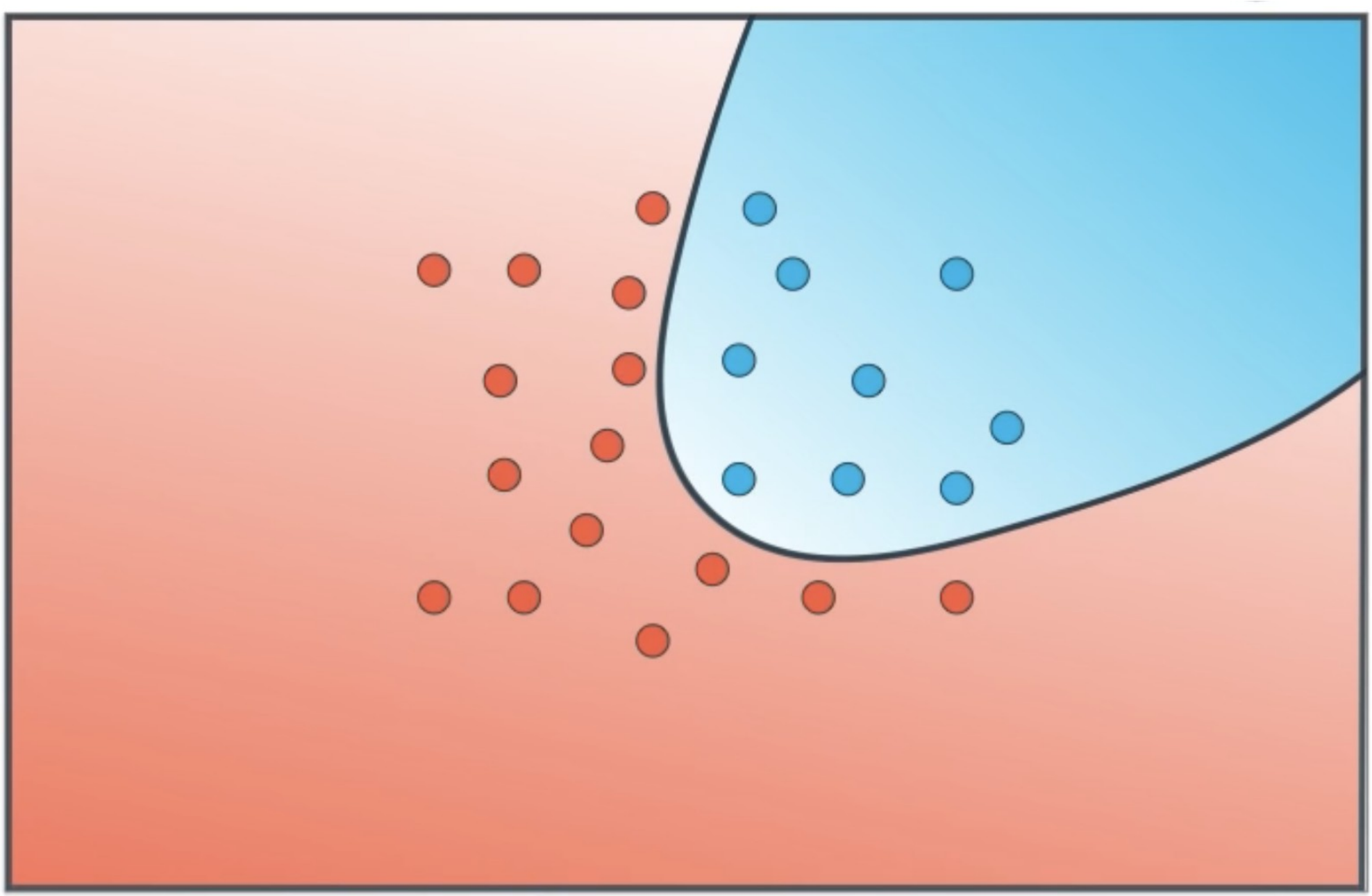

(x,y)
2x+7y > 4
2x+7y = 4
2x+7y < 4


x
2
7
y
-4
Activation function
“Blueness” prob. for point (x,y)

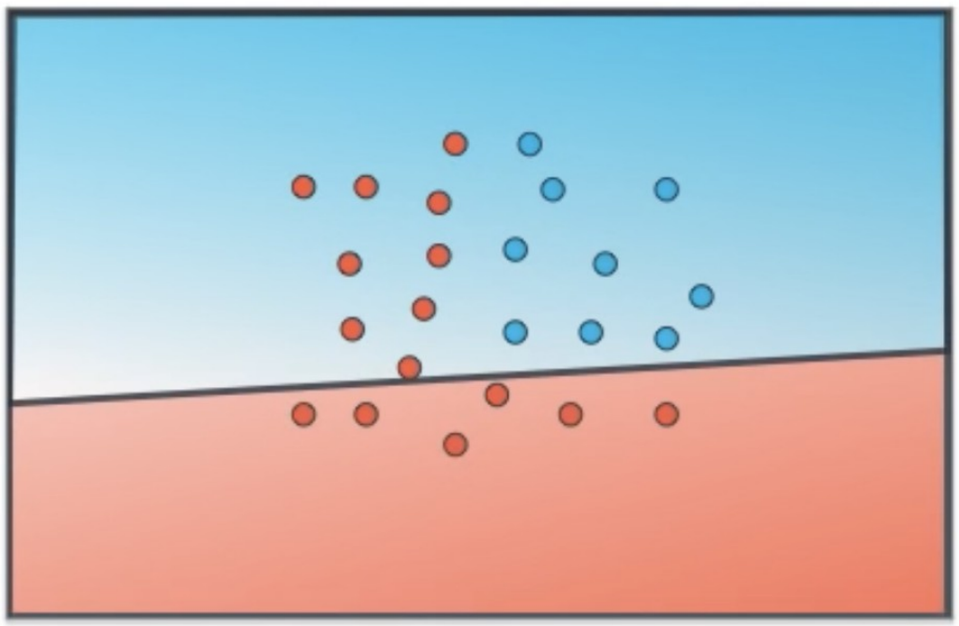
Non-linear regions

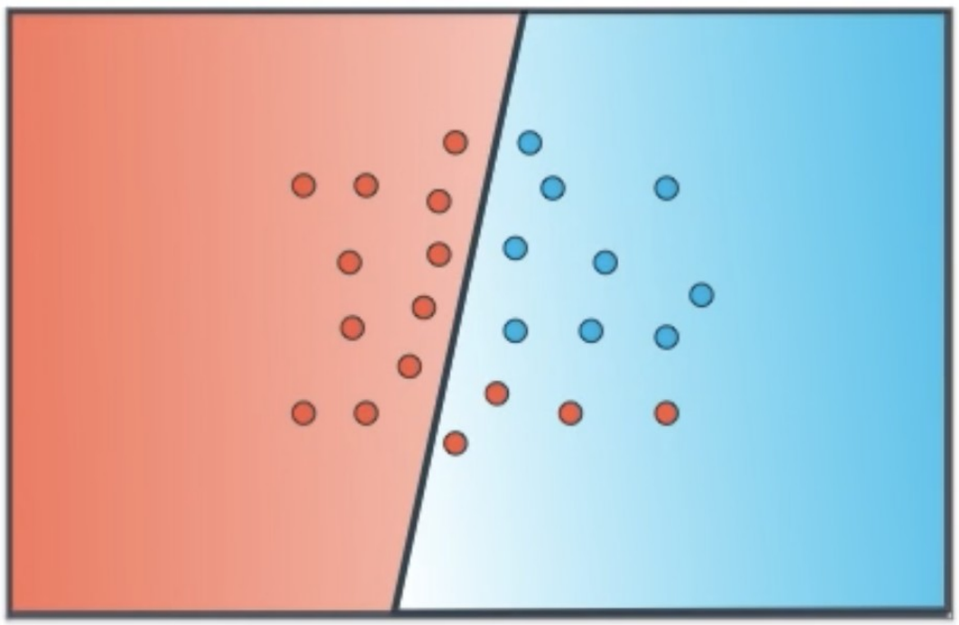


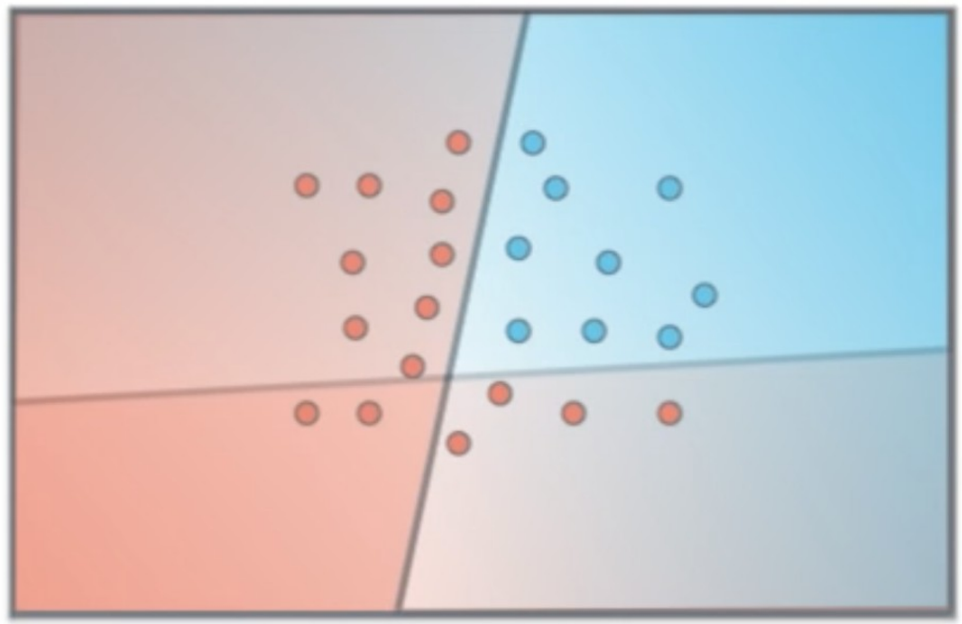
Non-linear regions



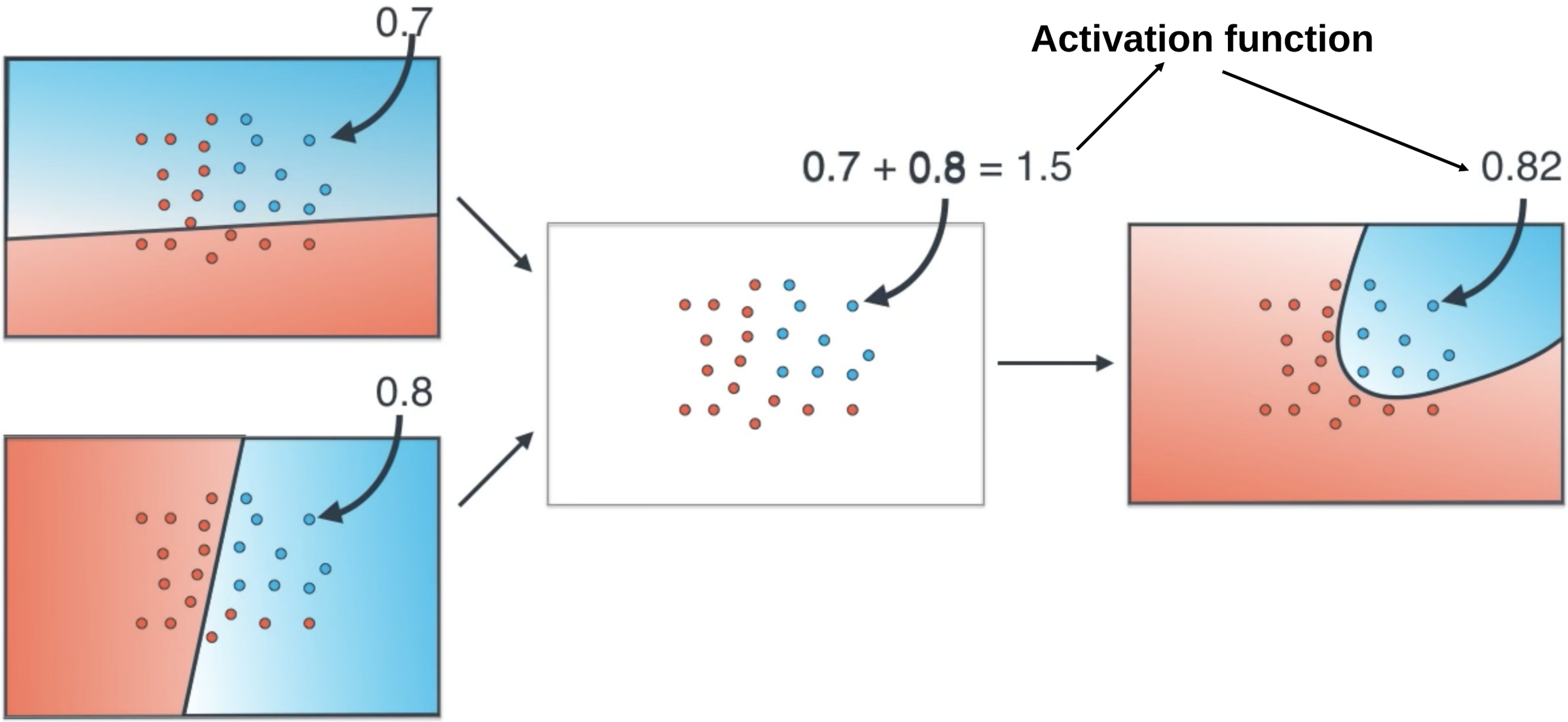
Non-linear regions



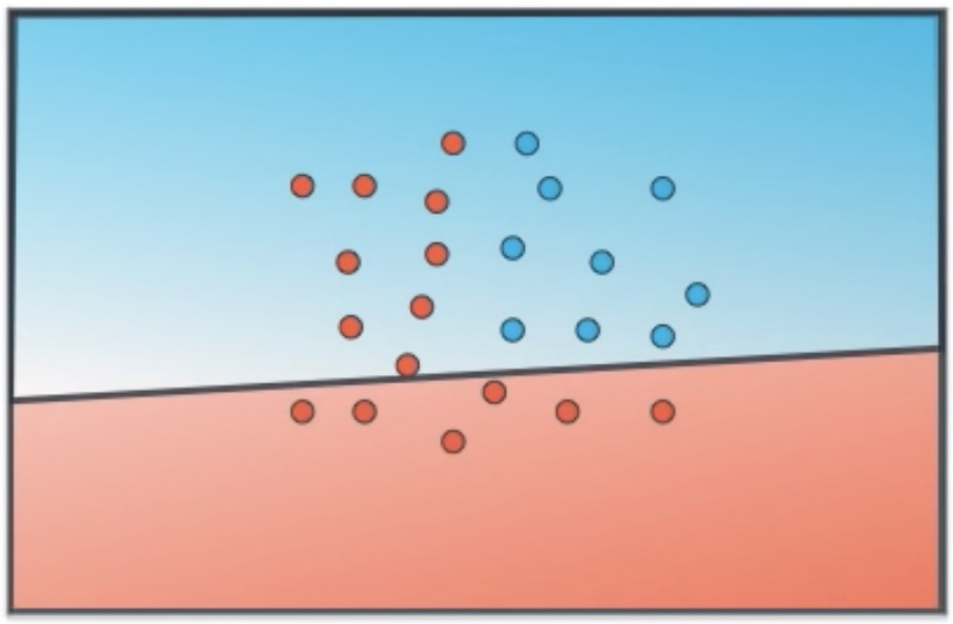


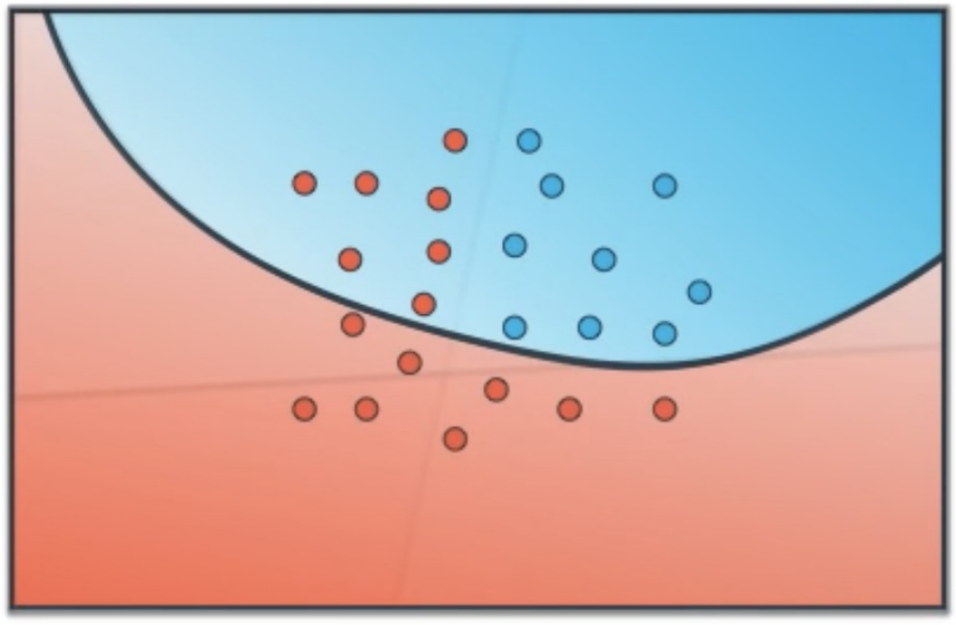


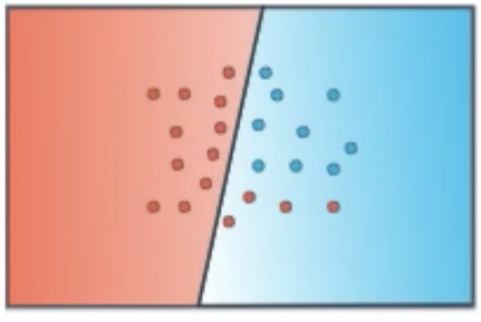
Non-linear regions



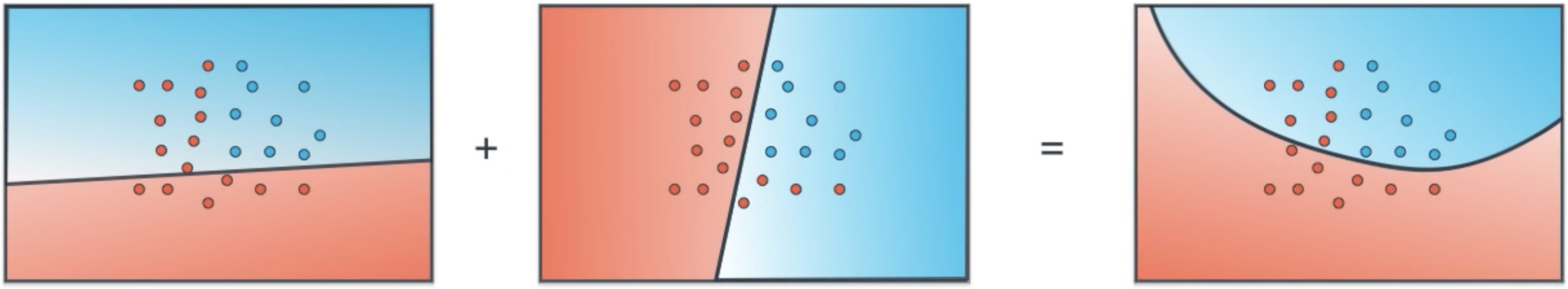
Non-linear regions



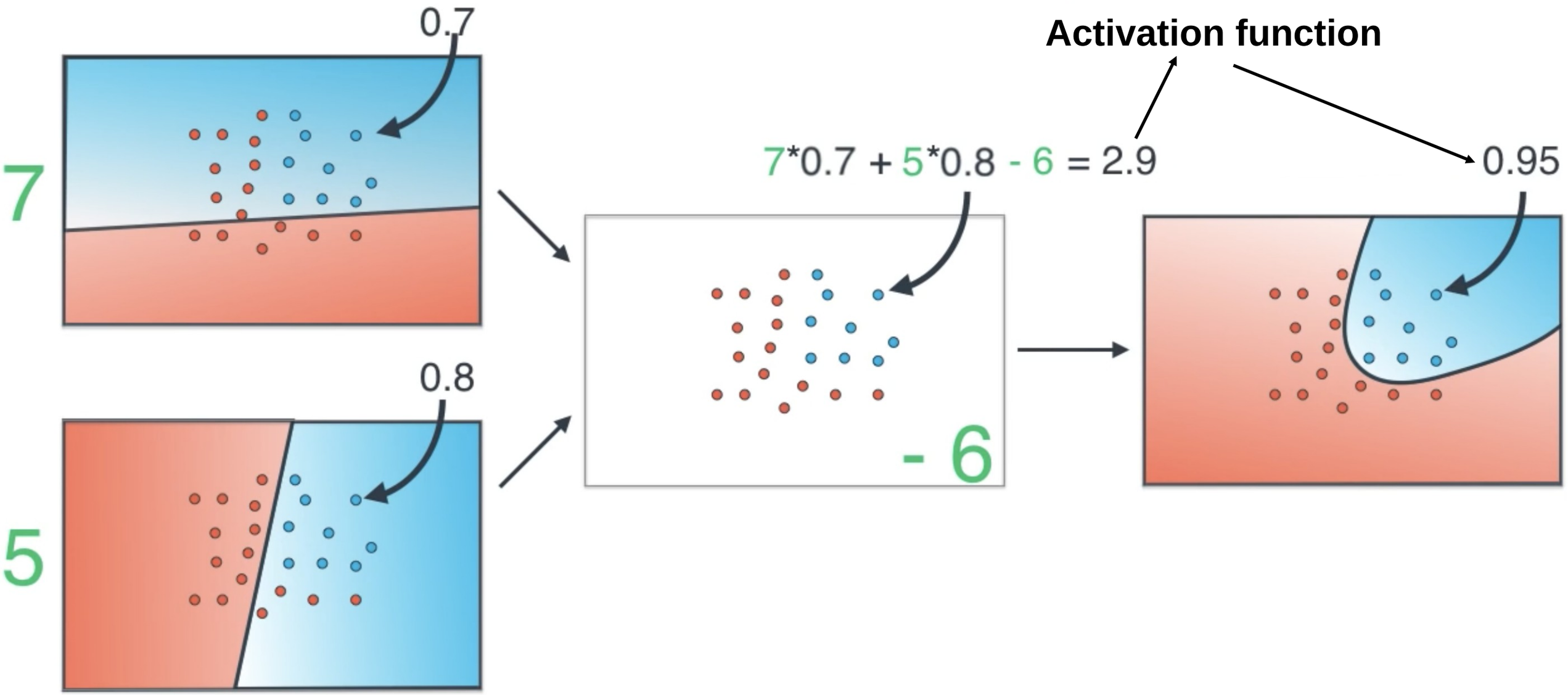




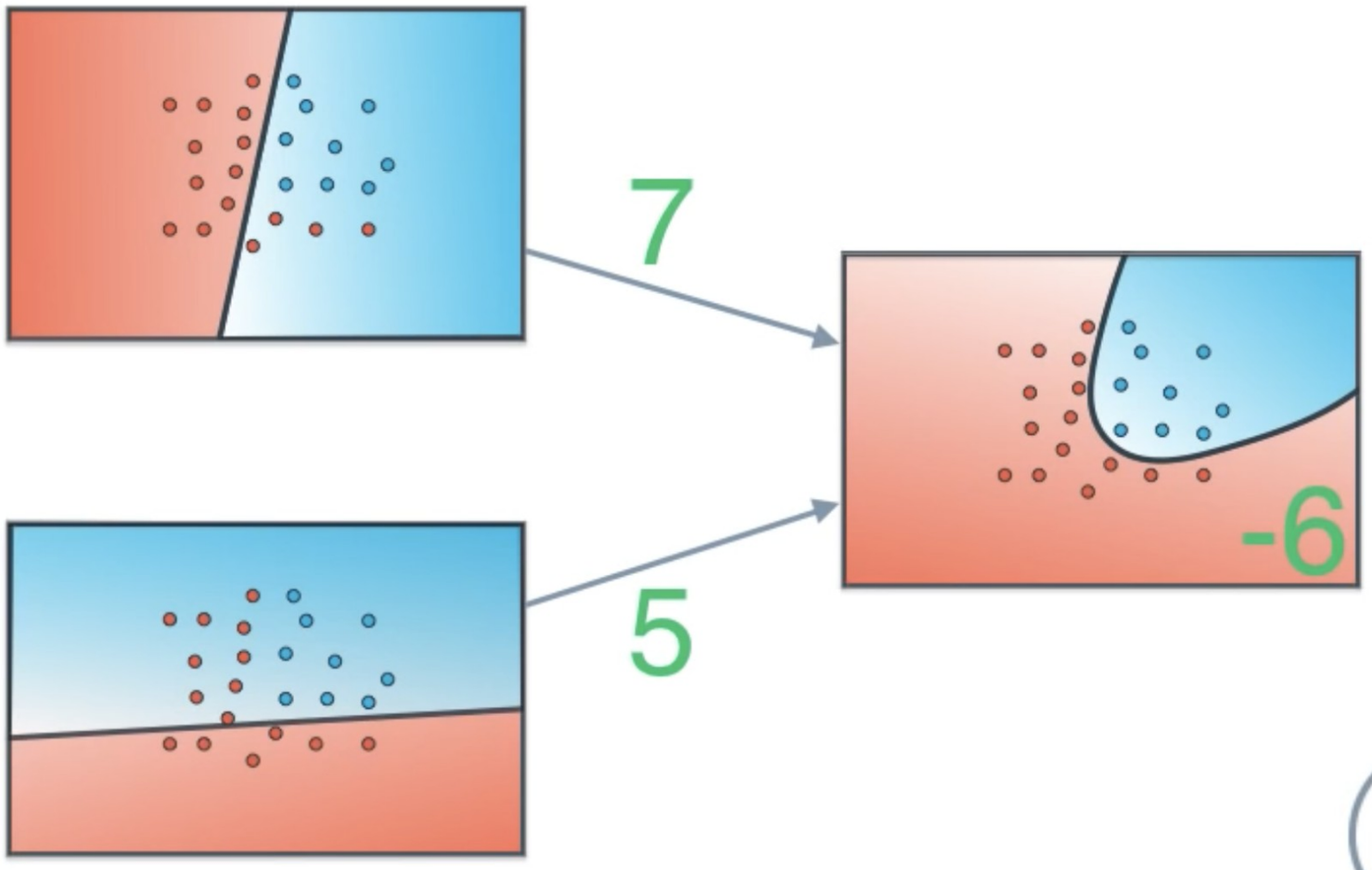
Non-linear regions

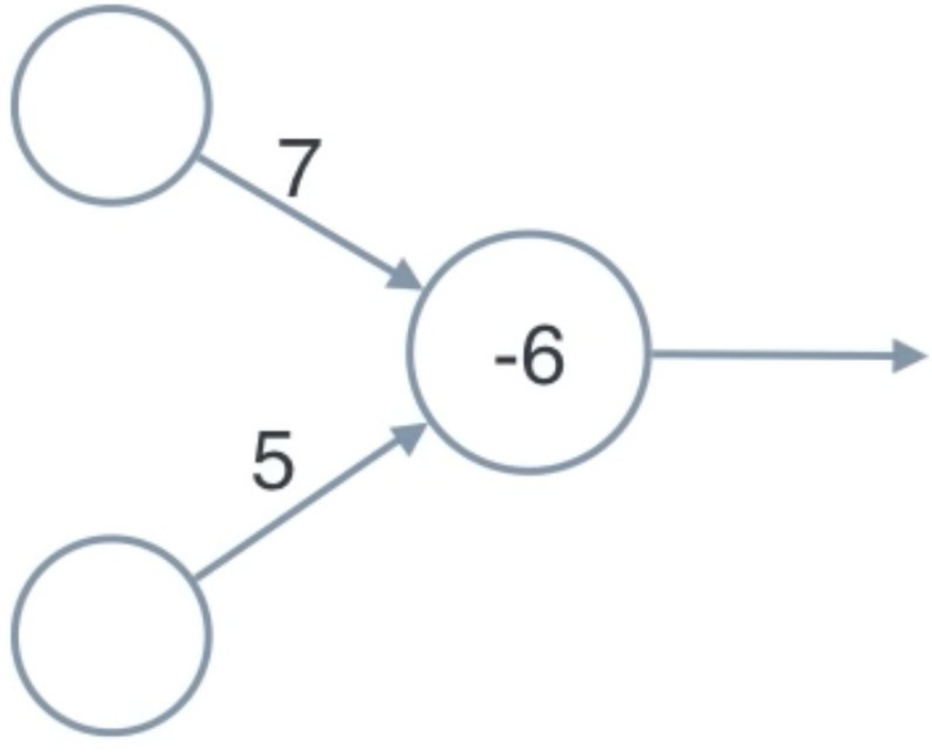


Non-linear regions

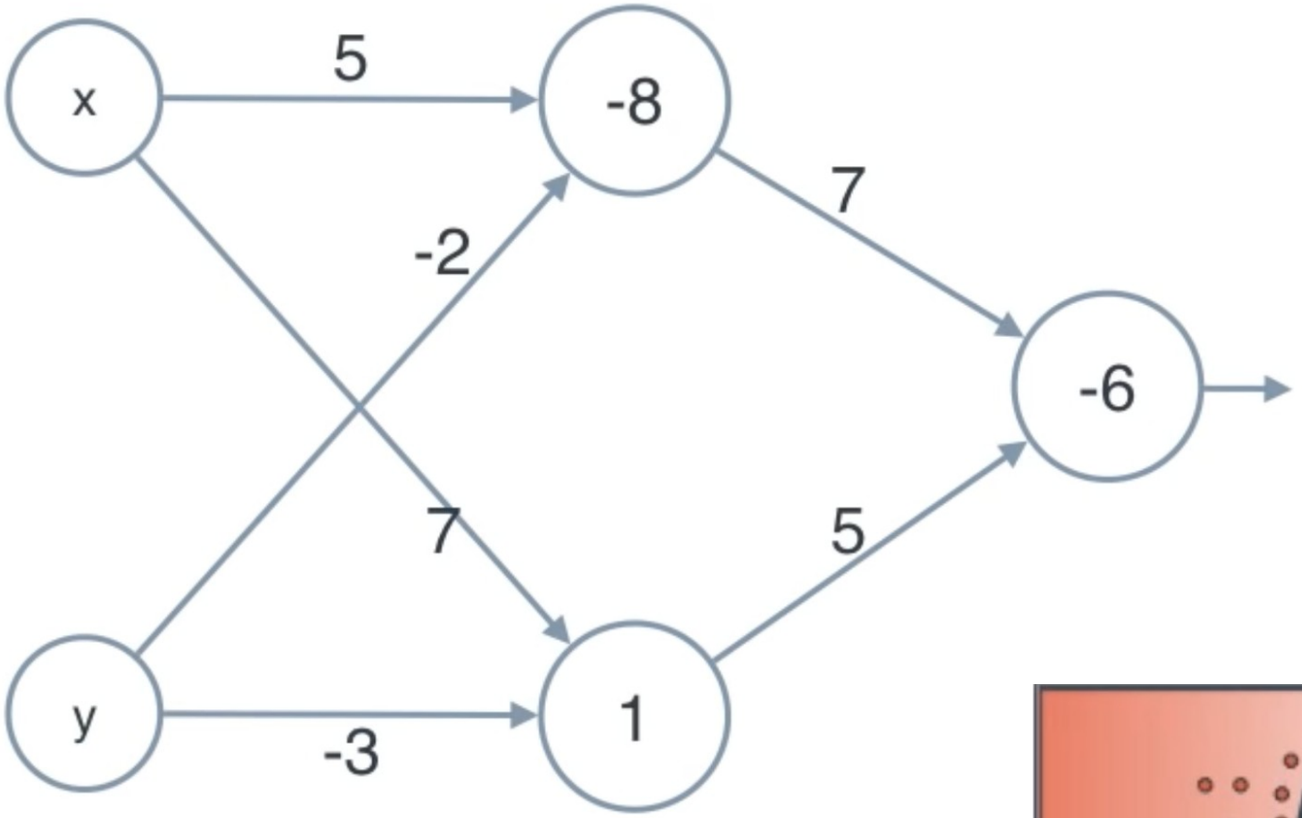


A hidden Neuron

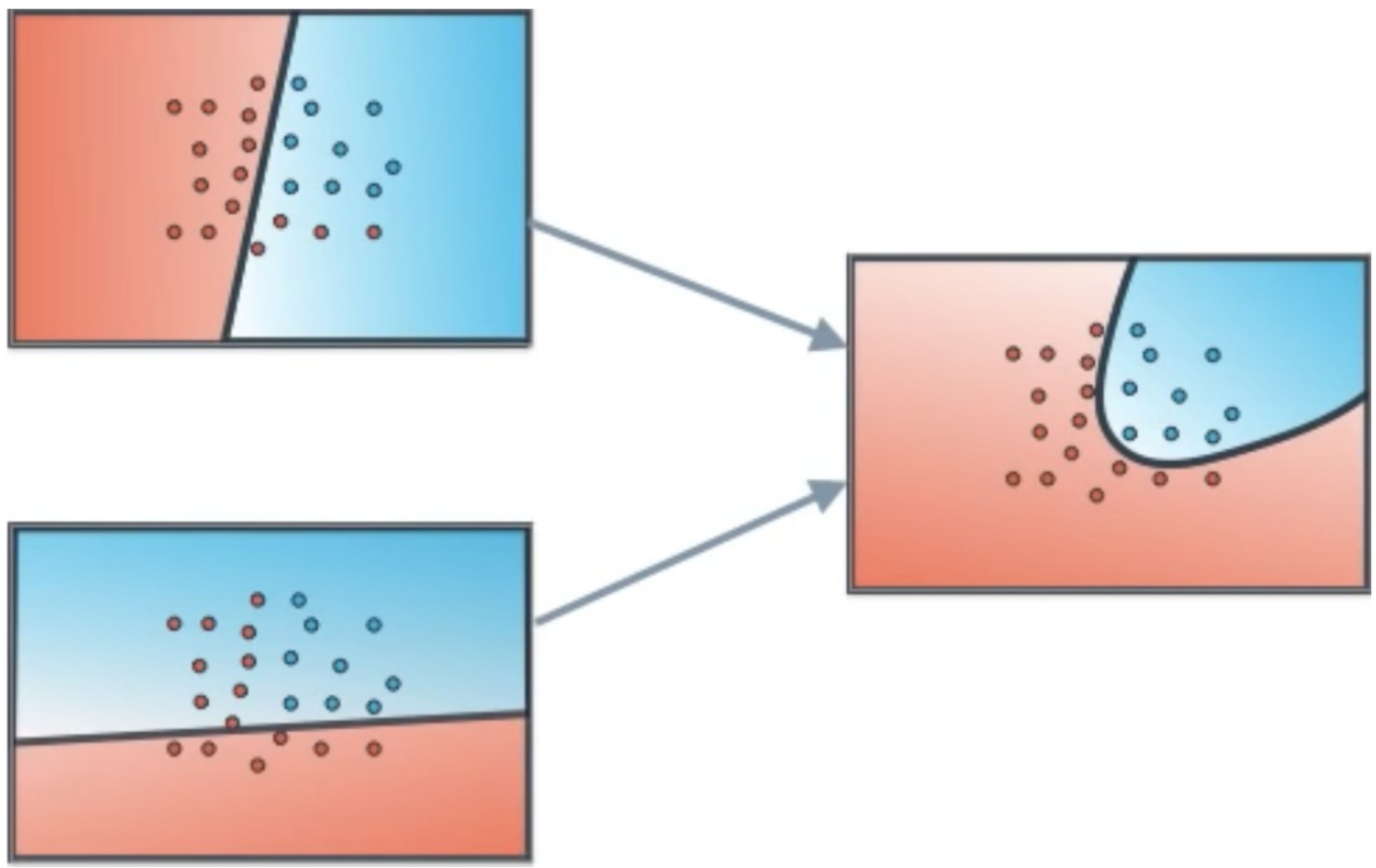




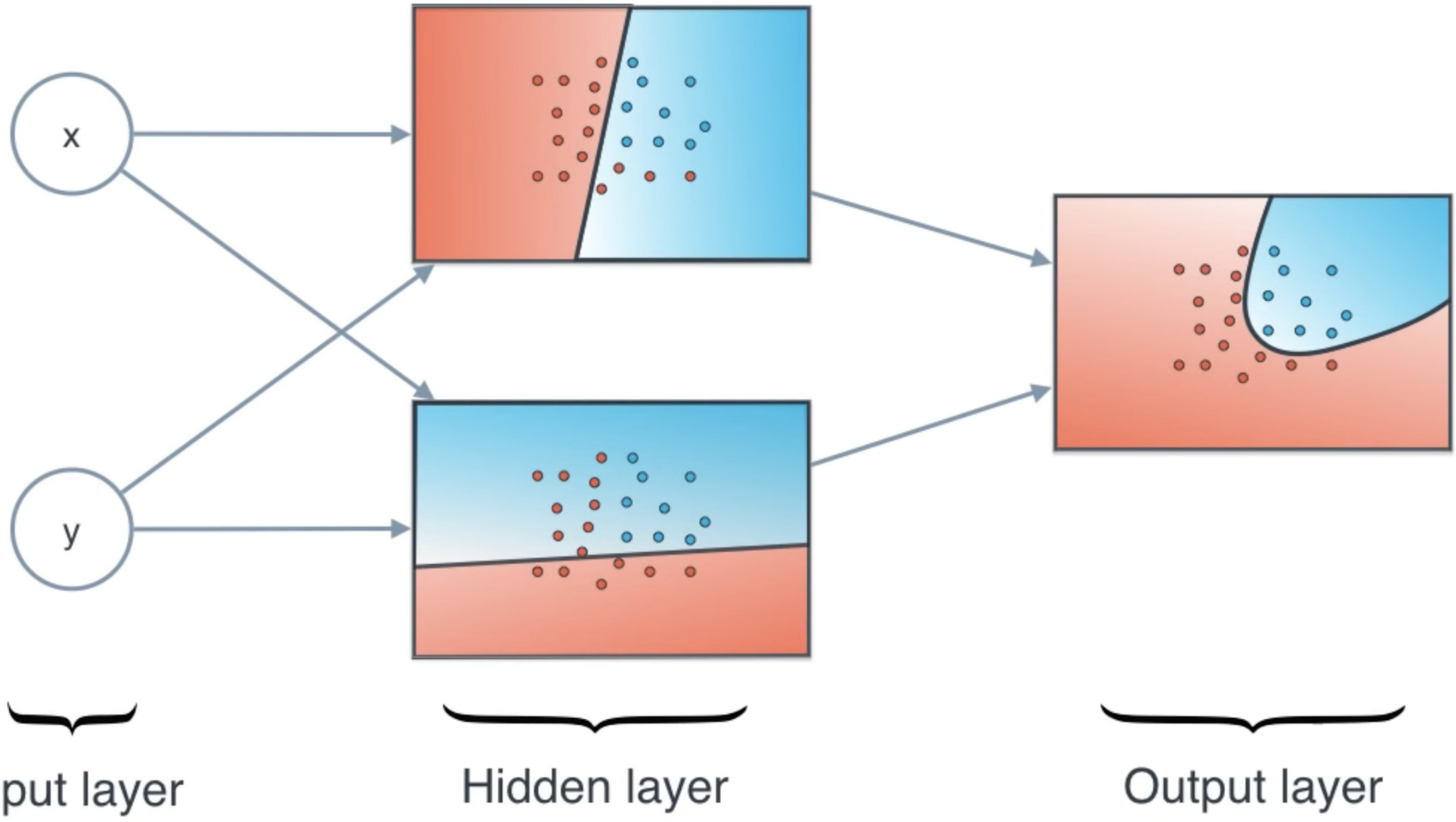
Neural Network



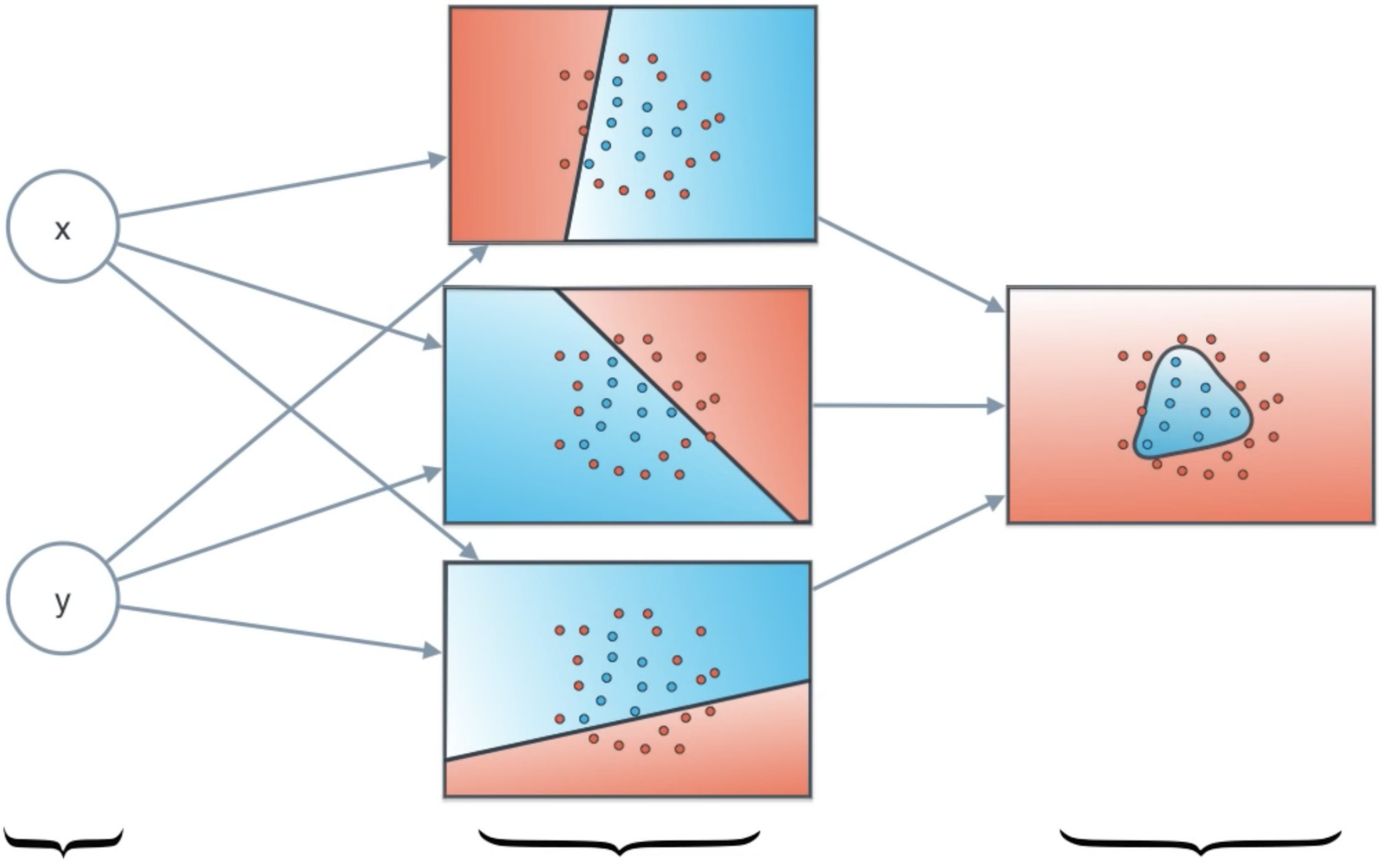
=



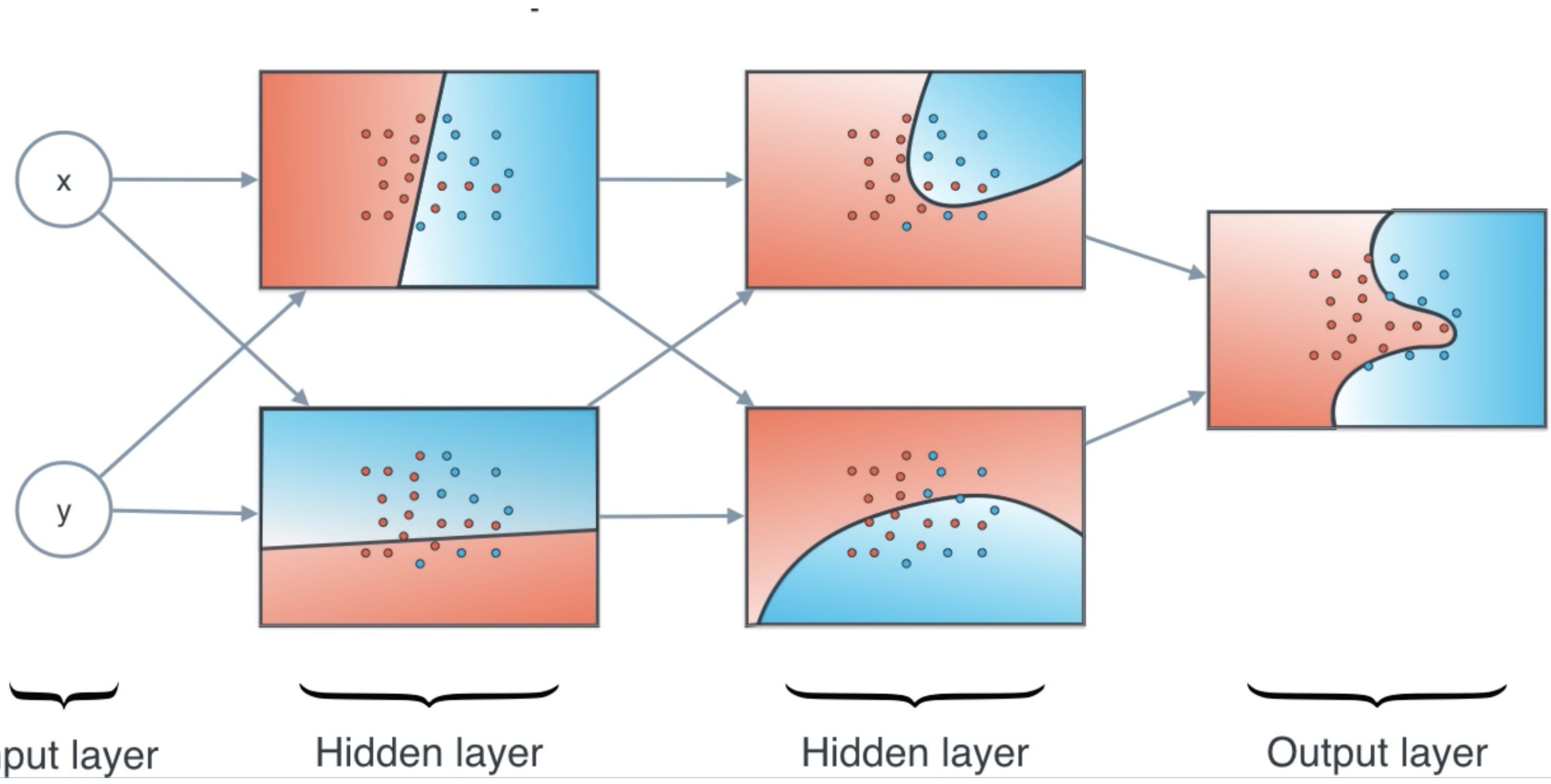
Neural Network



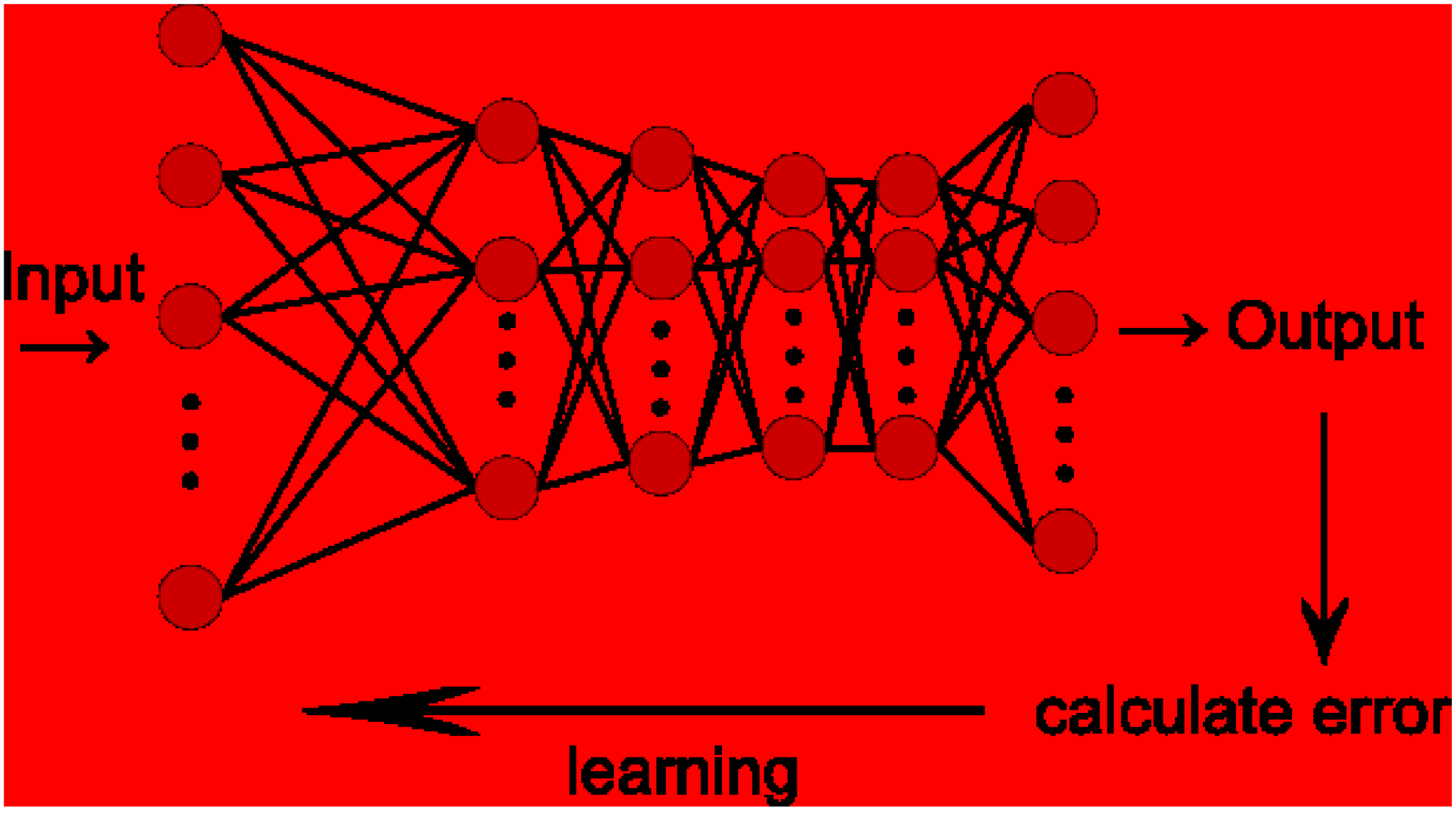
Neural Network



Deep Neural Network

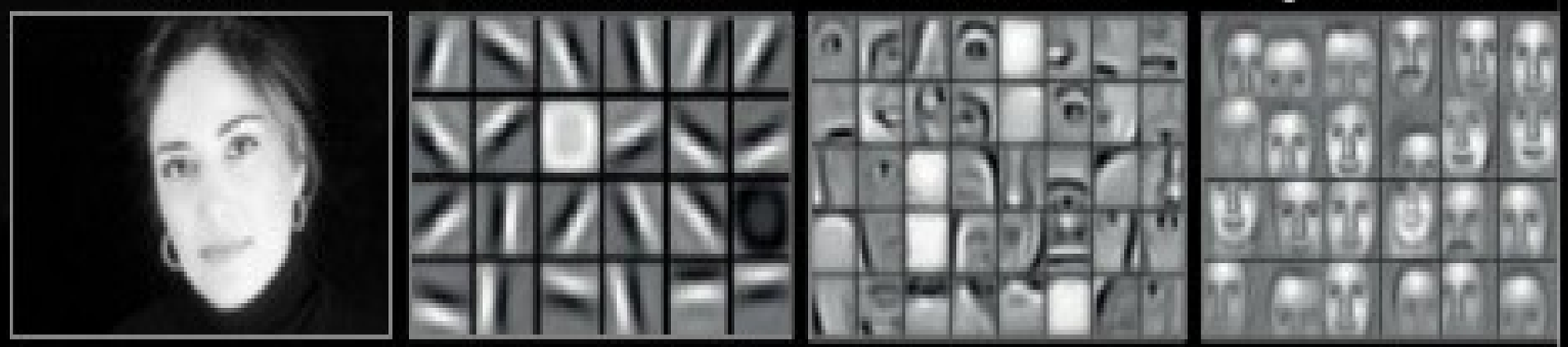


Deep Neural Network



“back propagation”

DEEP NEURAL NETWORK (DNN)



Convolutional Neural Network

Application components:

Task objective
- e.g. Identify face

Training data
- 10-100M images

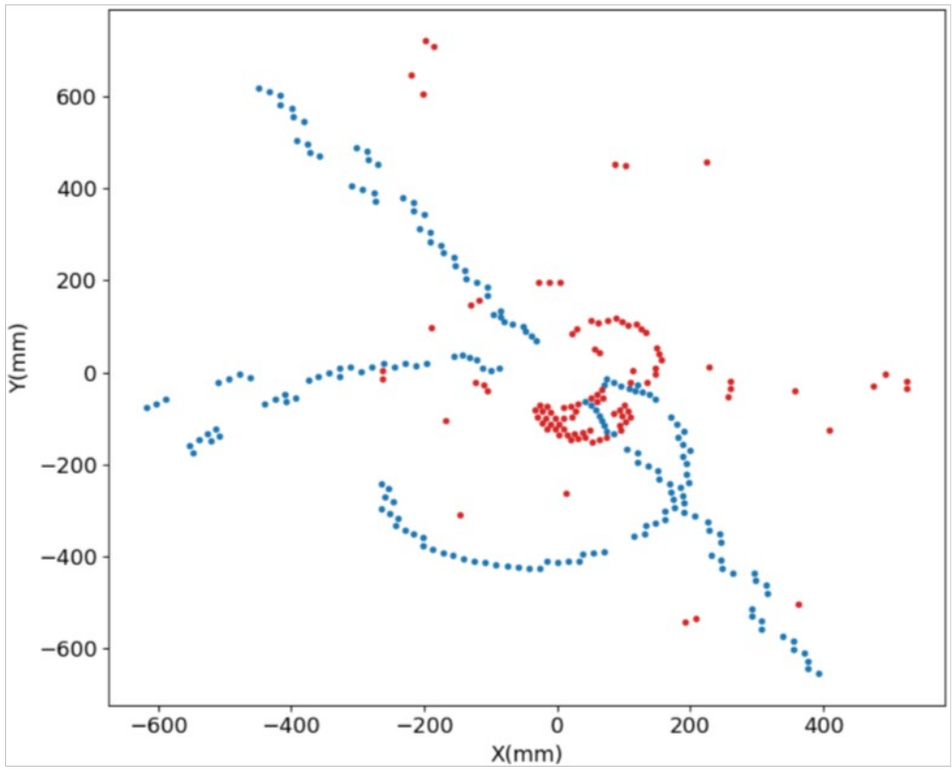
Network architecture
- ~10 layers
- 1B parameters

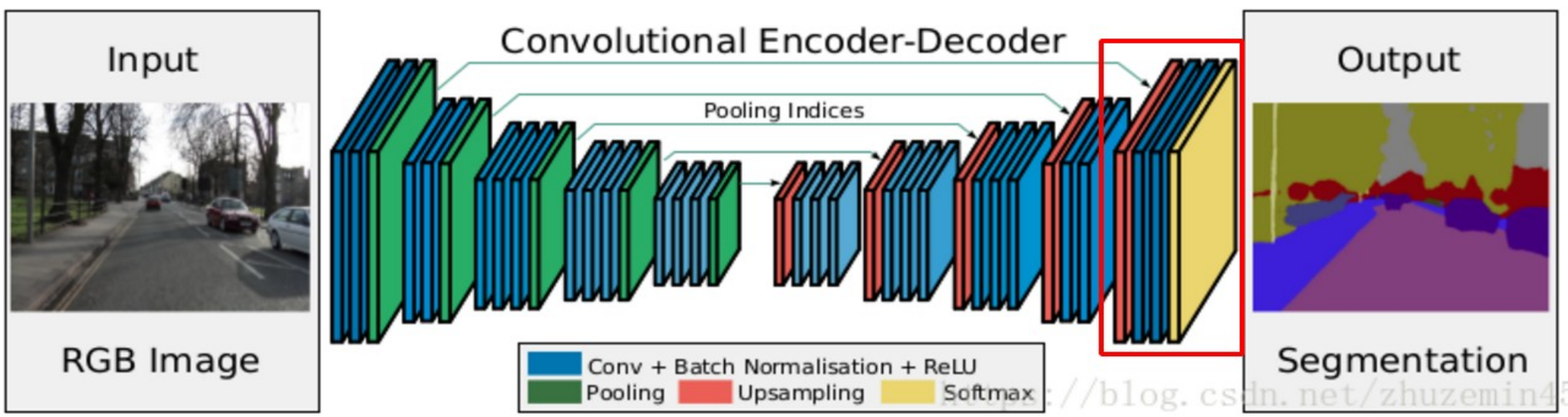
Learning algorithm
- ~30 Exaflops
- ~30 GPU days

Tracking sub-atomic particles using *deep learning*

Convolutional neural networks

- Supervised feature extraction of track images of BESIII drift chamber.
- Demonstration of noise reduction, particle identification and track finding capabilities.
- Most promising design: U-net convolutional network originating from biomedical image segmentation.
- Outlook: track momentum reconstruction.





JavaScript L1 (default)

H. de Vries, J.M., master thesis, University of Groningen 2019

Neural Network, pros and cons

Pros:

- Mimics biological systems
- Works in high dimensions (large feature space)
- Accounts for non-linear dependences
- Very flexible: *in principle* very powerful

Cons:

- Many parameters to tune (#layers, #neurons)
- Very slow training process
- Difficult to get insight in what it does
- Not advisable to use as a black box!