Introduction to Neural Networks

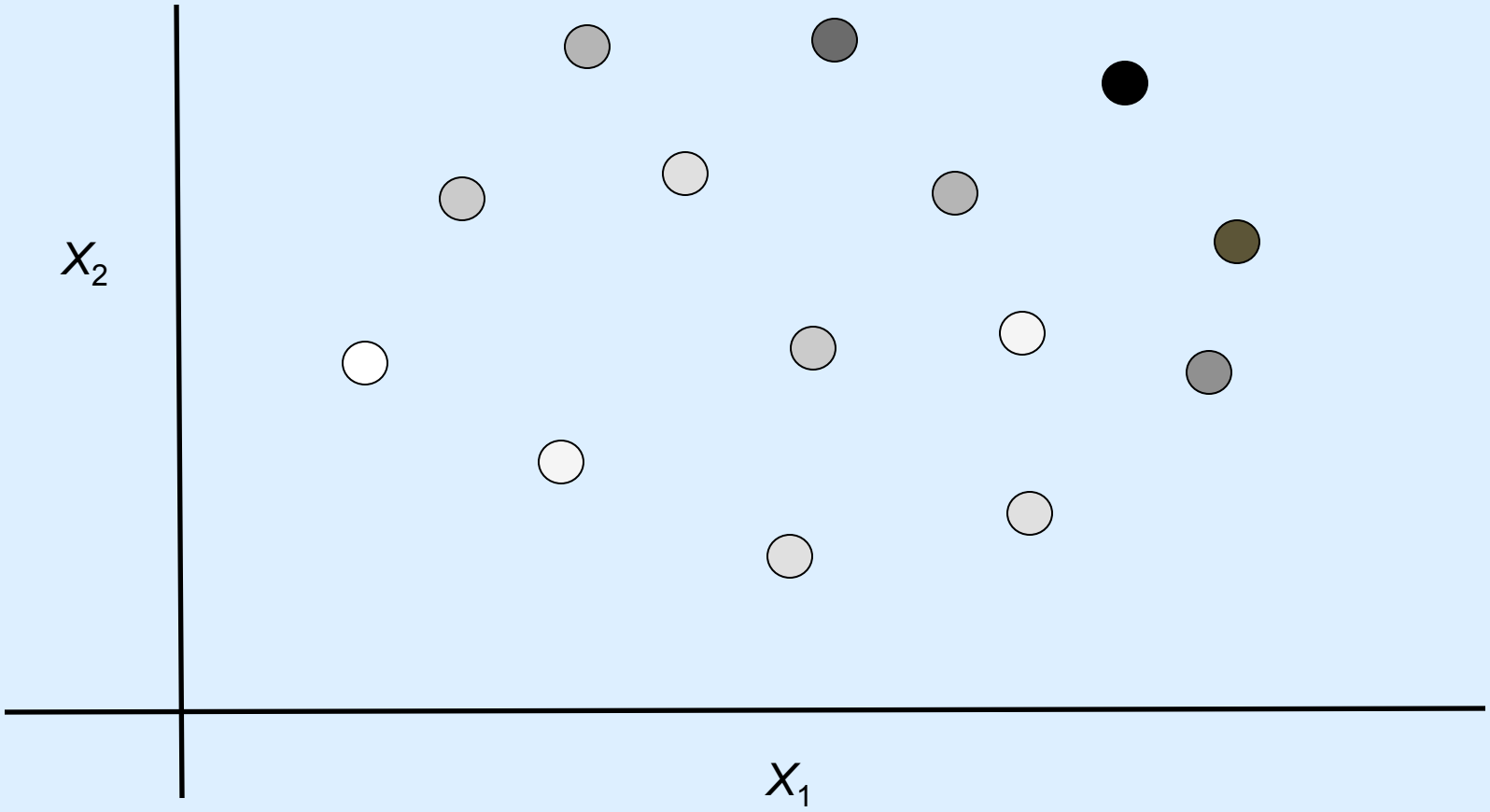
Johns Hopkins University
Engineering for Professionals Program
605-447/625-438
Dr. Mark Fleischer

Module 7.1: The Training Process

This Sub-Module Covers …

- What a typical training scenario looks like.
- Issues that arise during the training process.
  - Overtraining and efficiency
  - Lay the groundwork for describing procedures to handle multiple input/output data pairs and related issues
  - Motivate important performance measures covered in subsequent sub-modules

A Classification Problem



Can A Neural Network Model a Polynomial?

- From the manner of training a Neural Network, they can
  - Interpolate data given the training data.
  - Can they model a polynomial?
- How can we mathematically define a polynomial given specific data?

Collocation Method of Polynomials

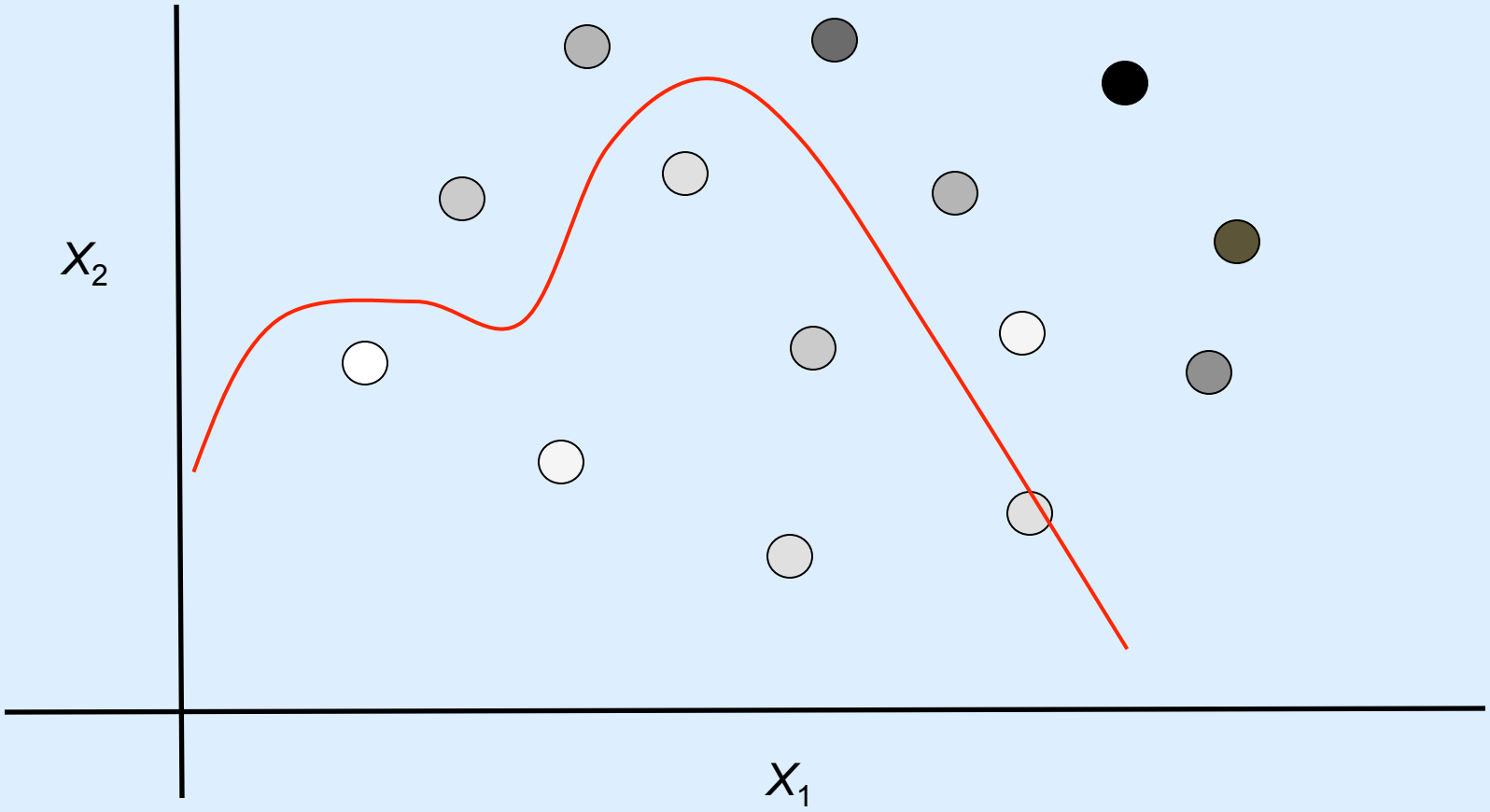
To define an *n*-degree polynomial function $P_n(x)$ that goes "through" a specified set of points (x,y), define

$$P_n(x) = \sum_{i=0}^{n} f_i L_i(x)$$

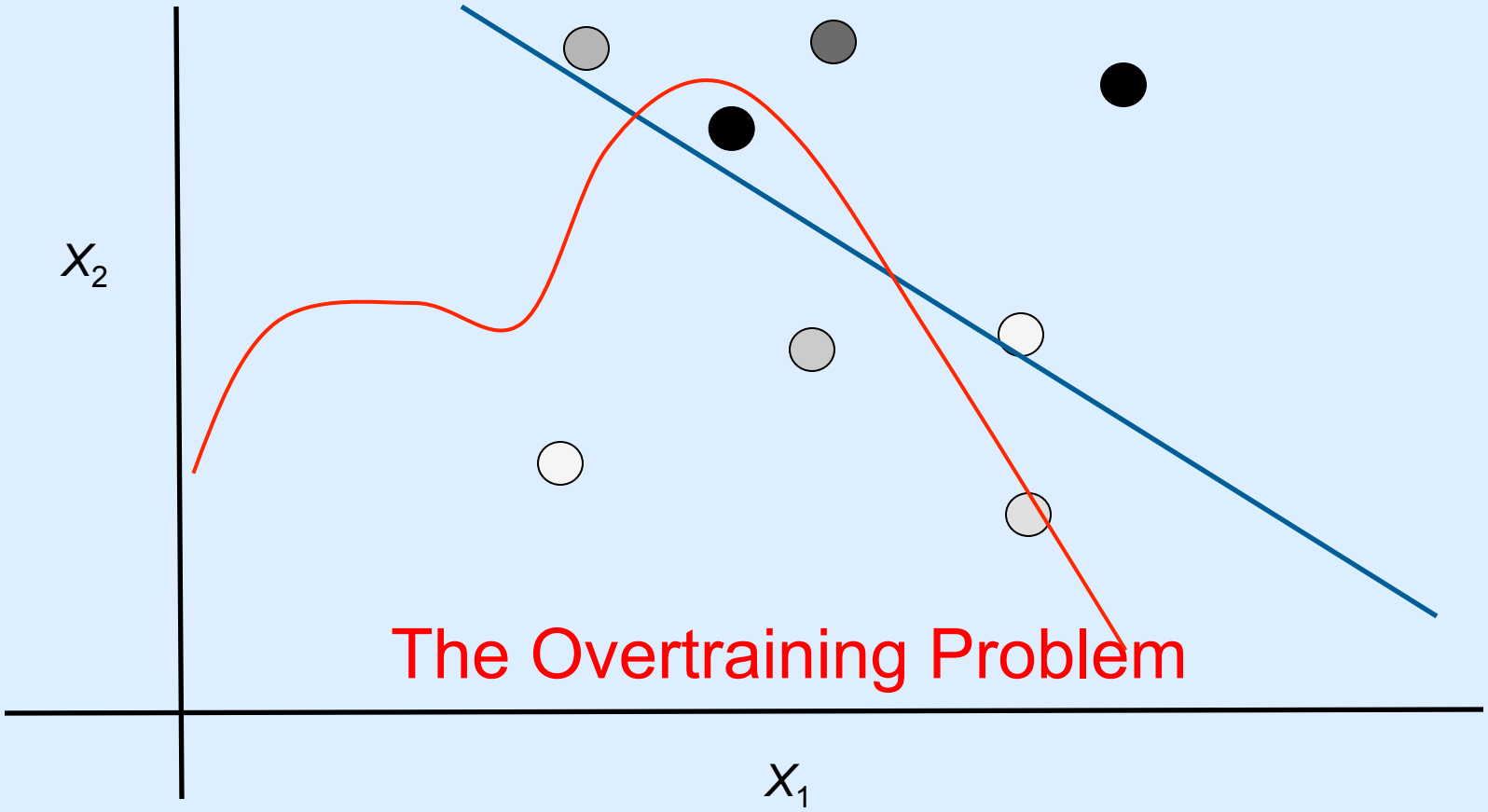
where

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

A Classification Problem



A Classification Problem



Training: Supervised Learning

- Want to use a neural network to handle data in some future scenario.
- Need to train the network with data we already have to do this.
- But we also need to assess how well the network works.
- Obviously, we can't use the same data we trained the network with to assess its performance. Why?

Training: Supervised Learning

- A common approach is to minimize errors in a FFBP setup.
- Obtain raw data:
  - parts corresponding to inputs,
  - parts corresponding to associated outputs.
- Divide the data into two parts: Training Data and Testing data.
- Take the training data and train and train and train, (but not too much) until errors are as small as possible.
- Then test the trained network using the testing data set to see how well it works using various performance criteria.
- Then make adjustments as necessary.

Partitioning Data Sets and Statistical Analysis

- Many ways to partition data
  - Just once as in the following example, or
  - Many times using different 'partitions' and then average the results (the weights) from each partition.
- Many statistical methods do this type of partitioning. Popular methods include:
  - Bootstrap Method
  - Jacknife Method
- Basically idea is to use the same data but create many training/testing sets. Randomly select out the training set and the testing set. E.g. for 20 I/O pairs, using 10 I/O pairs for training, 10 I/O pairs for testing. But

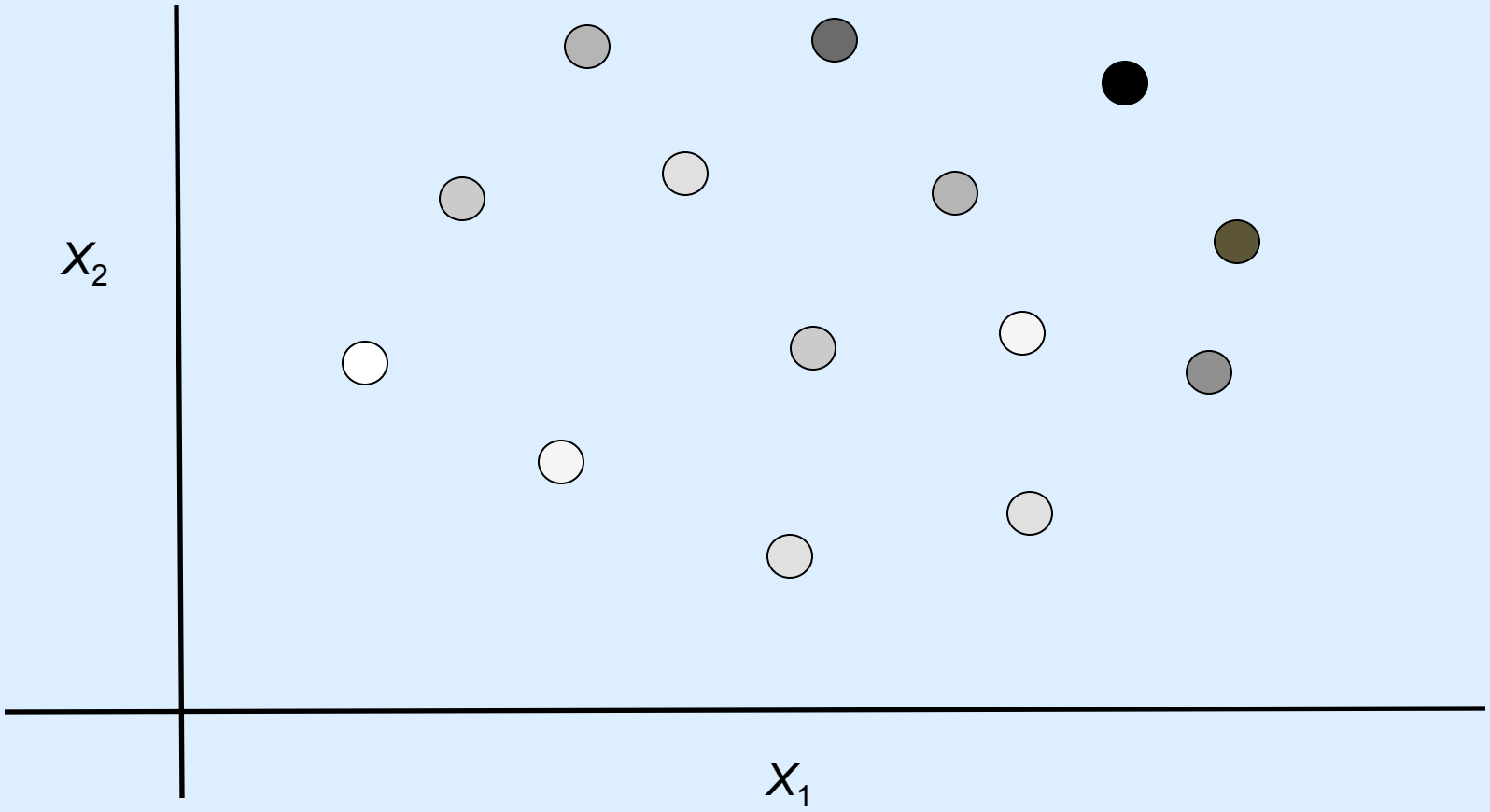
$$\binom{20}{10} = \frac{20!}{10!10!} = 184,756$$

This Scenario Requires

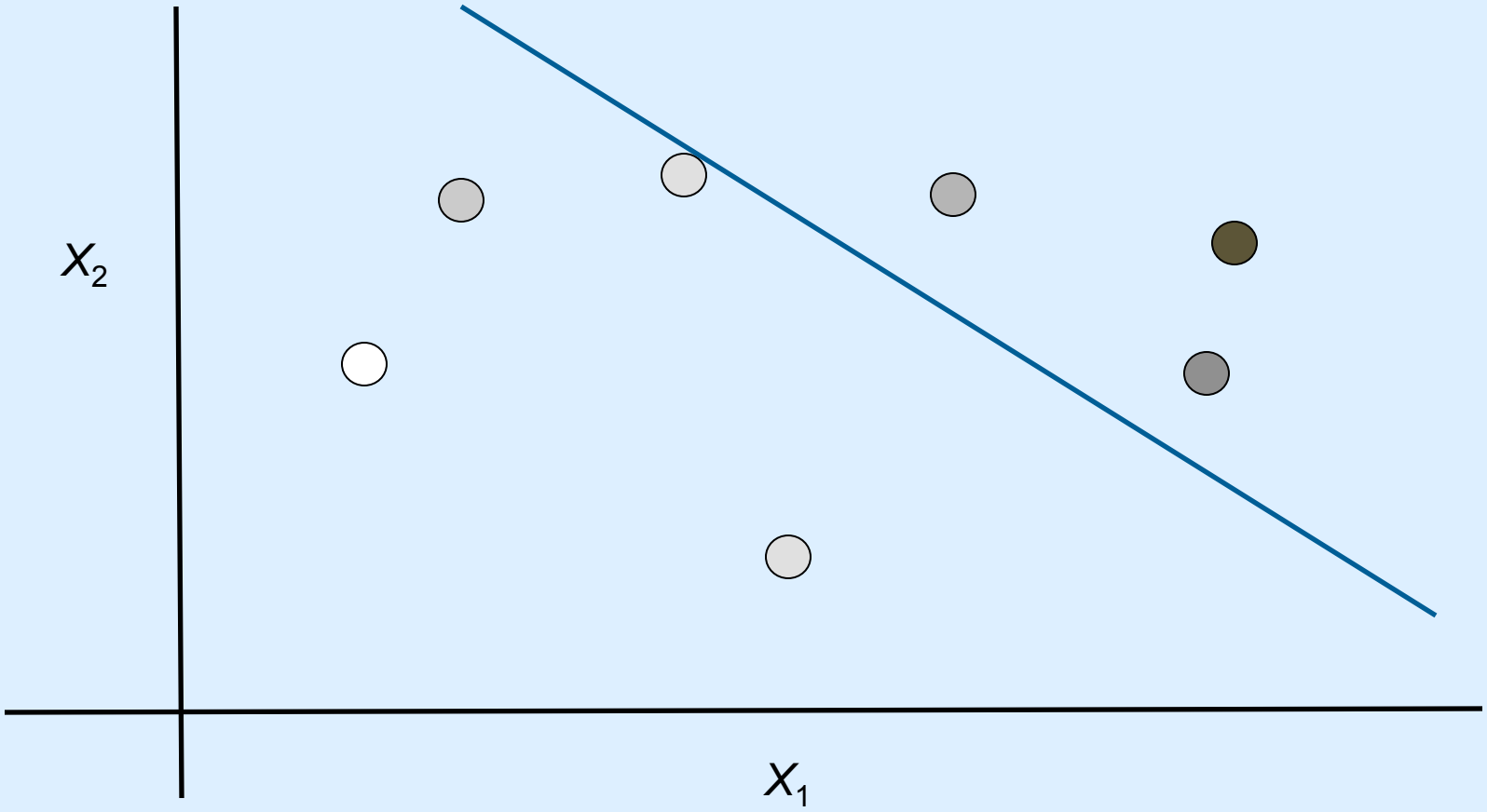
- A means for evaluating network performance (obviously).
  - Performance refers to how well the neural network performs the desired function for which it is being trained.
  - We will, for now, assume some method of measuring performance and get into the details in later sub-modules.
- Also means being able to use the neural network and associated data effectively and efficiently!
  - Computational efficiency is important for training AND assessing performance.

Efficiency

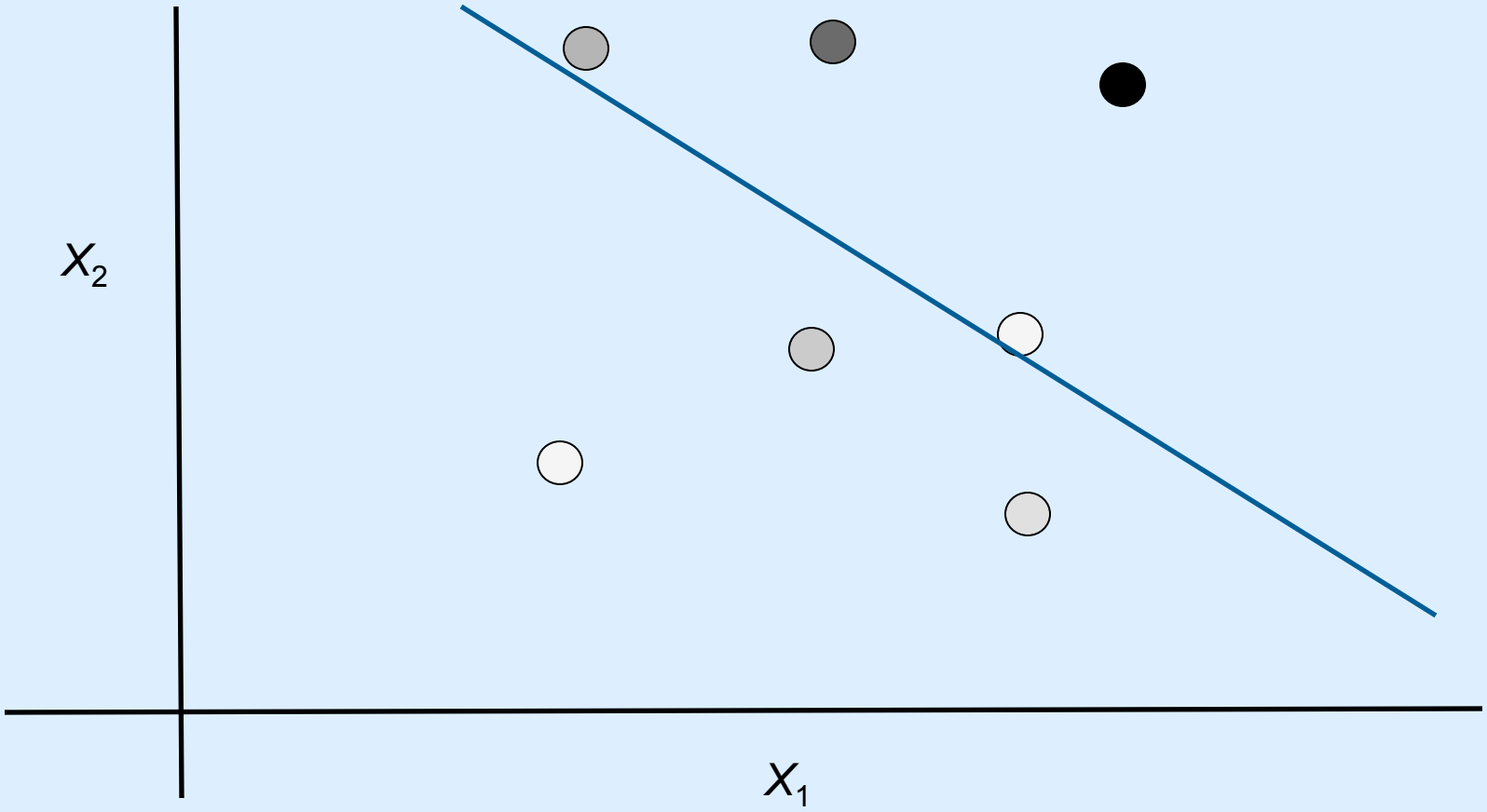
A Classification Problem



A Classification Problem

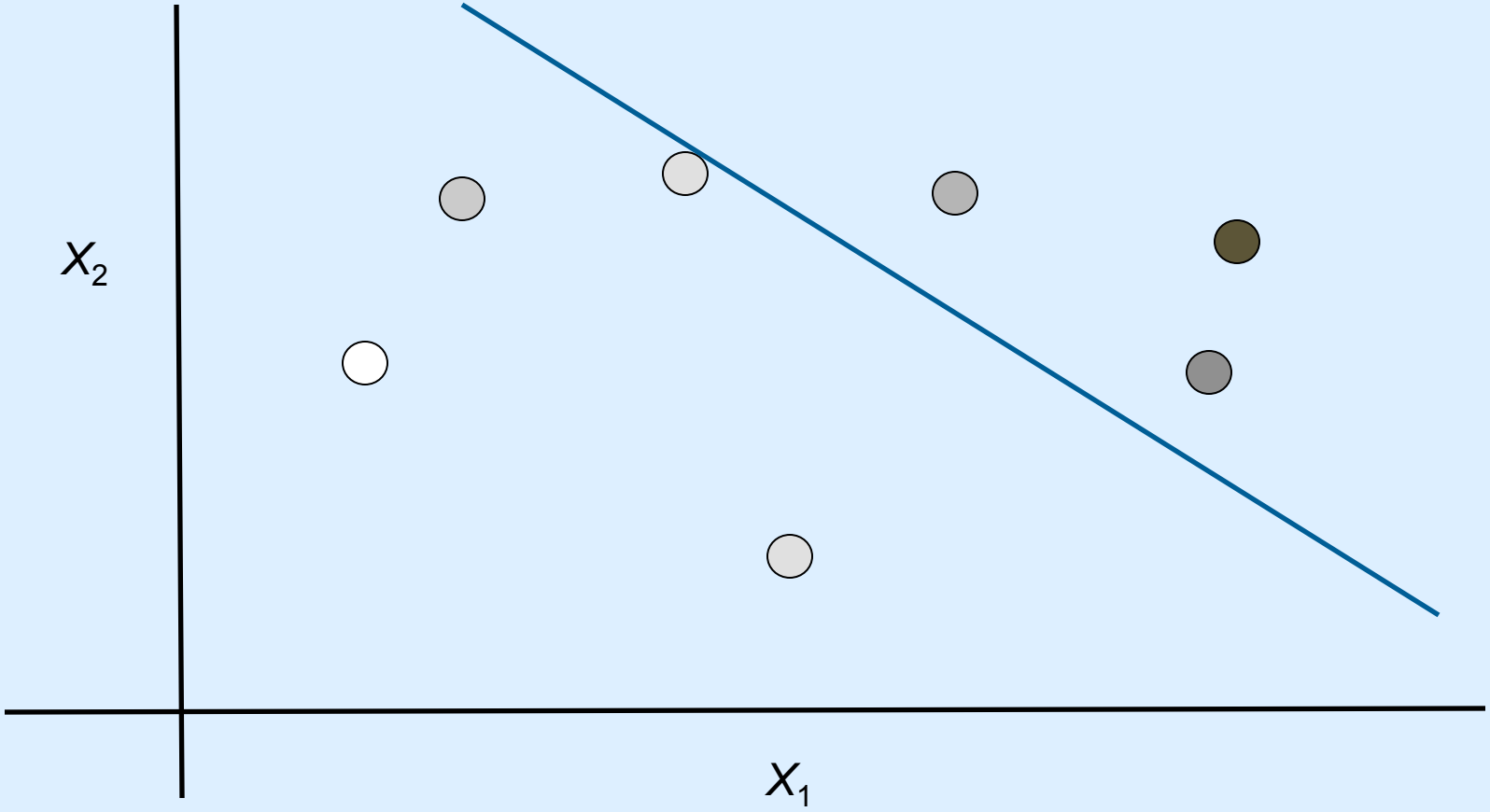


A Classification Problem

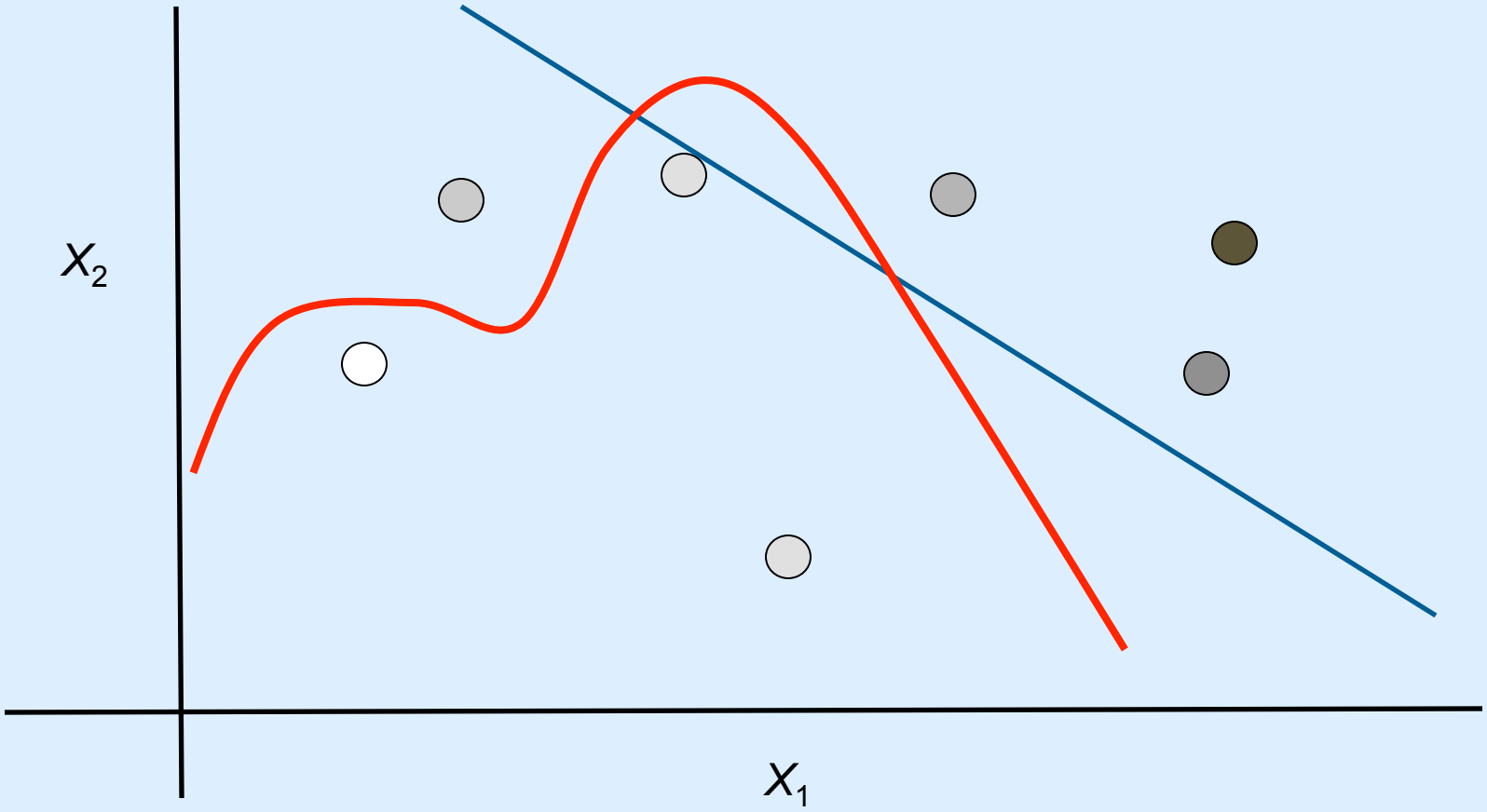


A Classification Problem

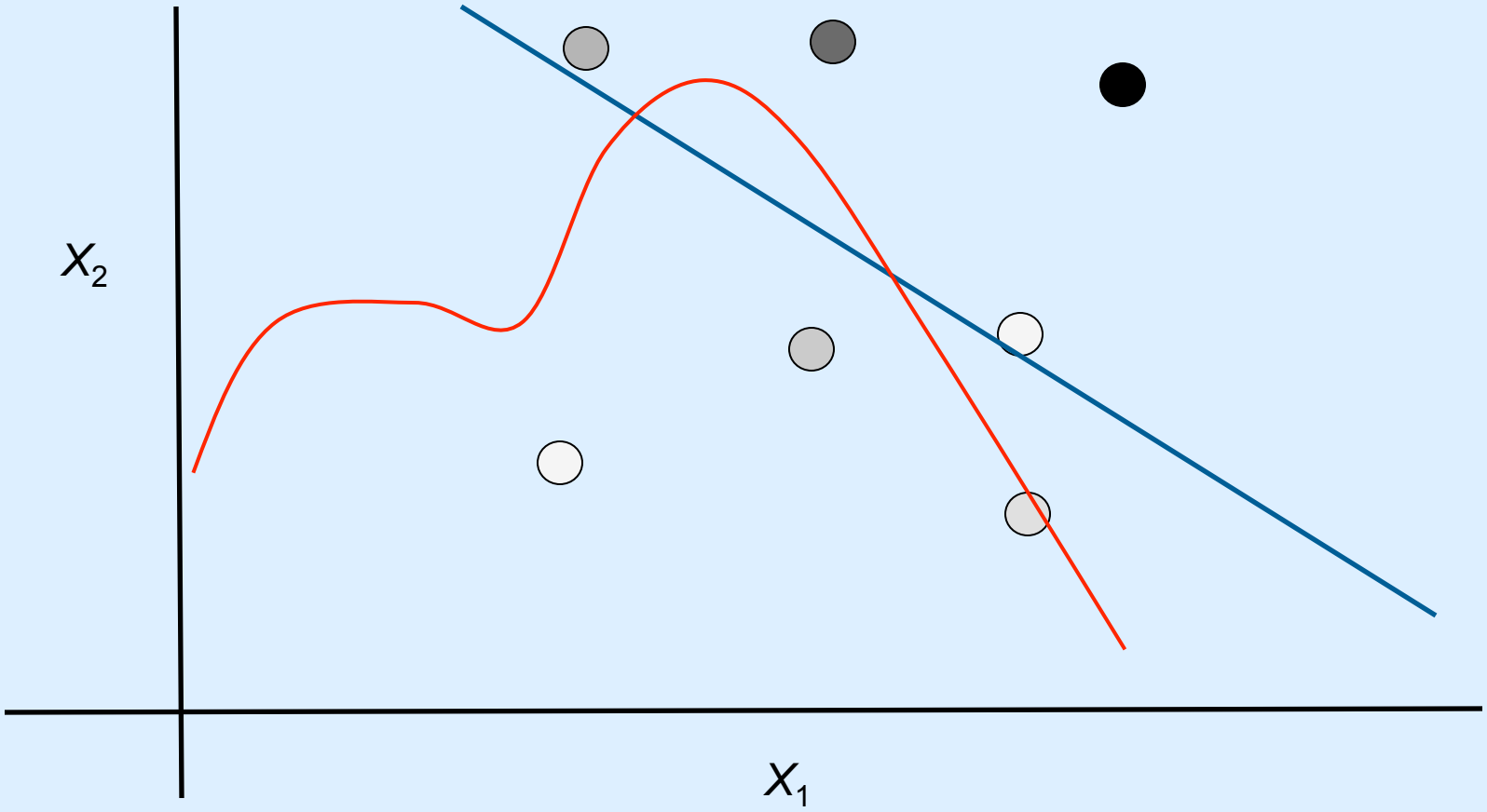
Back to Training Set



A Classification Problem



The Overtraining Problem



The Overtraining Problem

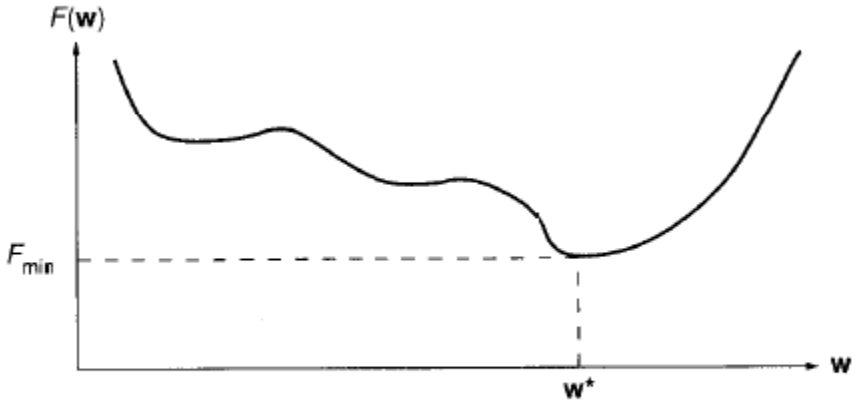


Fig. 5.1. • A typical error surface. A goal of neural network training is to find a network weight vector $\mathbf{w}^*$ that minimize the error F.

From *Neurocomputing*
by Hecht-Nielsen

The Overtraining Problem

- Random variations may be part of some underlying process.
- A Regression Line estimates the deterministic part of the process.
- Training attempts to capture this process.
- Training too much may end up including the effects of random variation in the deterministic part of the modeling of a process.

We end up training the network on the randomness instead of the underlying deterministic process!

Training on the errors.

Training Reprise

- During training, we use the error to compute the gradient in steepest descent methods, to guide us towards minimizing the error.
- During testing/evaluation, we use the error to minimize the testing error.
- Must do a lot of experimentation: training, testing, training, testing…
- A lot of computational effort is involved to get it right!

Training Efficiency

- How can we maximize use of our data?
- How do we decide how many times to train and test? How many different partitions to use?
- Efficient and effective training and testing methods must be employed.

Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program

605-447/625-438

Dr. Mark Fleischer

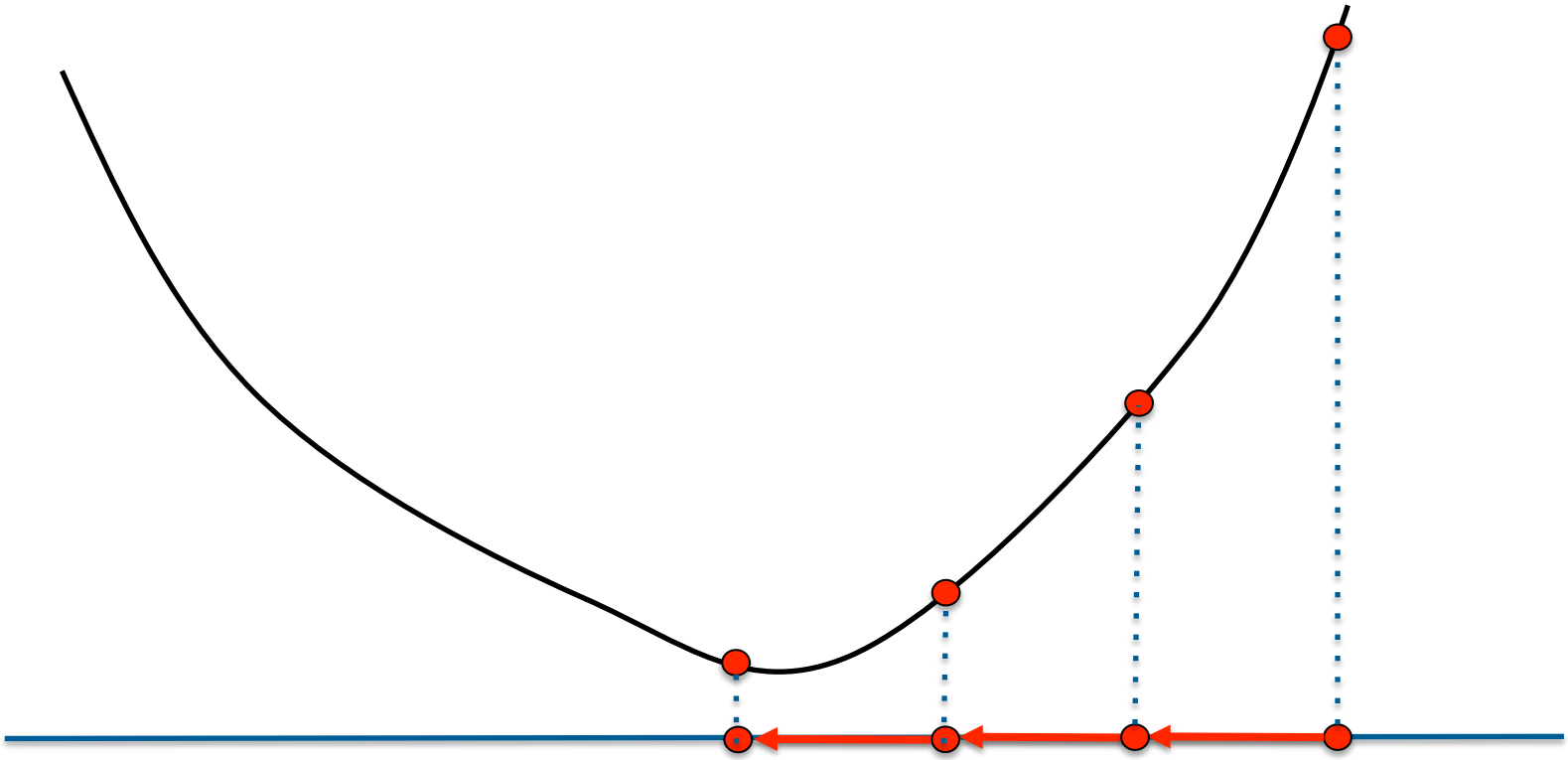
Module 7.2: Training Efficiency

This Sub-Module Covers …

- We know that in training and testing, there can be a high computational burden.
- Want to make training more efficient.
  - Explore ways for speeding up the FFBP algorithm with a momentum term;
  - Explore some gardening techniques (pruning techniques) to decrease the network size.

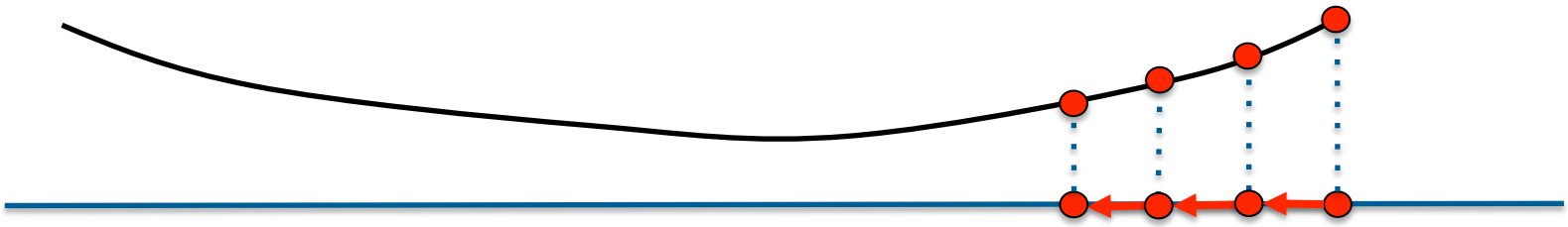
Training: Method of Steepest Descent

The problem with 'fixed' step sizes: $w_{k+1} = w_k - \eta \nabla f\left(w_k\right)$



Training: Method of Steepest Descent

The problem with 'fixed' step sizes: $w_{k+1} = w_k - \eta \nabla f\left(w_k\right)$



Issues with Steepest Descent Methods

- Can take a long time to find a local minimum when the step size η is too small.
- Can miss or overstep where the local minimum is when the step size η is too large and even lead to oscillations or meandering around the minima.

Getting to Minima Faster

- Some approaches can be proven to converge to the minima:
  - Simulated annealing, but there are issues in continuous variable problems.
  - Stochastic approximation methods.
  - Nelder-Meade …
- The No-Free Lunch Theorem applies!

The Momentum Term

- Strikes a good balance between ease of implementation and computational overhead and speeding up the process.
- Based on using historical information in the step size.

$$\Delta w_i(k) = -\eta \frac{\partial E}{\partial w_i} + \alpha \Delta w_i(k-1)$$

The Momentum Term

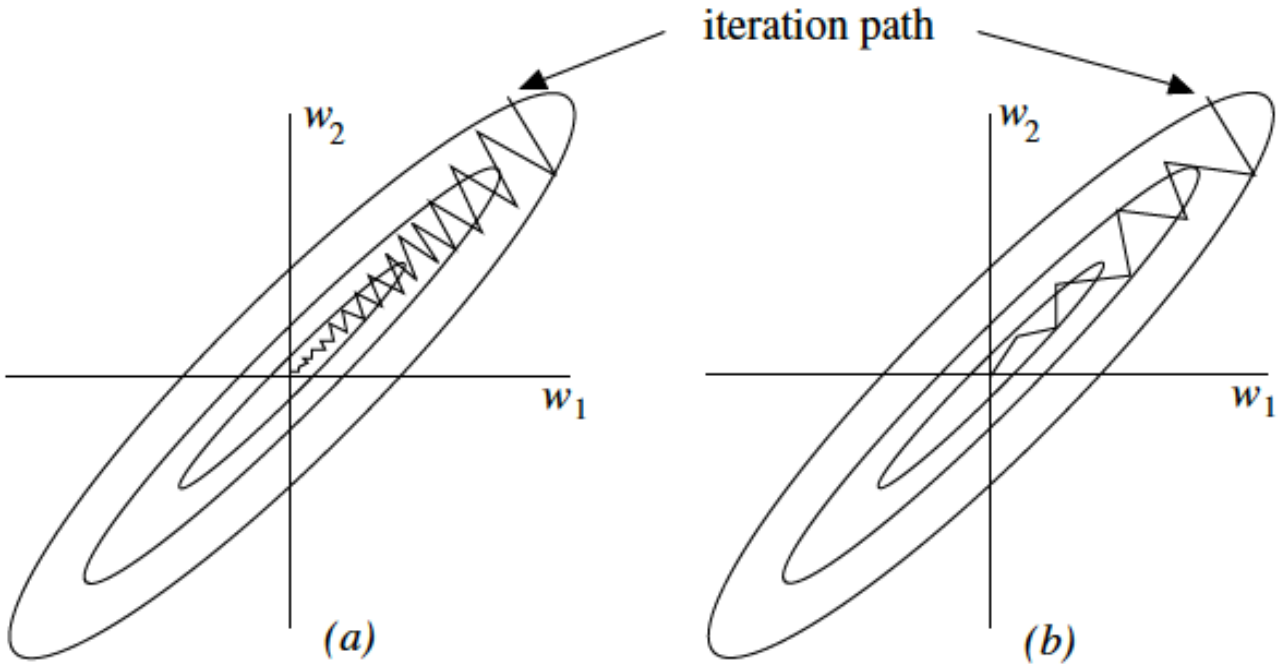


Fig. 8.1. Backpropagation without (a) or with (b) momentum term

From Rojas, p.186

Other Improvement Approaches

- Training and evaluating a network can take a very long time to get it right (we'll see what 'right' means later).
- We have seen how to improve the training algorithm with momentum terms.
- Let's improve training by modifying the network.

Can A Neural Network Model a Polynomial?

- From the manner of training a Neural Network, they can
  - Interpolate data given the training data.
  - Can they model a polynomial?
- How can we mathematically define a polynomial given specific data?
- What does this have to do with training efficiency?

Collocation Method of Polynomials

To define an *n*-degree polynomial function $P_n(x)$ that goes "through" a specified set of points (x,y), define

$$P_n(x) = \sum_{i=0}^{n} f_i L_i(x)$$

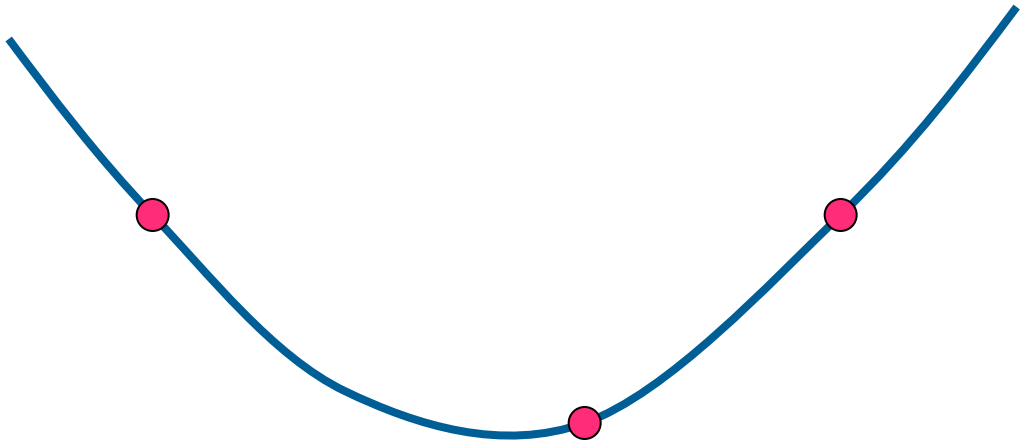
where

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

Example

Say we want a polynomial to go through specific points in some space. For three points . . .

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$



Remembering our function definitions:

$$P_n(x) = \sum_{i=0}^{n} f_i L_i(x) \qquad L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

then

$$P_n(x) = f_0 \underbrace{\left[\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \right]}_{1} + f_1 \left[\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \right] + f_2 \left[\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \right]$$

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$

Remembering our function definitions:

$$P_n(x) = \sum_{i=0}^{n} f_i L_i(x) \qquad L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

then

$$P_n(x) = f_0 \left[\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \right] + f_1 \left[\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \right] + f_2 \left[\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \right]$$

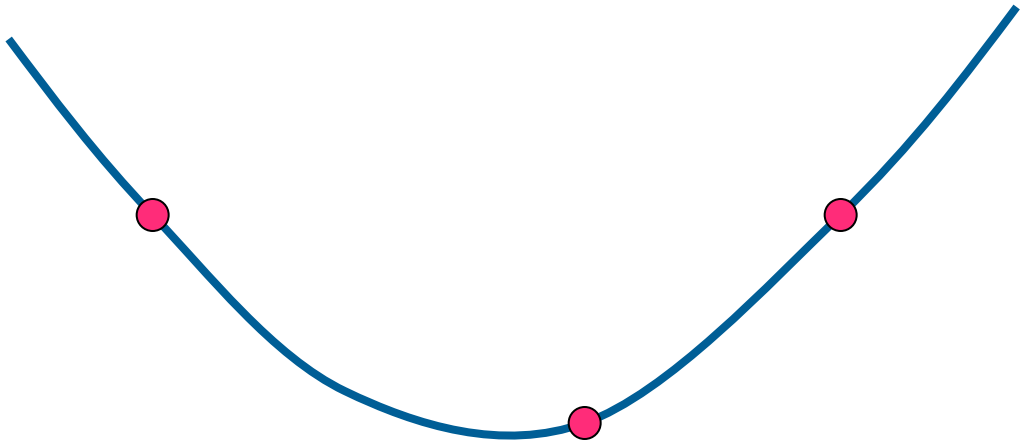
1

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$

Example

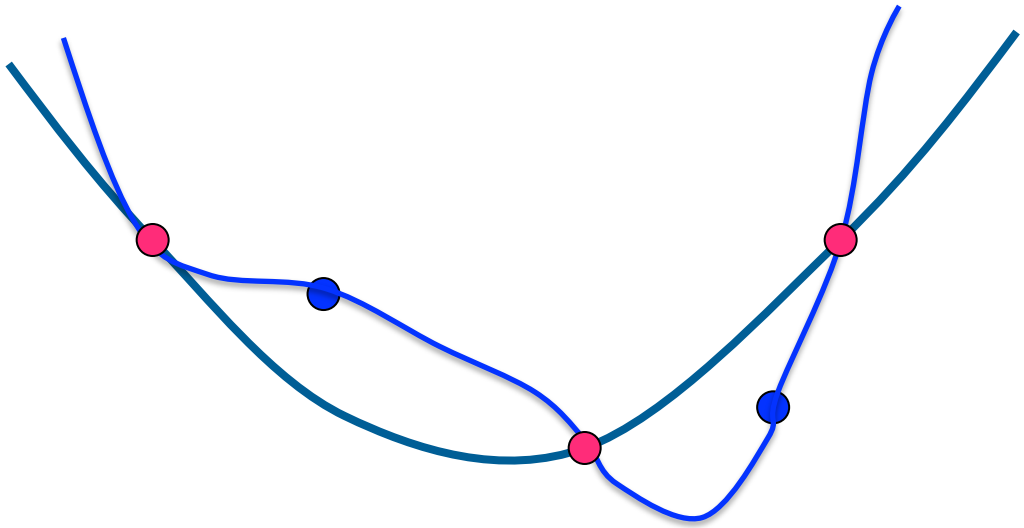
Say we want a polynomial to go through specific points in some space. For three points . . .

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$



Example

Now let's add some "dummy" points.



Remembering our function definitions:

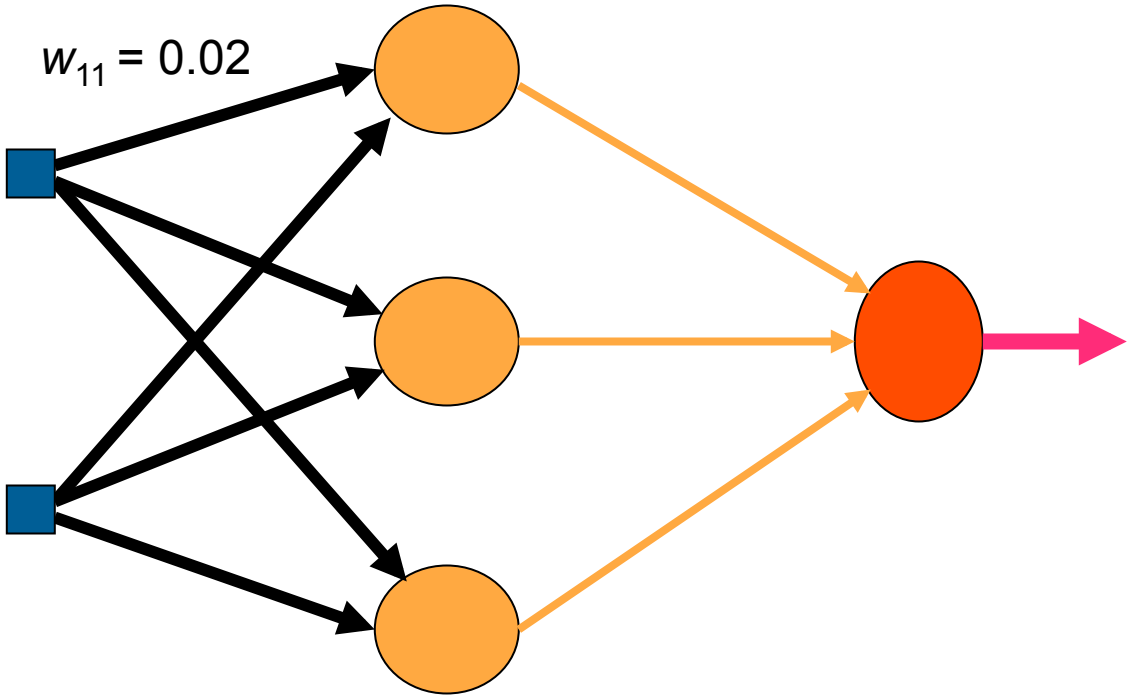
$$P_n(x) = \sum_{i=0}^{n} f_i L_i(x) \qquad L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

then

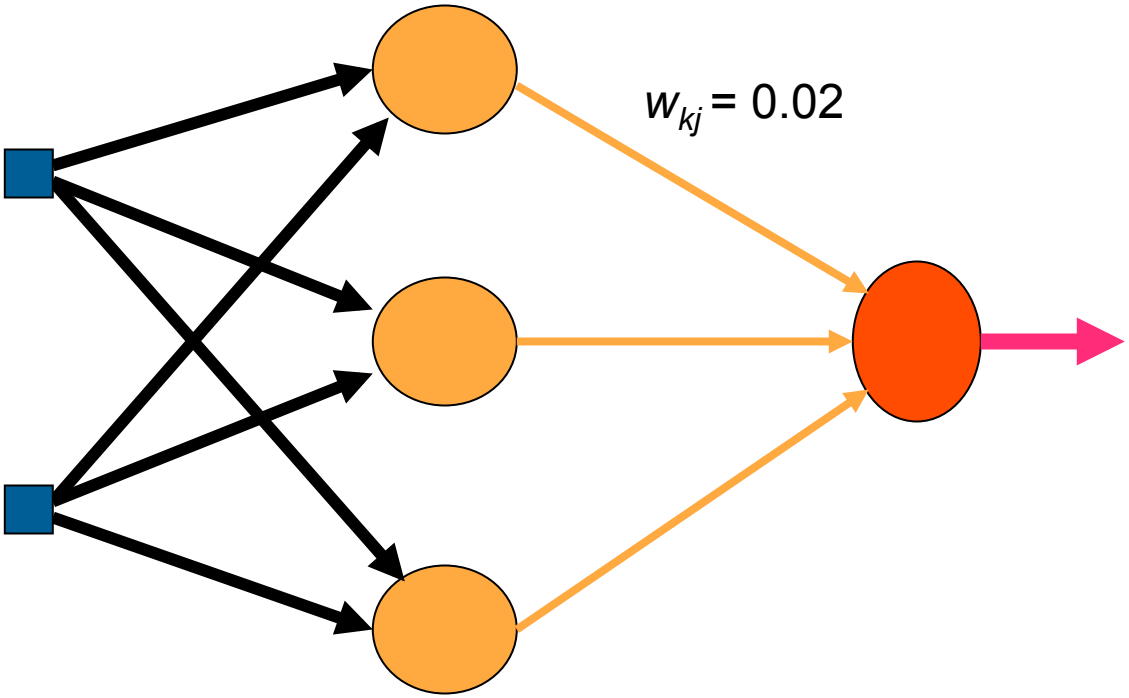
$$P_n(x) = f_0\left[\frac{(x-x_1)(x-x_2)(x-x_3)(x-x_4)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)(x_0-x_4)}\right] + f_1\left[\frac{(x-x_0)(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)(x_1-x_4)}\right] + f_2\left[\frac{(x-x_0)(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)(x_2-x_4)}\right] + f_3\left[\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)(x_3-x_4)}\right] + f_4\left[\frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_0)(x_4-x_1)(x_4-x_2)(x_4-x_3)}\right]$$

Well, you get the idea… complications, complexity, etc.

Pruning The Network



Pruning The Network



Summary – 7.2

- Described using the momentum term to speed-up gradient descent optimization (i.e., FFBP).
- Collocation Polynomial and how to define a polynomial function that passes through a specified set of points.
- Pruning the network to eliminate ‘weak’ links and/or nodes that do not carry much weight.

Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447/625-438
Dr. Mark Fleischer

Module 7.3: Training with Multiple I/O Pairs

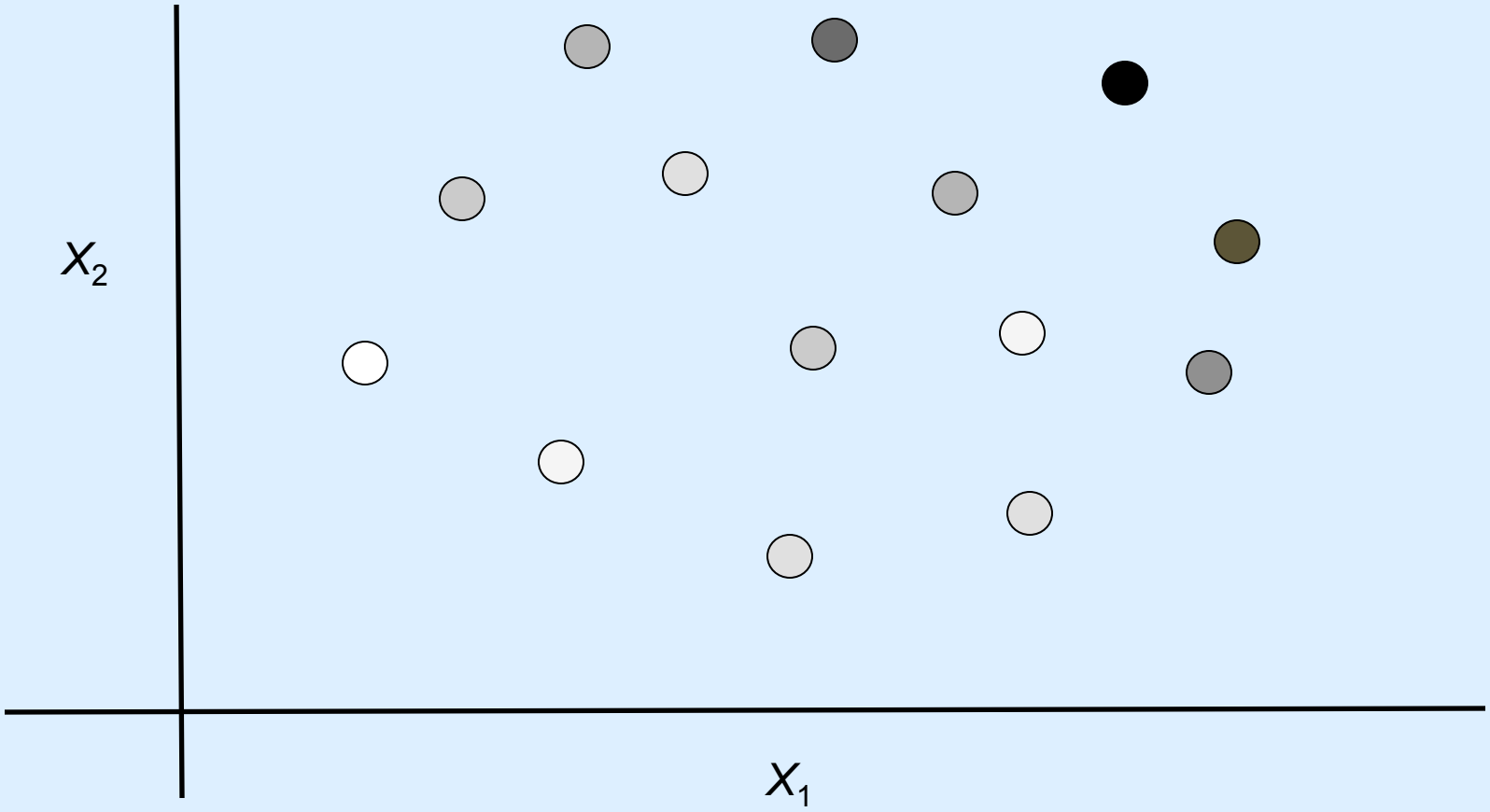
This Sub-Module Covers …

- What a typical training scenario looks like when there are multiple input/output pairs.
  - "online" or "incremental" training
  - "batch" training
- Implications and why they are different.

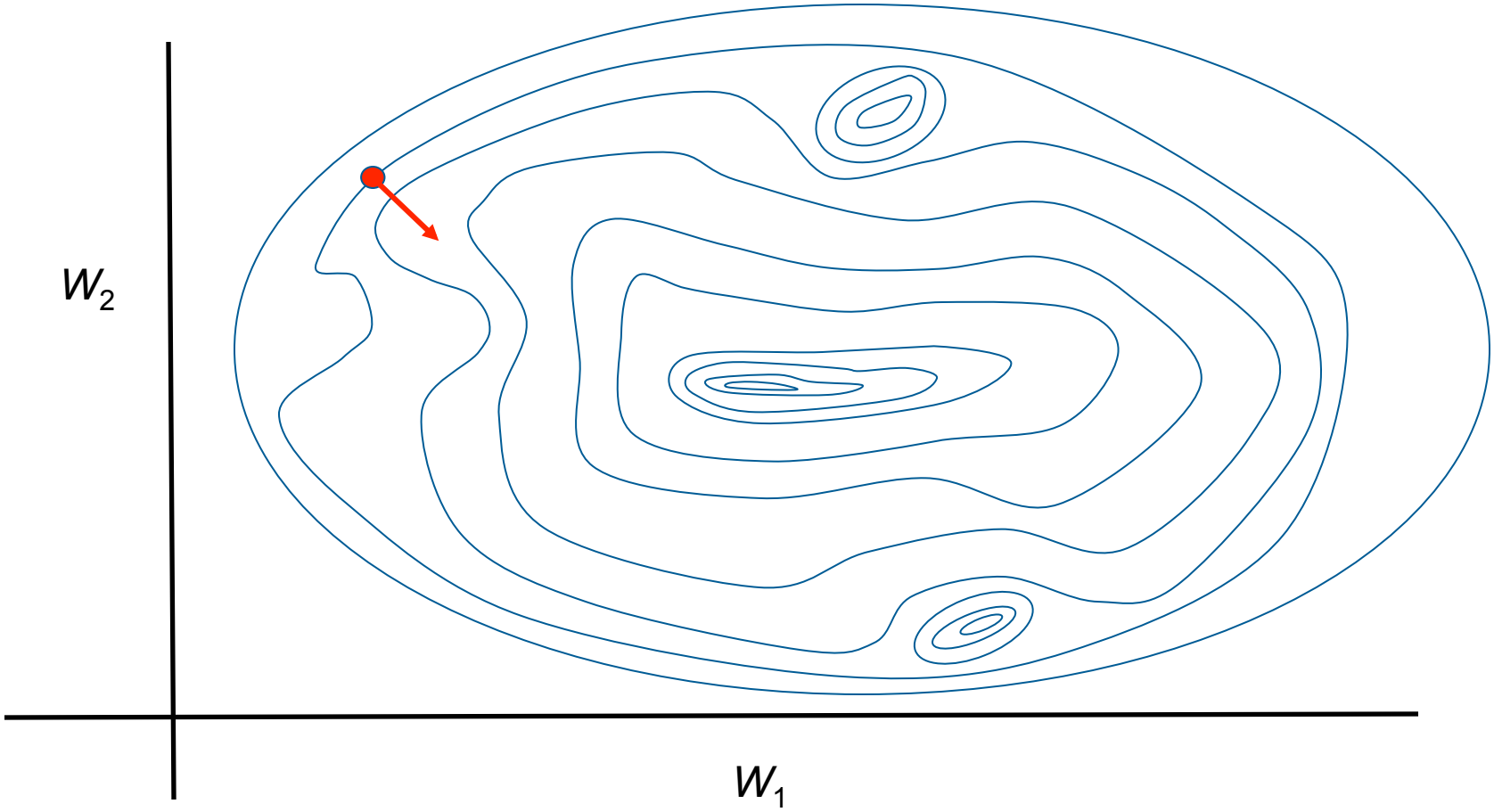
Training with Multiple Input/Output Pairs

- We've seen we can decrease errors with a single I/O pair with the Perceptron.
- Can do the same with multilayer networks.
- Still, with FFBP algorithm, training can get stuck in local optima---with one I/O pair we can get error as low as desired.
- Let's examine a type of problem with many I/O pairs.

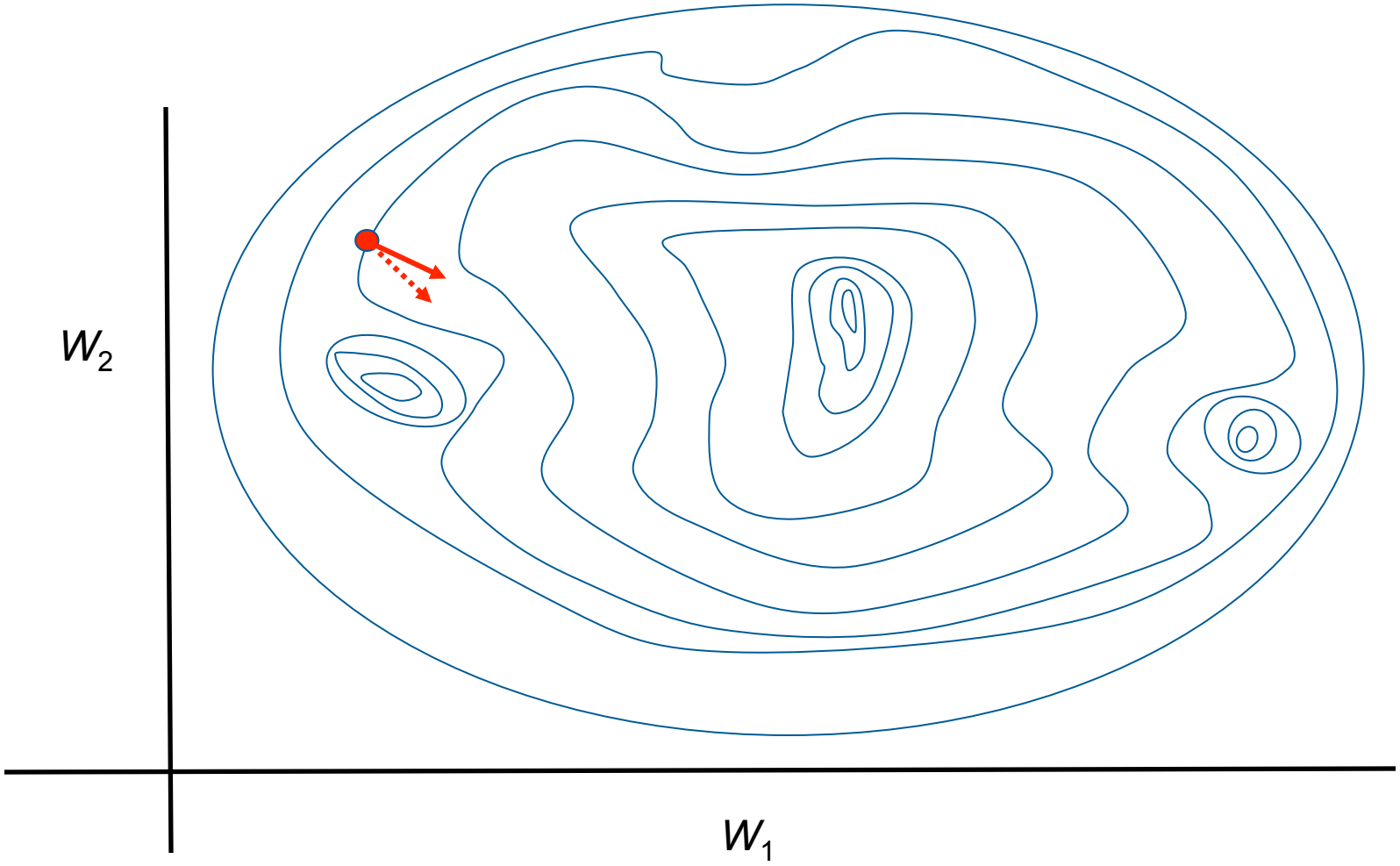
A Classification Problem



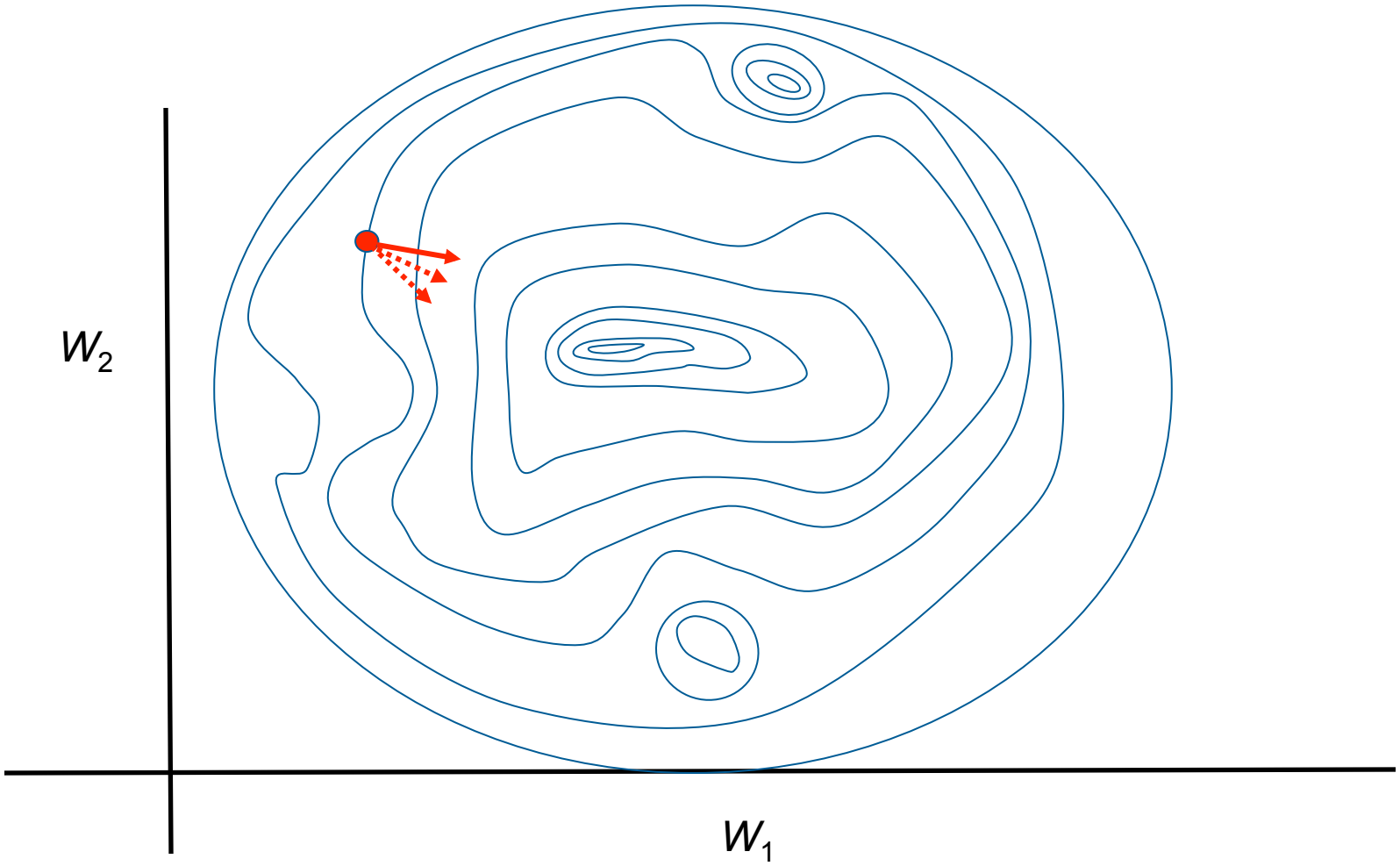
Present 1st Input/Output Pair



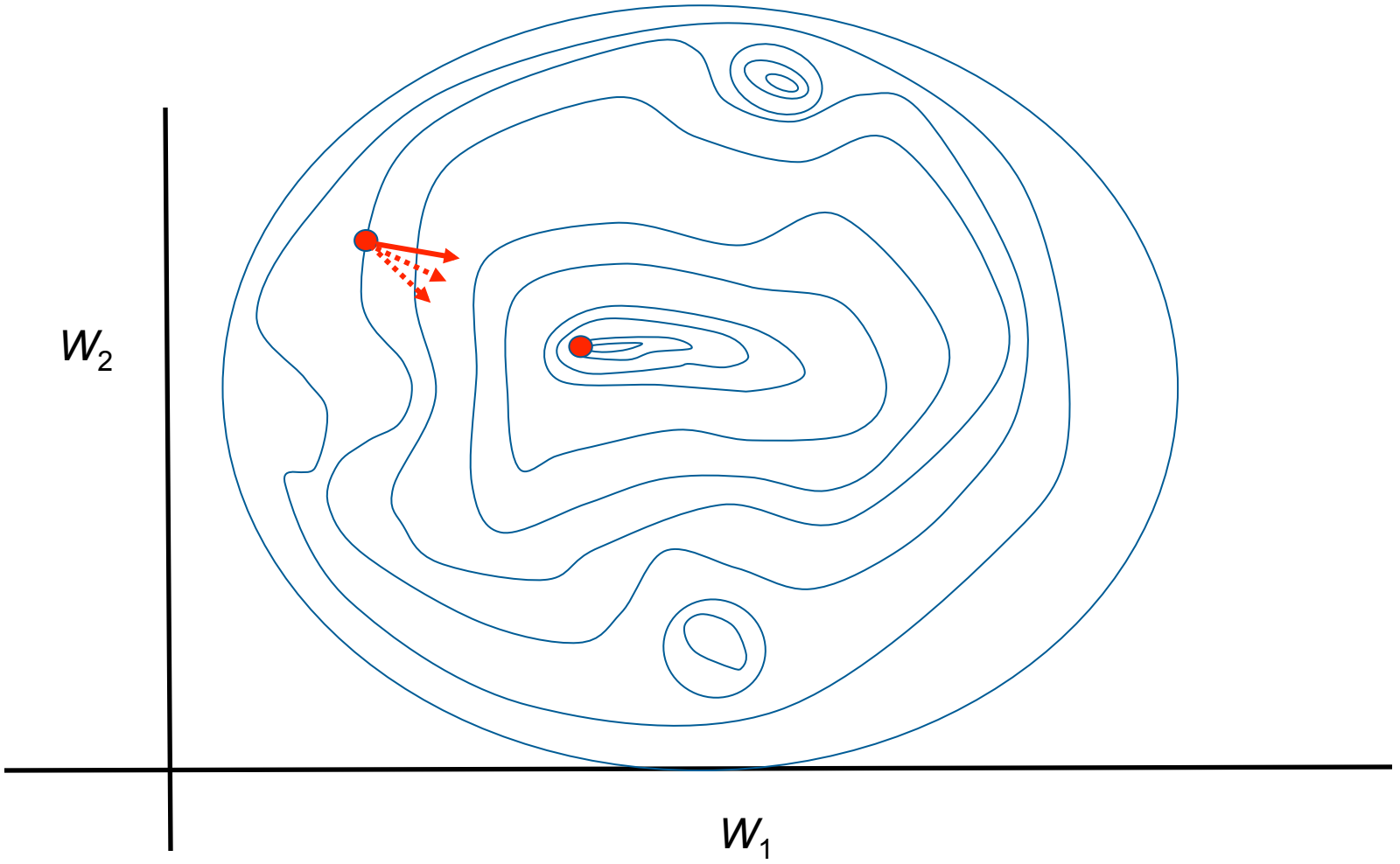
Present 2nd Input/Output Pair



Present 3rd Input/Output Pair



Updating Weights



Batch Training

Training cycle (c = 1 to C){

$\Delta w == 0$ // Initialize weight update

Present I/O pair (p = 1 to P) {

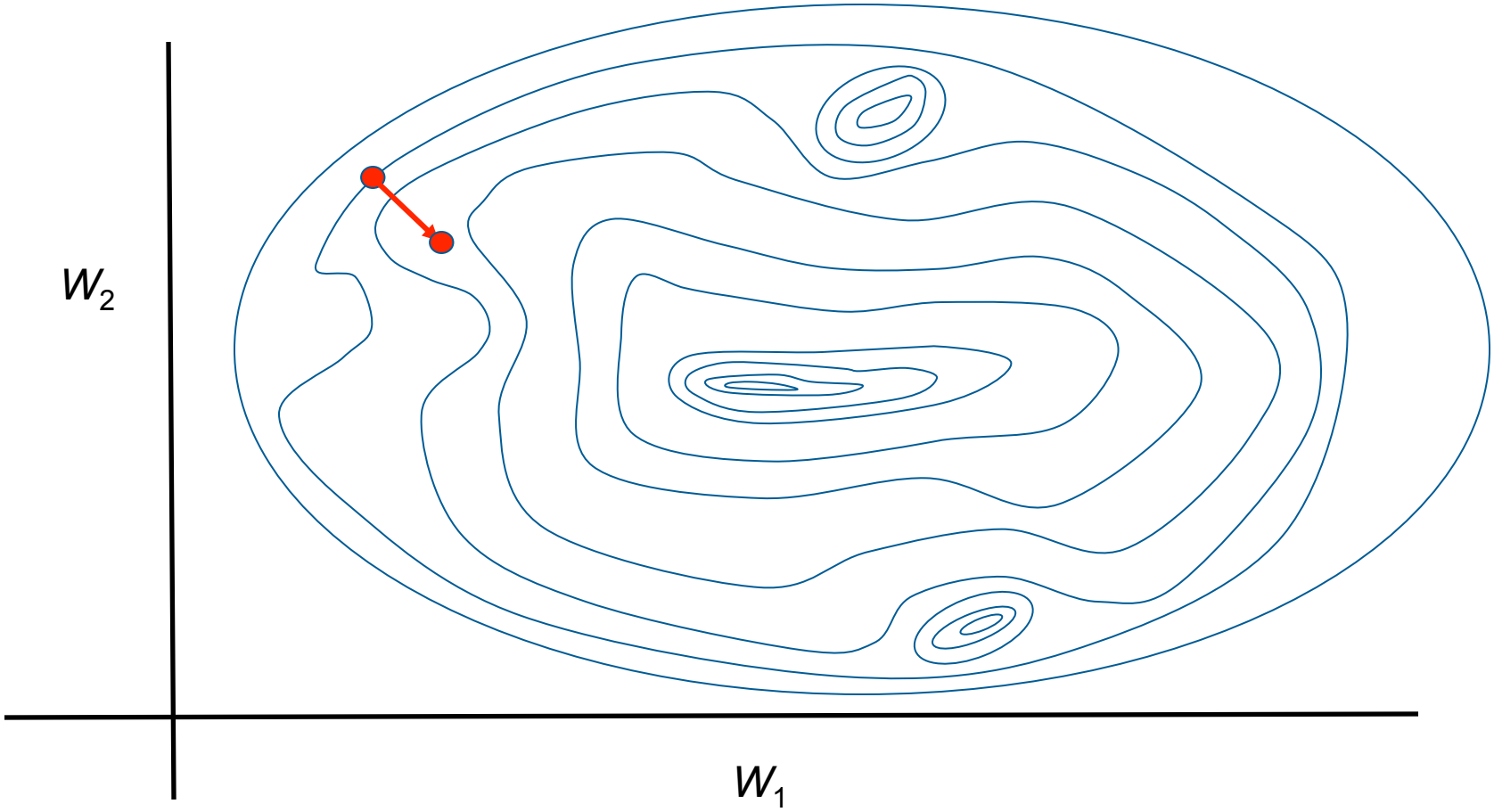
Δw += Calculate Gradient estimate();

} // Present next I/O pair

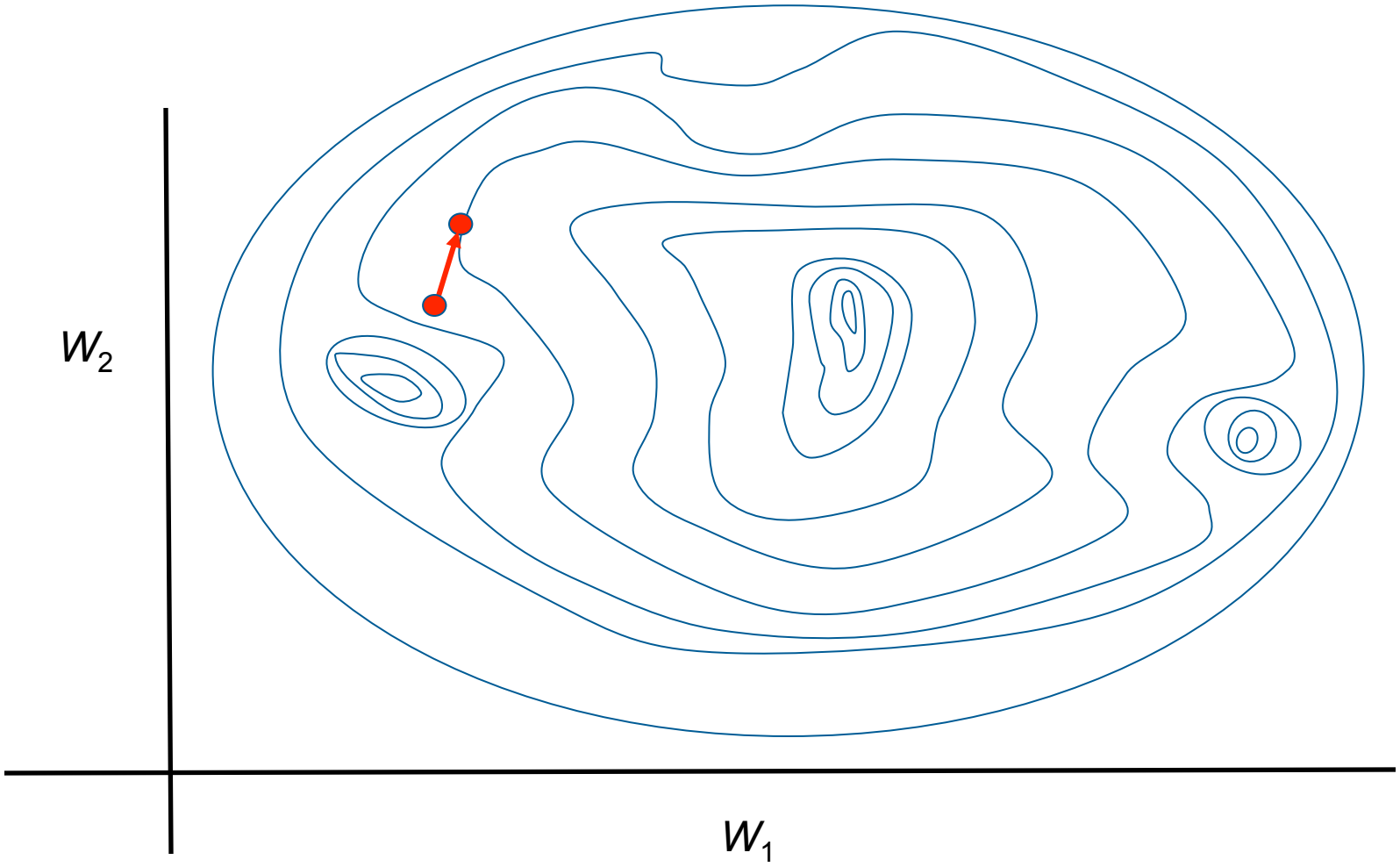
Update weight $w = w + \Delta w$

} // Next training cycle

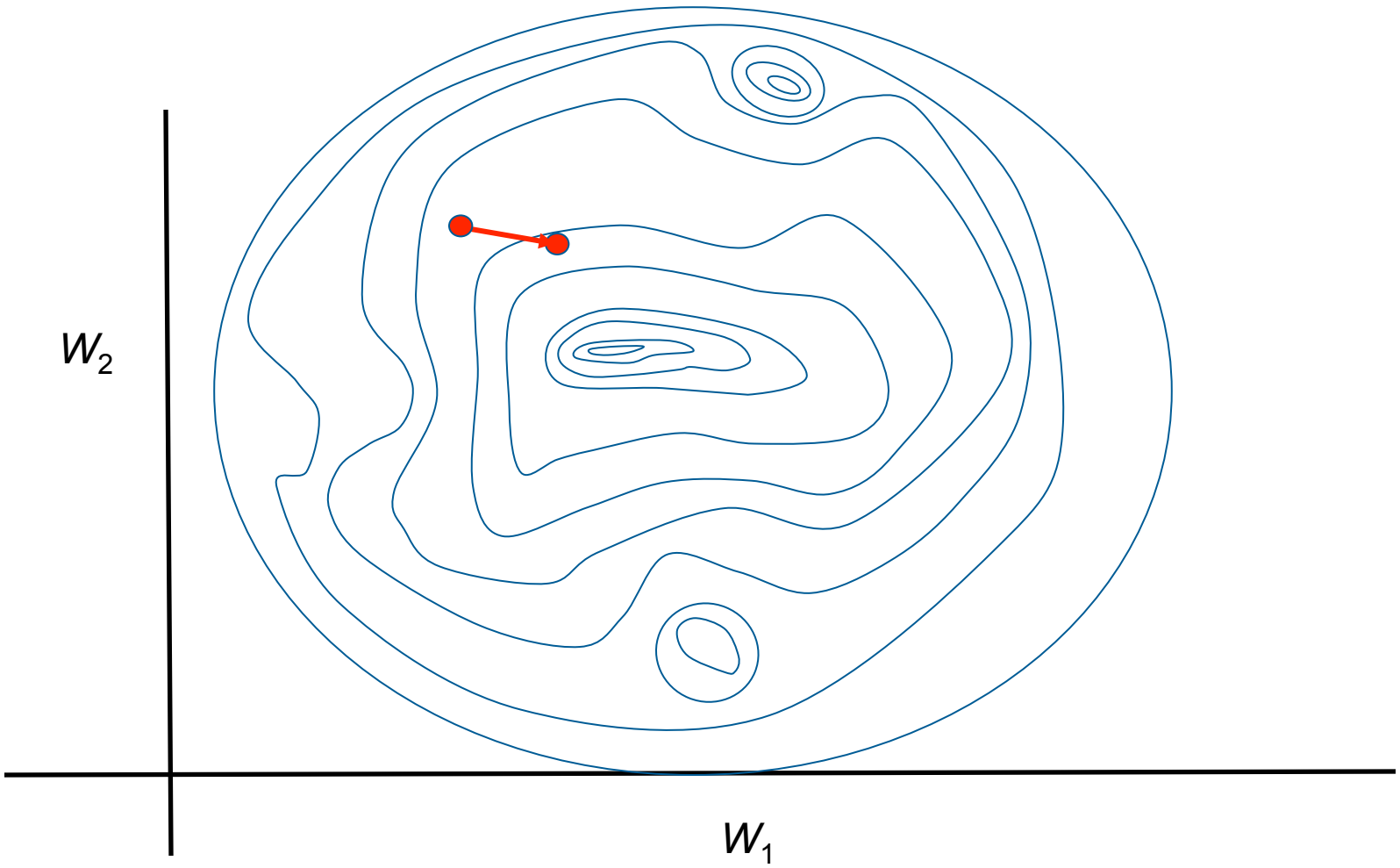
Present 1st Input/Output Pair



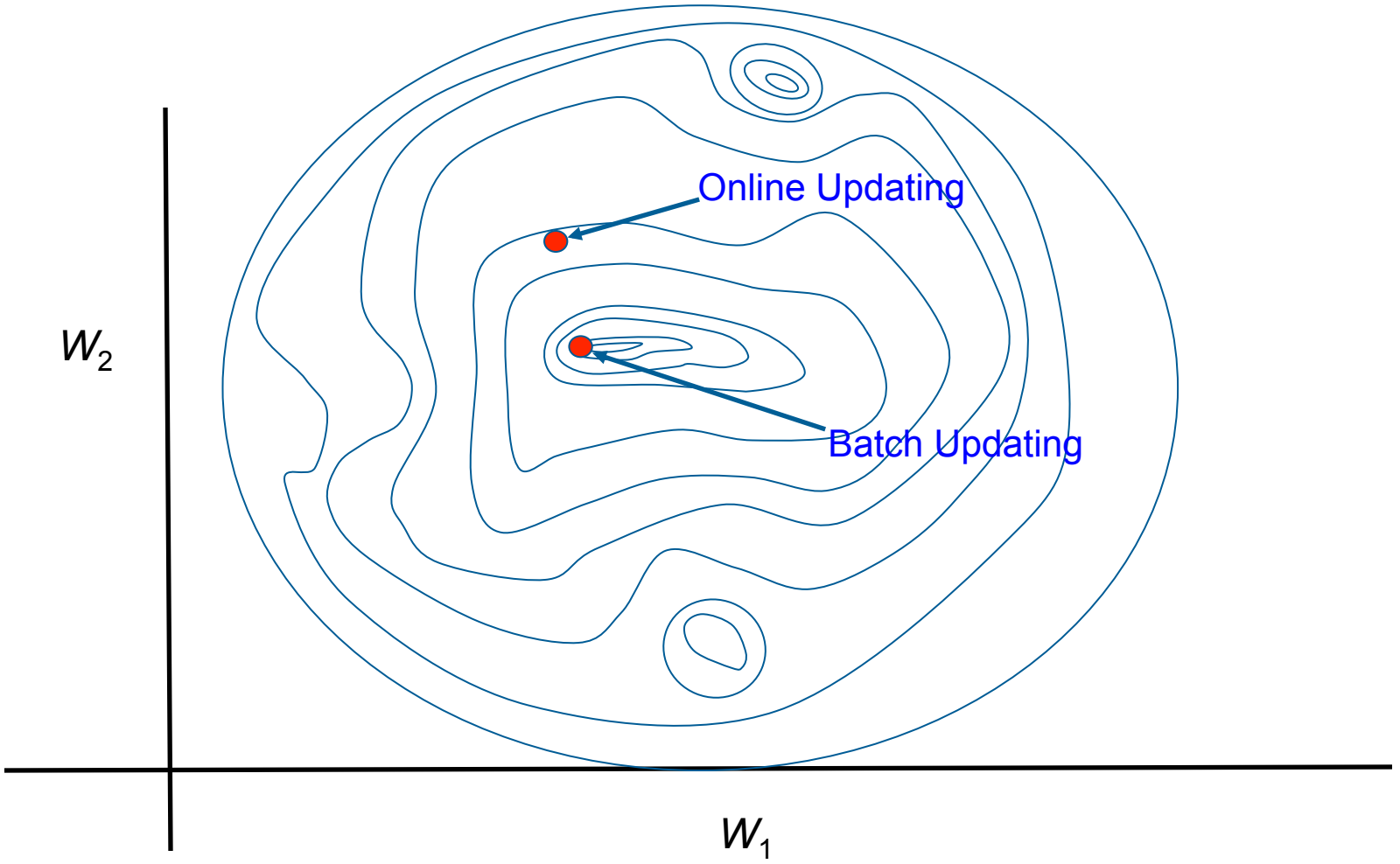
Present 2nd Input/Output Pair



Present 3rd Input/Output Pair



Comparing Updated Weights



Online (Incremental) Training

Training cycle (c = 1 to C){

Present I/O pair (p = 1 to P) {

Δw = Calculate Gradient estimate();

Update weight w = w + Δw

} // Present next I/O pair

} // Next training cycle

Batch vs. Online Training

Batch

Pros:

- Closer match to theoretical (true) gradient.
- Lower residual error.
- Permits other, useful modifications of FFBP to be used.

Online

Pros:

- Faster, earlier updates (but remember, each update changes the error function for subsequent I/O pairs).
- Useful if no ‘fixed’ training set exists or when the data are ‘nonstationary’.
- Creates effective/useful ‘noise’ that can overcome local minima.

Batch vs. Online Training

Batch

Cons:

- Requires some memory overhead.
- Can require more computation effort to get to comparable weights than online (requires all I/O pairs before an update occurs)

Online

Cons:

- Each update alters the error function landscape --- not the true gradient descent vector for all I/O pairs– perturbs the learning process.
- Sometimes noise isn't a good thing.

The Best of Both Worlds

- Use ‘online’ training for ‘initial training’ with a first part of the training data
  - Get a good, fast start with some updated weights.
- Then as we get closer to minimal error, resort to batch training
  - More carefully and accurate zero-in to true minimal error.

Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447/625-438
Dr. Mark Fleischer

Module 7.4: Receiver Operating Characteristics

This Sub-Module Covers …

- Performance evaluation.
- Proper training does not merely involve minimizing the error function using the training data.
- Must have some appropriate way of gauging how well the network performs using data it hasn't seen during the training phase.

Let's Examine Classification Problems

- How well does the network correctly classify an input it has not previously seen?
- Simplest approach is to consider two classes
  - Corresponds nicely to mathematical logic,
  - Corresponds nicely to computer logic,
  - Some action is good/bad, or true/false
- How do we come up with reasonable, mathematically sound ways of evaluating performance in such a context?
- Consider the early days of radar development in WWII.
- Receiver Operating Characteristics.
- Basis is that everyone has their own particular vexations that set them 'off'!

Mandrake, those missiles are coming. Red Alert!

or are they?.....

Our friend Gen. Ripper …

- has an apparent ‘sensitivity’ to communist infiltration, indoctrination, subversion and conspiracies.
- This sensitivity led him to trigger and order …

Receiver Operating Characteristics

- Partition a set into two mutually exclusive subsets:
  - one subset contains entities with a characteristic *C* and the other subset contains contains entities that do not have characteristic *C*.
  - Sets could be a "real target" and "not a target"; or "signal" and "noise"; "Wow" and "no extraterrestrial signal"; "heart disease" and "absence of heart disease", etc. …
- A detector to determine the "detected set" and to indicate which set a detected "signal" belongs to—define set *D*.
  - The detector is not perfect---won't give the correct information all the time.
  - i.e., sometimes it'll 'detect' things that you don't want it to 'detect'.
  - Other times, it'll detect something (send a signal) when it isn't truly warranted.
- Partition D into two mutually exclusive subsets: Entities that are detected and entities that are not detected.

Assessing the Most Basic Classification

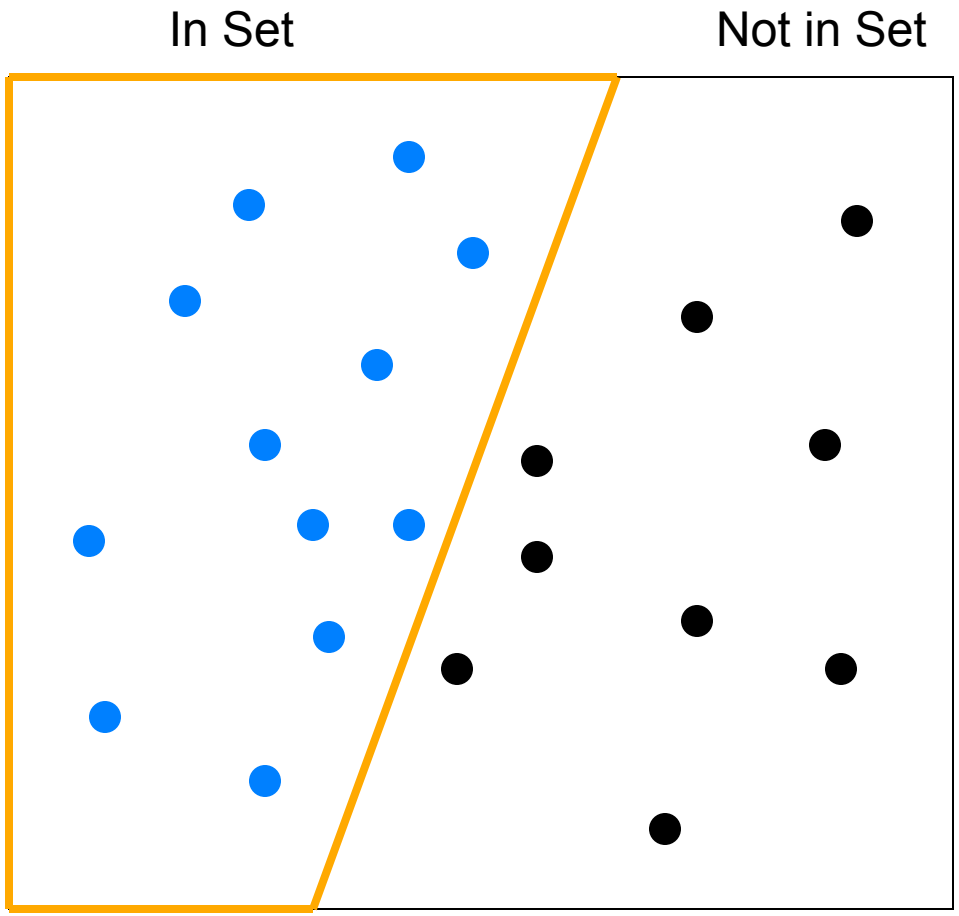
- Either an input produces the desired response, or it does not.

Duh!

Say we want to “detect” only certain types of ‘signals’.

Don’t want to detect signals that are not of this ‘type’.

Receiver Operating Characteristics



The Purpose of Detection

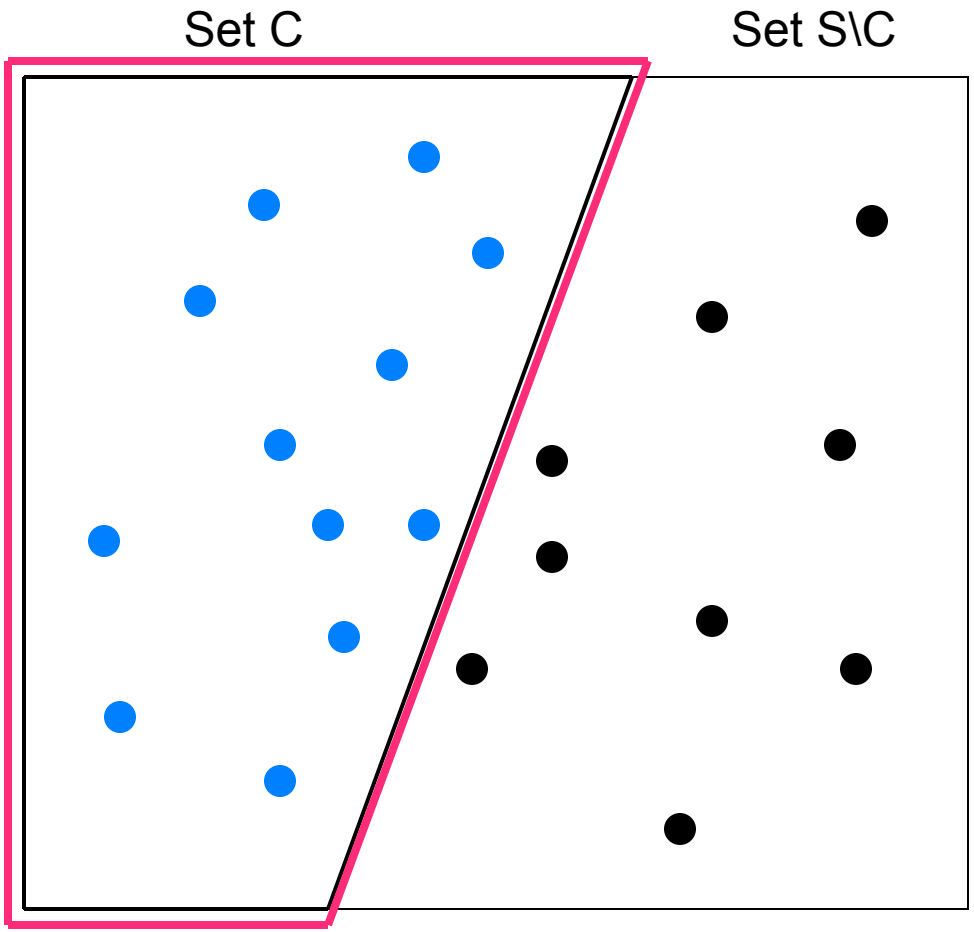
- We want to design our detector system (or medical test or neural network) to function such that a ‘detected’ signal means that which caused it to be detected also has the ‘characteristic’.
- We want detection to be strongly associated with a specified characteristic.

Set Definitions

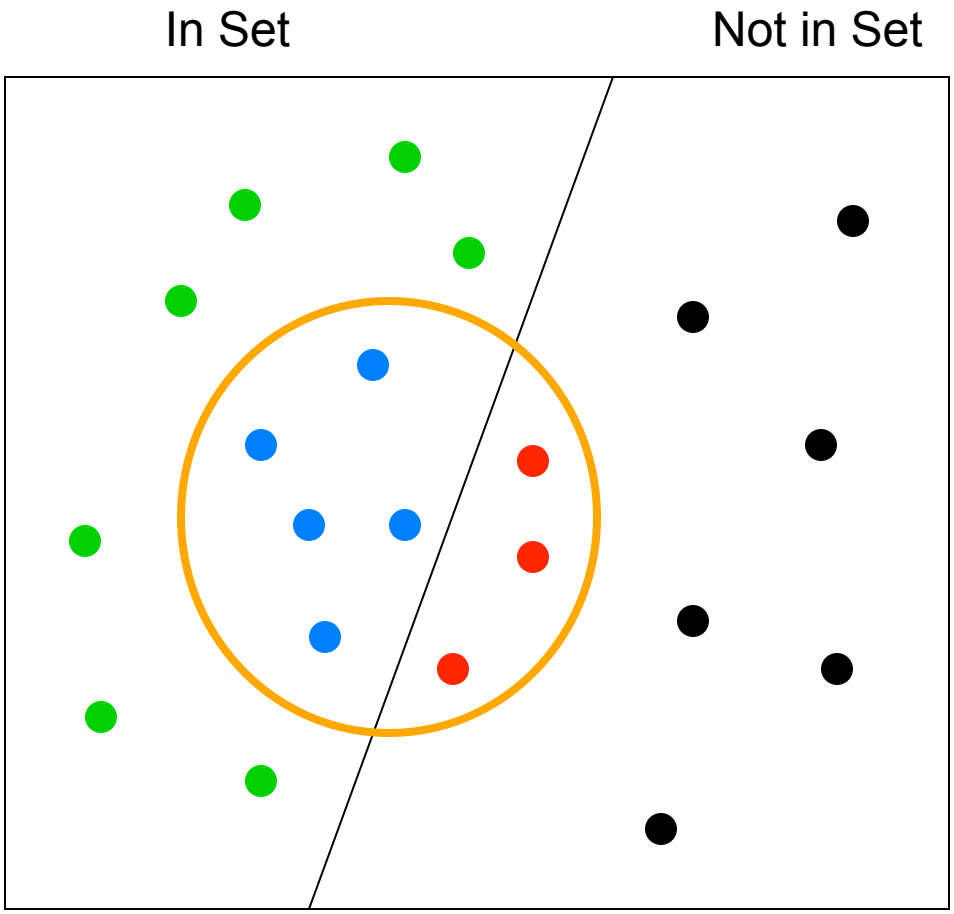
We want to detect signals from set C, i.e., want to detect whether an entity has the characteristic.

- Define the following sets.
  - $S_1 = \{x_i \mid x \in D \wedge x \in C\}$
  - $S_2 = \{x_i \mid x \in C\}$
  - $S_3 = \{x_i \mid x \notin D \wedge x \notin C\}$
  - $S_4 = \{x_i \mid x \notin C\}$

Ideally, we want this



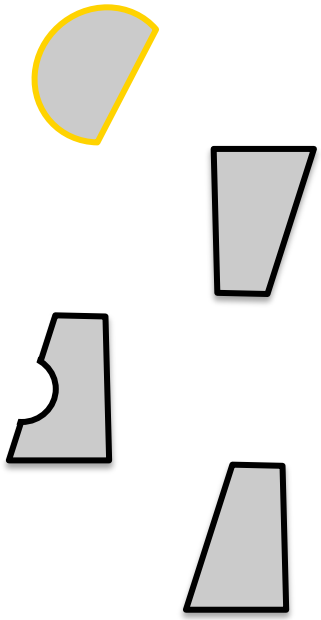
In Reality, we usually have this



Everything in the yellow circle is 'detected'; i.e., produces a positive test result

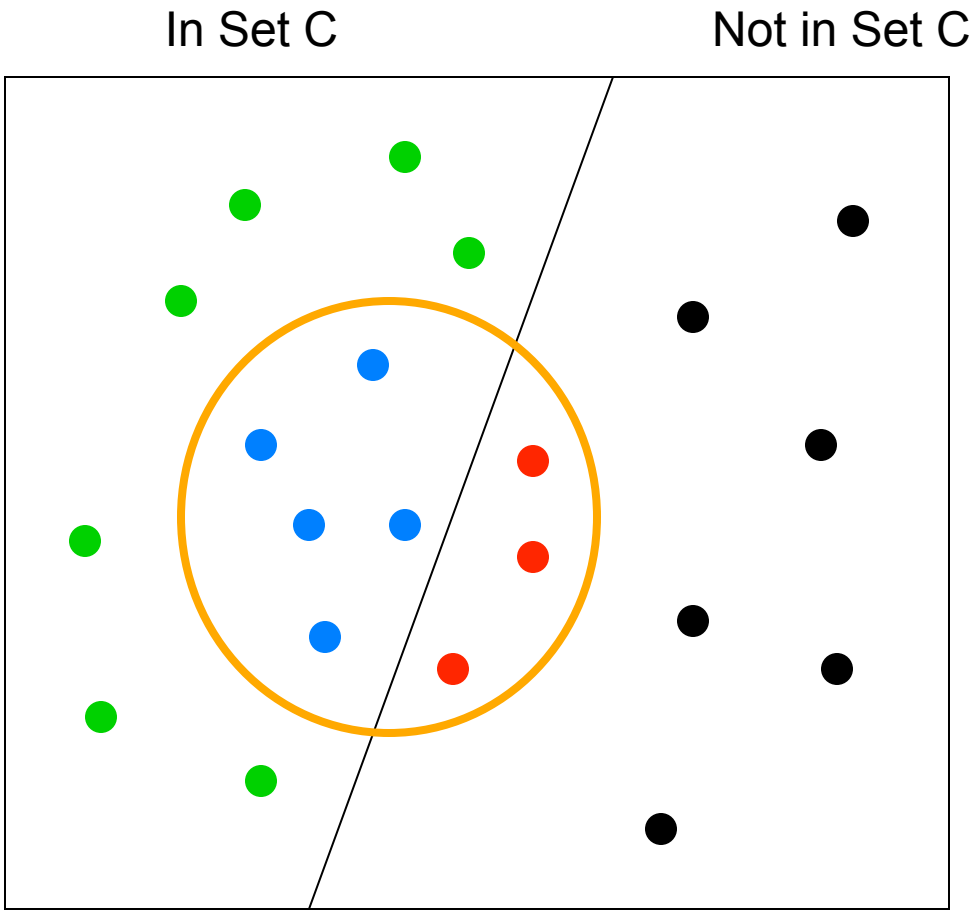
Receiver Operating Characteristics

- Define the following sets.
  - $S_{TP} = \{x_i \mid x \in D \wedge x \in C\}$
  - $S_{TP+FN} = \{x_i \mid x \in C\}$
  - $S_{TN} = \{x_i \mid x \notin D \wedge x \notin C\}$
  - $S_{TN+FP} = \{x_i \mid x \notin C\}$



Note, any element $x_i \notin D$ is an element which tests negative.
Any element $x_i \in D$ is an element which tests positive.

Receiver Operating Characteristics

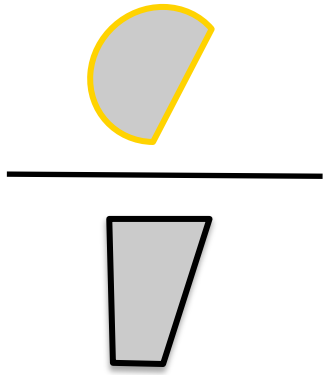


Blue is TP and Red is FP. Green is FN. Black is TN

Some Quantities we are interested in:

$$\Pr\{x \in D | x \in C\} = \frac{\Pr\{x \in D \cap C\}}{\Pr\{x \in C\}}$$

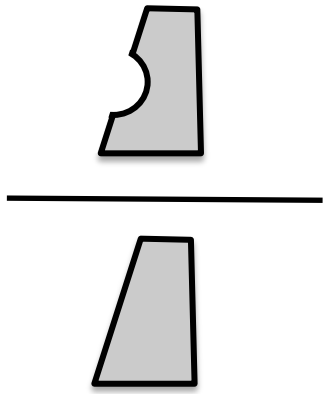
$$\Pr\{D|C\} = \frac{\Pr\{D \cap C\}}{\Pr\{C\}}$$



Sensitivity

Some Quantities we are interested in:

$$\Pr\left\{x \notin D \middle| x \notin C\right\} = \Pr\left\{\bar{D}\middle|\bar{C}\right\} = \frac{\Pr\left\{\bar{D} \cap \bar{C}\right\}}{\Pr\left\{\bar{C}\right\}}$$



Specificity

Receiver Operating Characteristics

Recall the following sets.

- $S_{TP} = \{x_i \mid x \in D \wedge x \in C\}$
- $S_{TP+FN} = \{x_i \mid x \in C\}$
- $S_{TN} = \{x_i \mid x \notin D \wedge x \notin C\}$
- $S_{TN+FP} = \{x_i \mid x \notin C\}$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

The Terms

Sensitivity: The higher this is, the greater the likelihood that if an object is in the characteristic set, it will result in a positive test result---*i.e.*, it will be detected hence the term.

The Terms

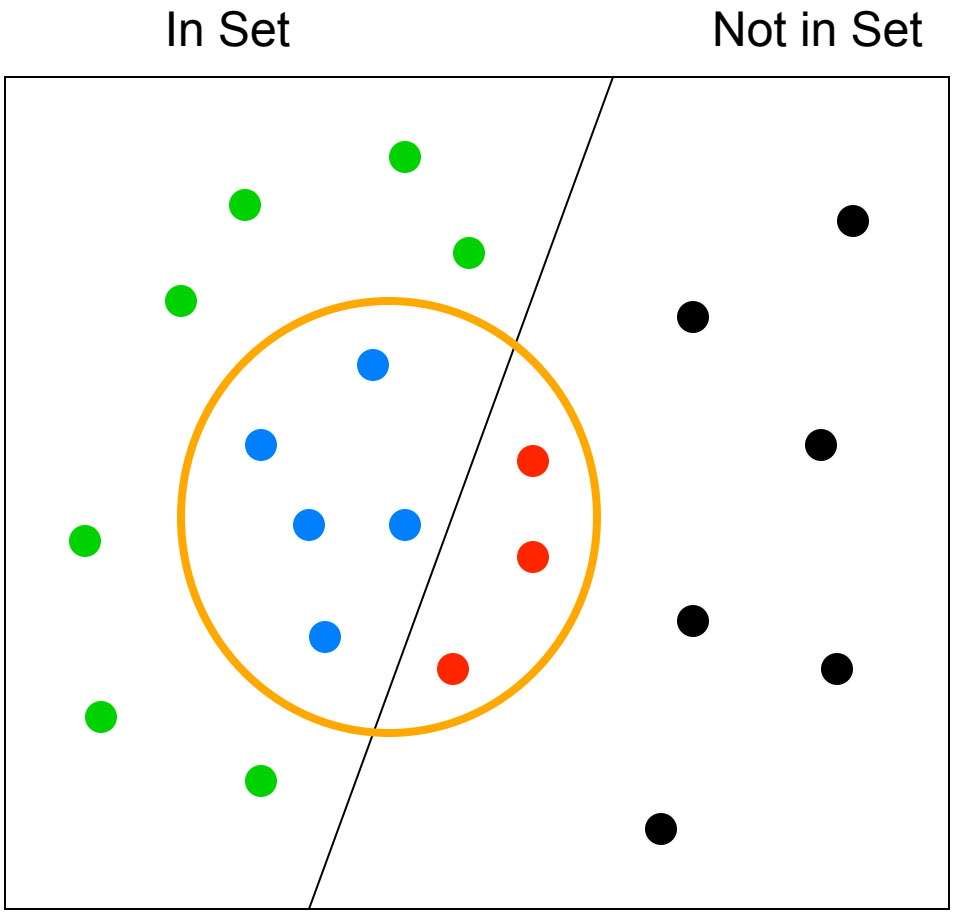
Specificity: A measure of the probability that if an object is *not* in the characteristic set then it will *not* be detected. Thus, the detection is specific to the object having the characteristic.

Matrix Form

	In Set	Not in Set
Test Positive	##	##
Test Negative	##	##

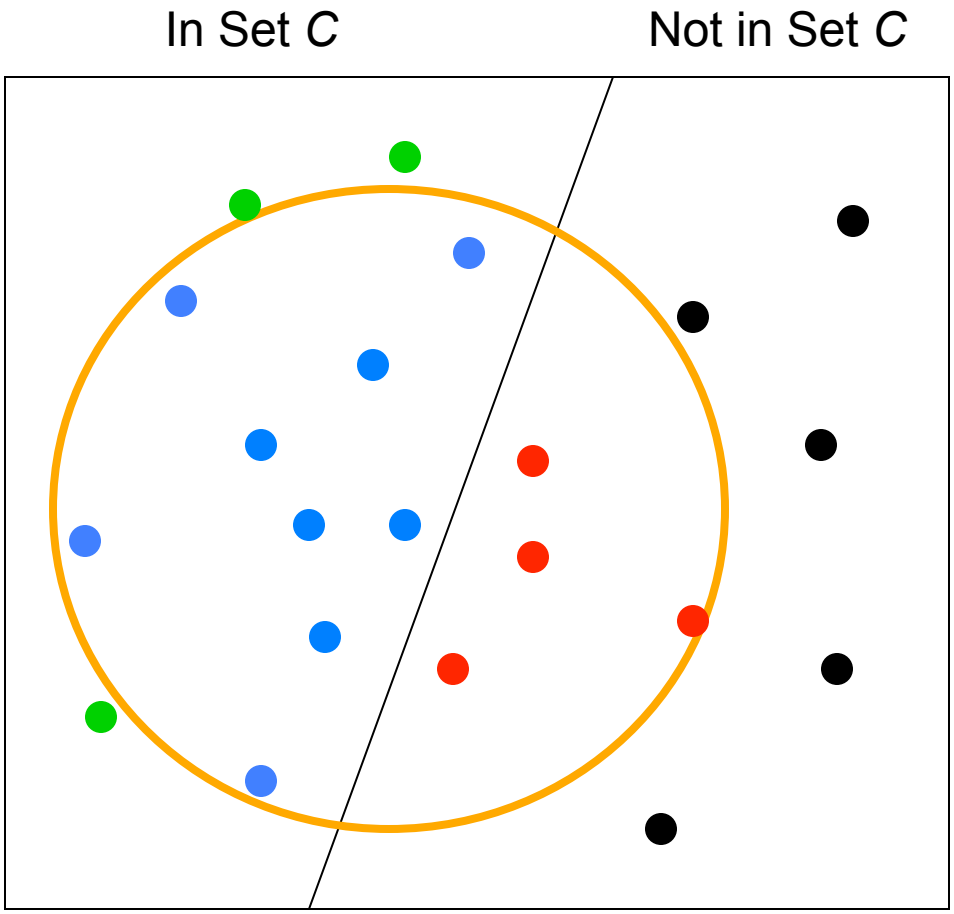
Confusion Matrix

Receiver Operating Characteristics



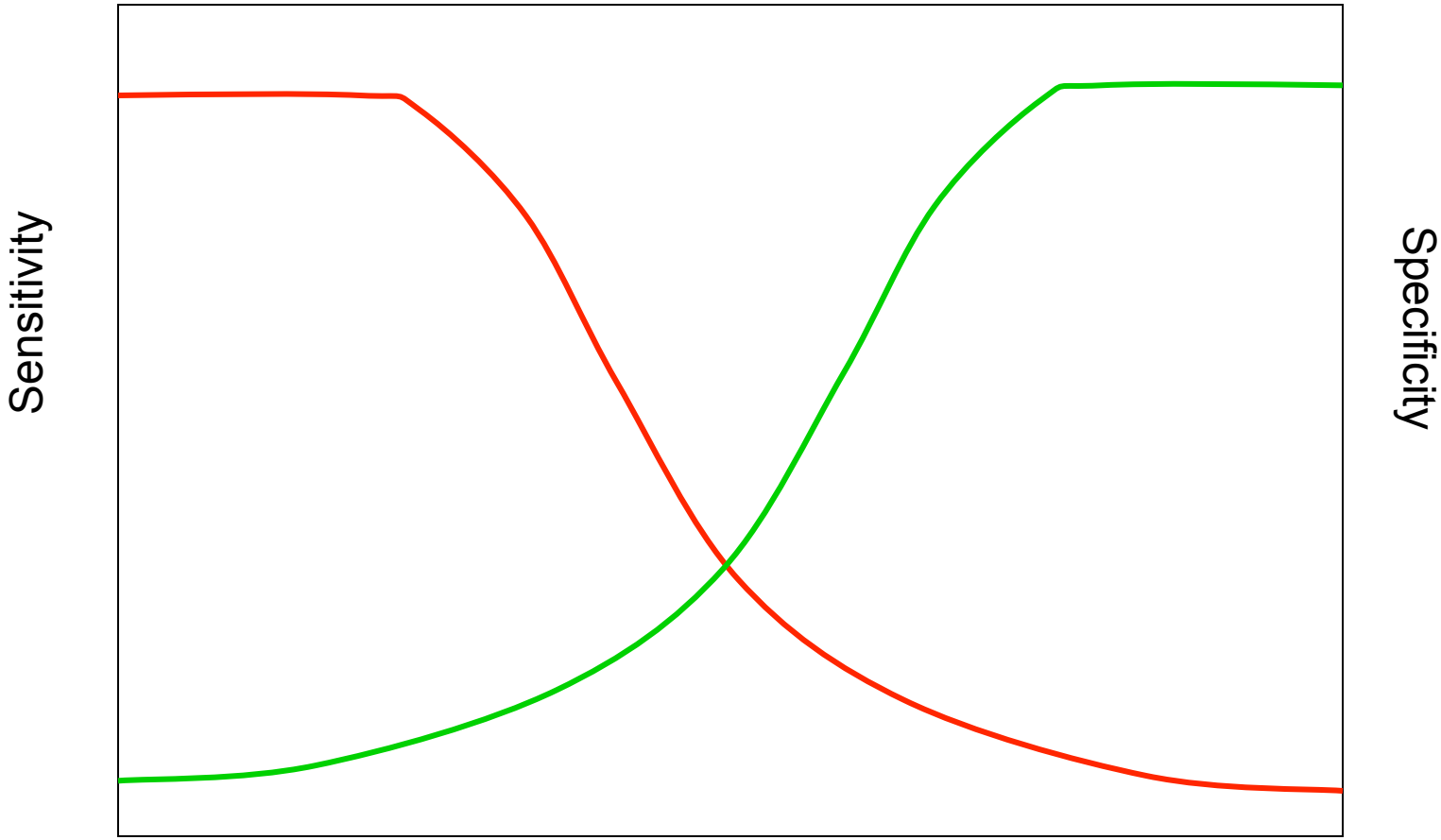
Blue is TP and Red is FP. Green is FN. Black is TN

Receiver Operating Characteristics



Blue is TP and Red is FP. Green is FN. Black is TN

Characteristic Curves



Other Useful Quantities

$$\Pr\{x \in C | x \in D\} = \Pr\{C|D\} = \frac{\Pr\{C \cap D\}}{\Pr\{D\}}$$

Positive Predictive Value---PPV

$$\Pr\{x \notin C | x \notin D\} = \Pr\{\bar{C}|\bar{D}\} = \frac{\Pr\{\bar{C} \cap \bar{D}\}}{\Pr\{\bar{D}\}}$$

Negative Predictive Value---NPV

Summary

- ROCs are useful ways of assessing performance of neural networks especially in the context of the accuracy of classification-type applications.
- Can be used with classic MSEs to assess neural network training and performance.