EECS 391: Introduction to AI

Soumya Ray

http://vorlon.case.edu/~sray/eecs391_sp12/index.html

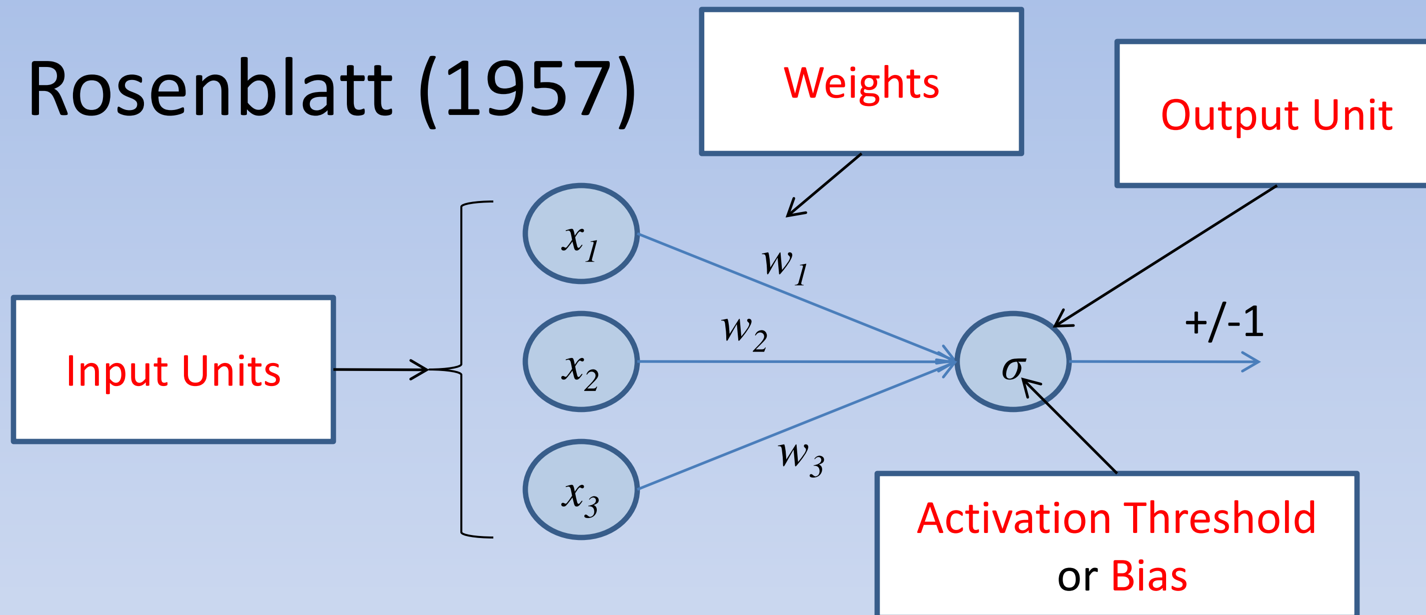
Email: sray@case.edu

Office: Olin 516

Office hours: Tues 2:30-4:00 or by appointment

Perceptron/Linear Threshold Unit

- Rosenblatt (1957)

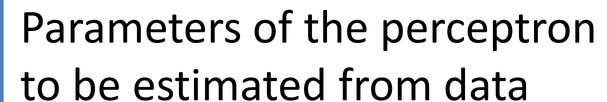


$$h(\mathbf{x};\mathbf{w},\sigma) = \begin{cases} +1 \text{ if } \mathbf{w} \bullet \mathbf{x} \geq \sigma \\ -1 \text{ else} \end{cases} = sign(\mathbf{w} \bullet \mathbf{x} - \sigma)$$

Activation Function
sign(x)=+1 if x ≥ 0, −1 else

Parameters of the Perceptron

$$h(\mathbf{x};\mathbf{w}) = \begin{cases} +1 \text{ if } \mathbf{w}\bullet\mathbf{x} \geq \sigma \\ -1 \text{ else} \end{cases} = sign(\mathbf{w}\bullet\mathbf{x} - \sigma)$$



Evaluation Phase

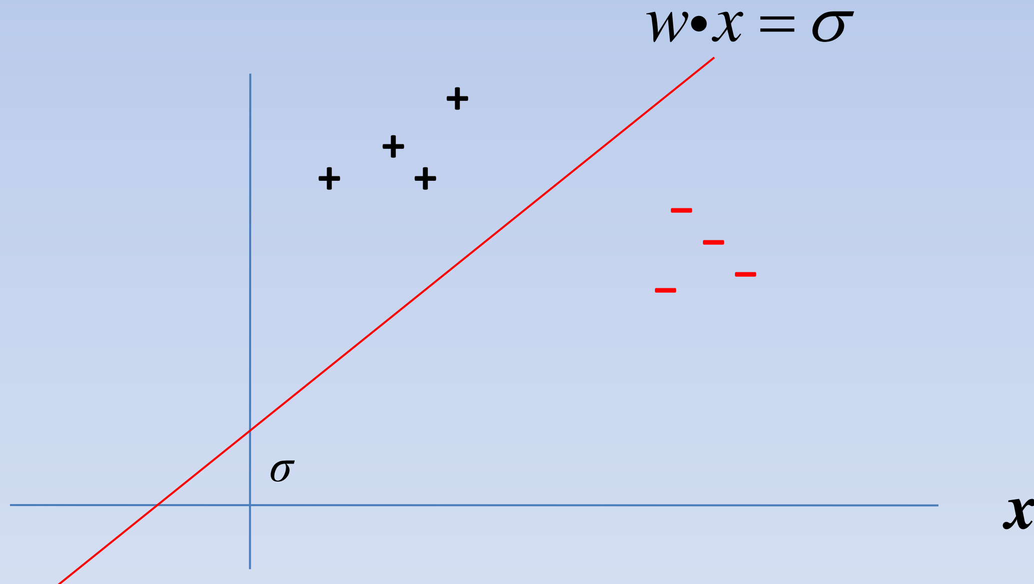
- Given (w, σ), classify an example $\mathbf{x}$

- $w=(1,2)\ \sigma=0.5$

x_1	x_2	h
0	0	-1
0	1	1

Geometry of the Perceptron

- A perceptron's separating surface defines a hyperplane in feature space



"Decision Boundary" or
"Separating Surface"

Training Phase

- Given Data:

x_1	x_2	y
0	0	-1
0	1	1

- Find (w, σ) so that the resulting perceptron matches y

Loss Functions

- We will define a function called a "loss function"

$$L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}, \sigma)$$

- This function will measure the difference between our *current estimates* of y ($\hat{\mathbf{y}}$) and the *true* y (which is known), over all training examples, with respect to (w, σ)
- Then our goal will be to *minimize* the loss function *with respect to* (w, σ)

Training Phase

- Given a training sample and their class labels

$$D = \begin{pmatrix} x_{11} & \cdots & x_{1n} & -1 & y_1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} & -1 & y_m \end{pmatrix}$$

Add "dummy" column for σ, allows σ to be treated same as w

- Find parameters

$$\mathbf{w} = (w; \sigma)$$

- To **minimize** "loss" $L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w})$

A function that measures the difference between the actual labels $\mathbf{y}$ and the predicted labels $\hat{\mathbf{y}}$ in terms of $\mathbf{w}$

Loss function

- Define loss function to be "squared loss"

$$L(\mathbf{y},\hat{\mathbf{y}};\mathbf{w})=\frac{1}{2}\sum_{i=1}^{m}(y_i-\hat{y}_i)^2=\frac{1}{2}\sum_{i=1}^{m}(y_i-sign(\mathbf{w}\bullet\mathbf{x}_i))^2$$

- Notice that $sign(1)=1$ and $sign(-1)=-1$, so if we can get $\mathbf{w}\cdot\mathbf{x}$ to equal $\mathbf{y}$ we can drop the *sign* function
  - This is much easier to deal with

$$L(\mathbf{y},\hat{\mathbf{y}};\mathbf{w})=\frac{1}{2}\sum_{i=1}^{m}(y_i-\mathbf{w}\bullet\mathbf{x}_i)^2$$

Iterative parameter update

$$L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)^2$$

- Calculate gradient with respect to parameters $\mathbf{w}$:

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)\frac{\partial(y_i - \sum_j w_j x_{ij})}{\partial w_j} = \sum_{i=1}^{m}(y_i - \mathbf{w}\bullet\mathbf{x}_i)(-x_{ij})$$

- Parameter Update:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\frac{dL}{d\mathbf{w}}$$

Learning rate
or
step size

Implementation

- Initialize parameters to small values in (-1,1)
- Loop through all examples, calculating gradient and adding up $\frac{\partial L}{\partial w_j} = \sum_{i=1}^{m} (y_i - \mathbf{w} \bullet \mathbf{x}_i)(-x_{ij})$
- Perform weight update $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{dL}{d\mathbf{w}}$
- Until the max gradient is close to zero (say, 1e-4)

Convergence

- This is an application of gradient descent
- Notice that this loss function is everywhere differentiable and bounded below
  - A well defined (unique!) minimum exists for any $\mathbf{D}$
  - This iterative procedure will find it
- But loss will not go to zero unless problem is separable by linear decision boundary (“linear separability”)

Training Phase

- Given Data:

x_1	x_2	y	$y\text{-}hat$
0	0	-1	1
0	1	1	1

- Start with $(\mathbf{w}=0,\ \sigma=-1)$

Perceptrons can do logic!

- Conjunctions

$$x_1 \wedge x_2 \wedge x_3 \Leftrightarrow y$$

$$1 \bullet x_1 + 1 \bullet x_2 + 1 \bullet x_3 \geq 2.5$$

- At least m-of-n

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3) \Leftrightarrow y$$

$$1 \bullet x_1 + 1 \bullet x_2 + 1 \bullet x_3 \geq 1.5$$

Things that can't be learned

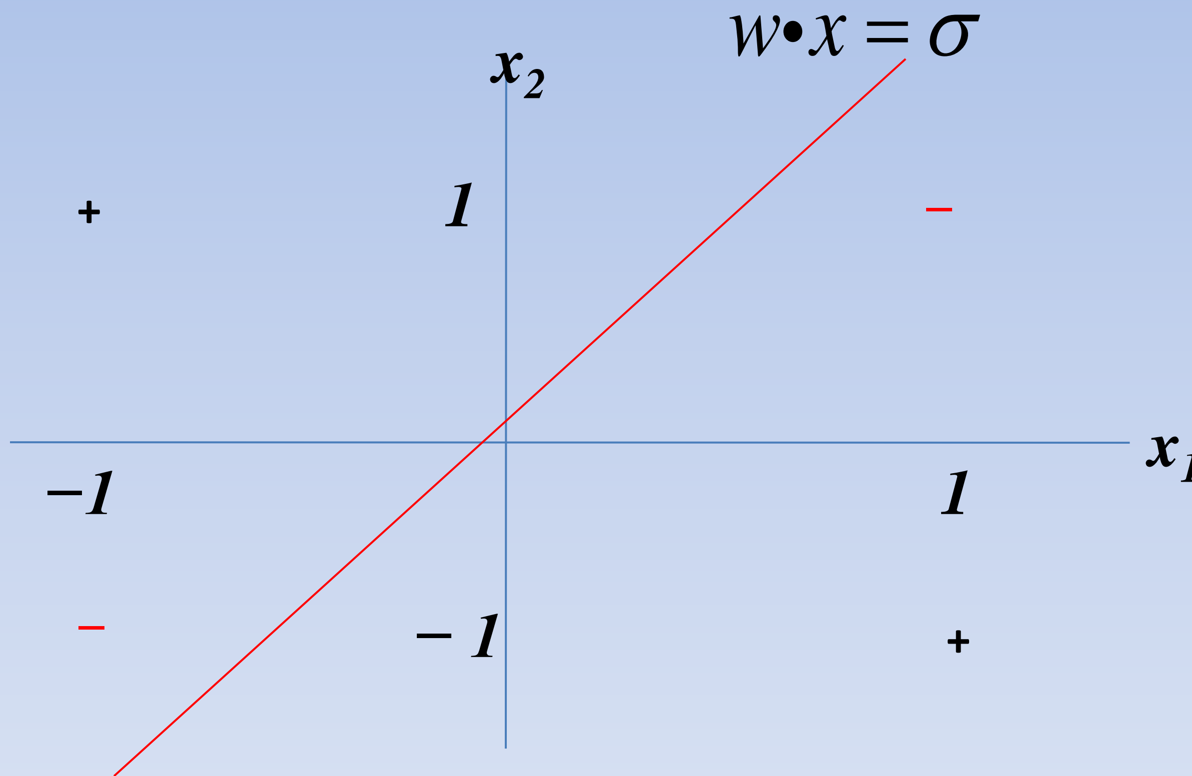
- Complex disjunctions

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \Leftrightarrow y$$

- Exclusive-OR

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \Leftrightarrow y$$

XOR and the Perceptron

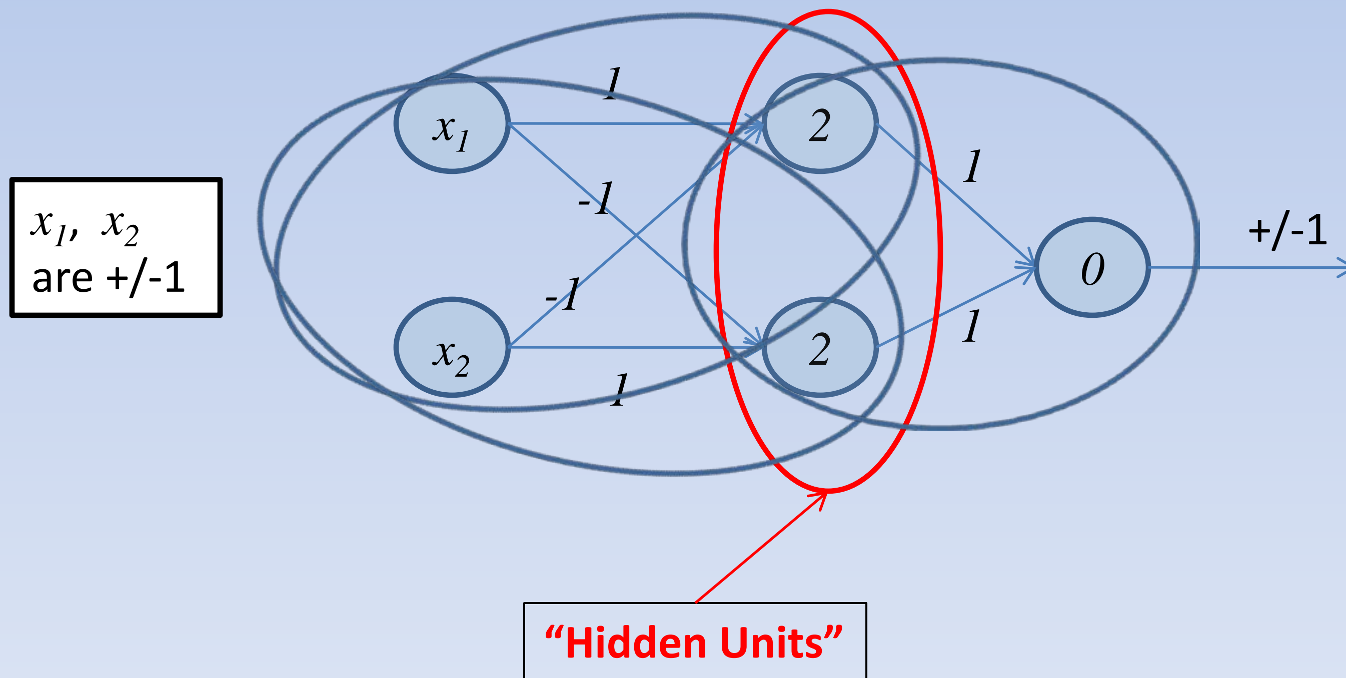


x_1	x_2	f
F	F	F
T	F	T
F	T	T
T	T	F

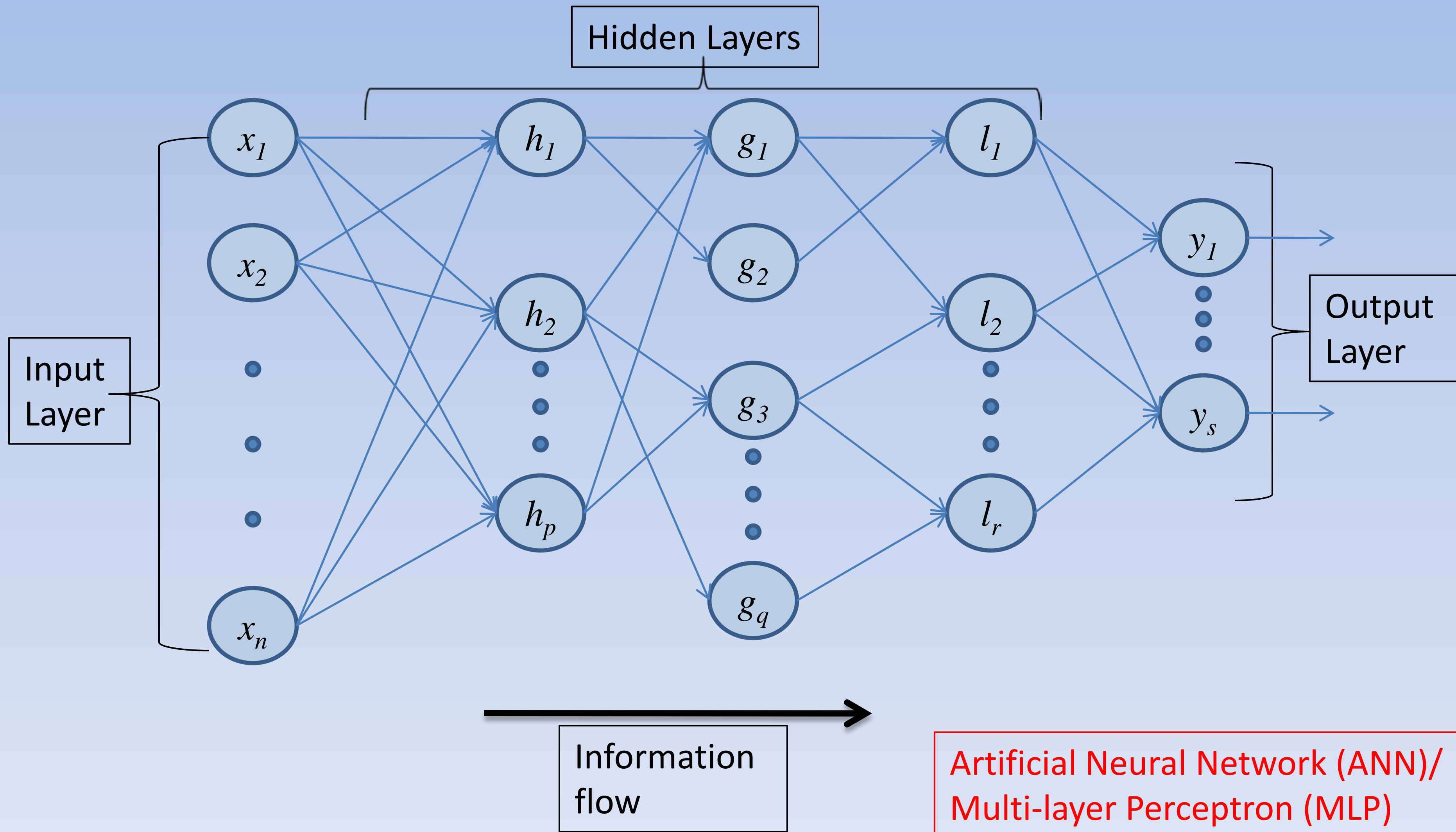
How to fix this?

Idea

- So far, single "neuron"
- Let's connect them into a network



"Feedforward" Network Topology



Representation Ability

- Every Boolean function can be represented by a network with one hidden layer of units (Minsky and Papert)
- Every bounded continuous function can be represented by a network with one hidden layer (Cybenko et al)
- Every function on R^n can be represented by a network with two hidden layers! (Cybenko et al)
  - (with arbitrary numbers of hidden units)

Tradeoffs

- Very large number of degrees of freedom
  - Network topology
    - Number of layers, number of hidden units in each layer, edge configuration, even different activation functions
  - Network parameter values
- Power comes at a price
  - Very very easy to "fit the noise"
  - Very long time and large samples required to train

Evaluation Phase

- Given an example, propagate it forward through the network, layer by layer, till output is reached

Training Phase

- As before, given a training sample and their class labels

$$D = \begin{pmatrix} x_{11} & \cdots & x_{1n} & 1 & y_1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} & 1 & y_m \end{pmatrix}$$

- Find parameters $\mathbf{w}$

- To minimize "loss" $L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w})$

Hidden Layer Activation Function

- For a perceptron, we used a $sign()$ function as an activation function

- This function is problematic for optimization because it isn't differentiable
  - For perceptron this was OK, we were able to relax the formulation and handle this
  - But for a general ANN we need a smooth activation function

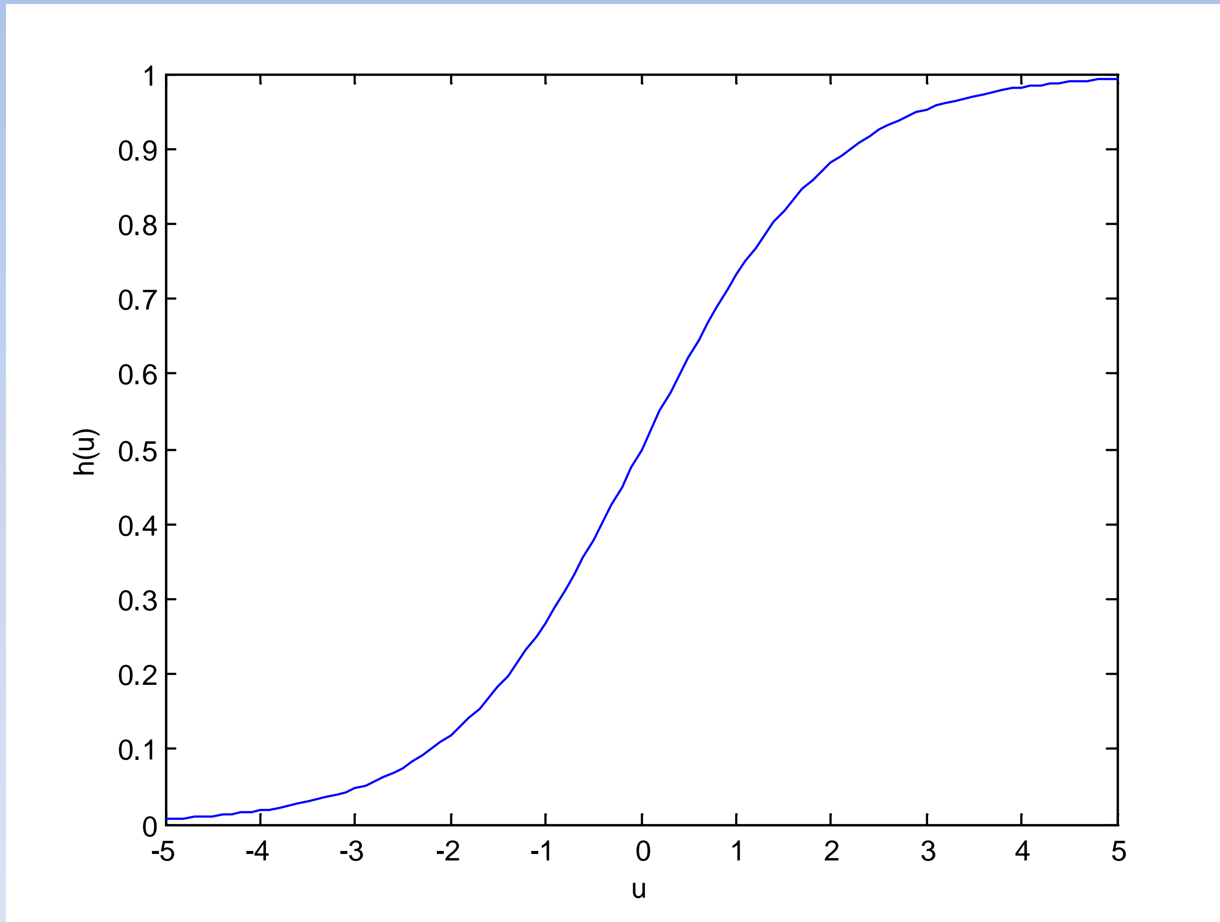
Sigmoid Activation Function

$$h(\mathbf{x};\mathbf{w},\sigma) = sign(\mathbf{w}\bullet\mathbf{x}-\sigma)$$

$$h(\mathbf{x};\mathbf{w},\sigma) = (1+e^{-\mathbf{w}\bullet\mathbf{x}})^{-1}$$

Activation Function: Sigmoid

$$h(u) = (1 + e^{-u})^{-1}$$



Backpropagation

x_1 x_2 x_n

h_1 h_2 h_p

g_1 g_2 g_3 g_q

l_1 l_2 l_r

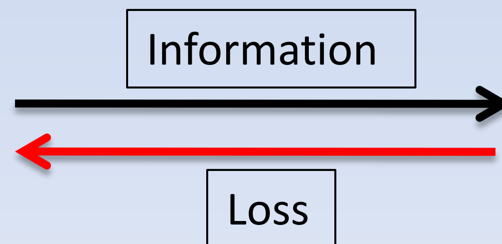
y_1 y_s

$\hat{y}_1$

$L(y_1, \hat{y}_1)$

$w_{g_3 l_2}$

$\frac{\partial L}{\partial w_{g_3 l_2}}$



Backpropagation

- Feed the examples forward through the network, observe the output and calculate the loss
- Perform "layer-wise" gradient descent on the loss function with respect to each weight, starting with output layer
  - For each weight in each layer, calculate its contribution to the overall loss using the derivative
  - Update the weight in the negative gradient direction

Evaluation Methodology

Issue

- We use learning algorithm A to produce a classifier for a binary classification task.
  - How do we know if the classifier is any good?

Goal

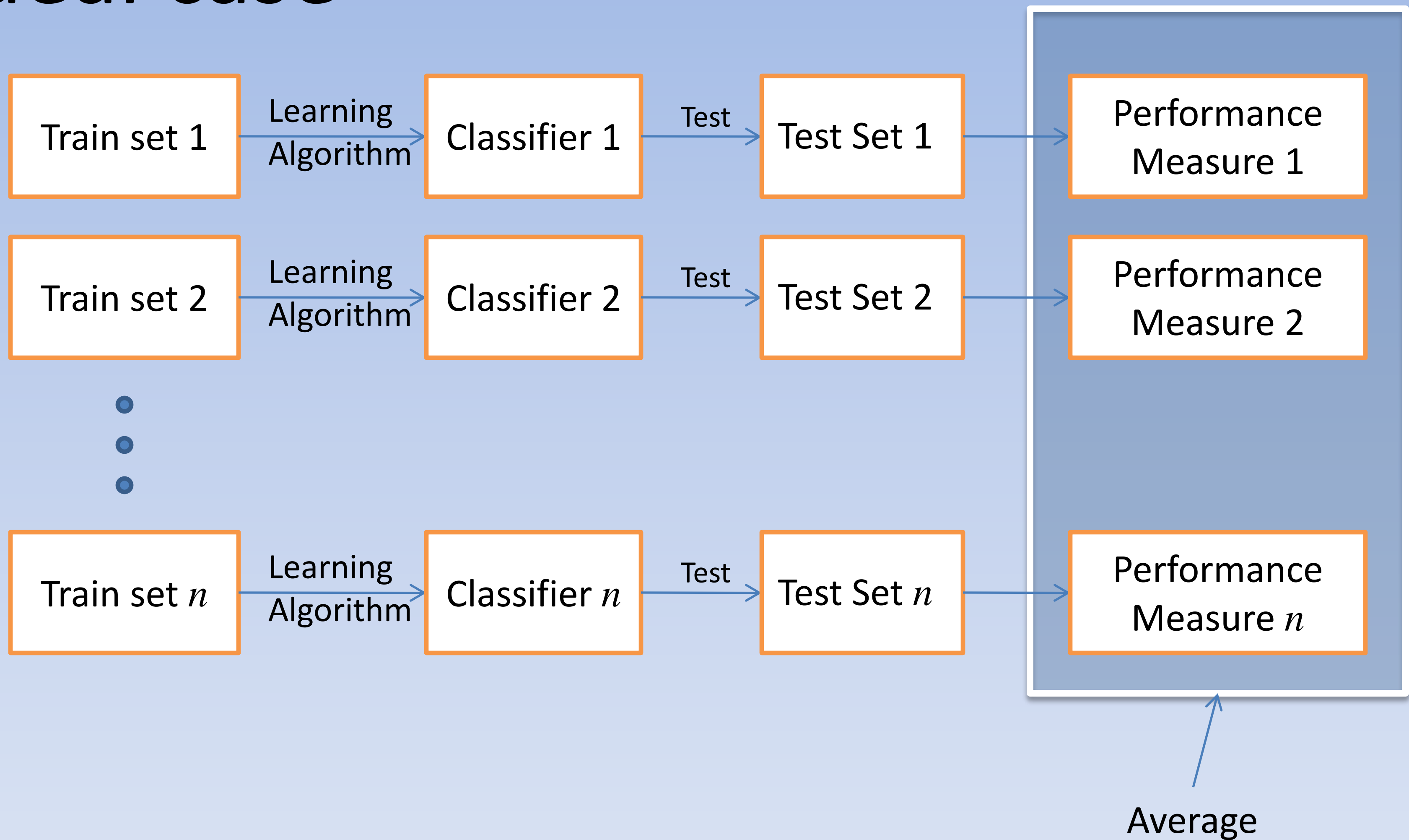
- Want a reliable measure of expected future performance of a learning algorithm on some learning problem

- How to measure future performance?

Idea

- Use separate sets of examples for training and evaluation
- The examples for evaluation will be new to the learned classifier
  – Proxy for “future examples”
- Ideally, do this lots of times to get reliable estimates

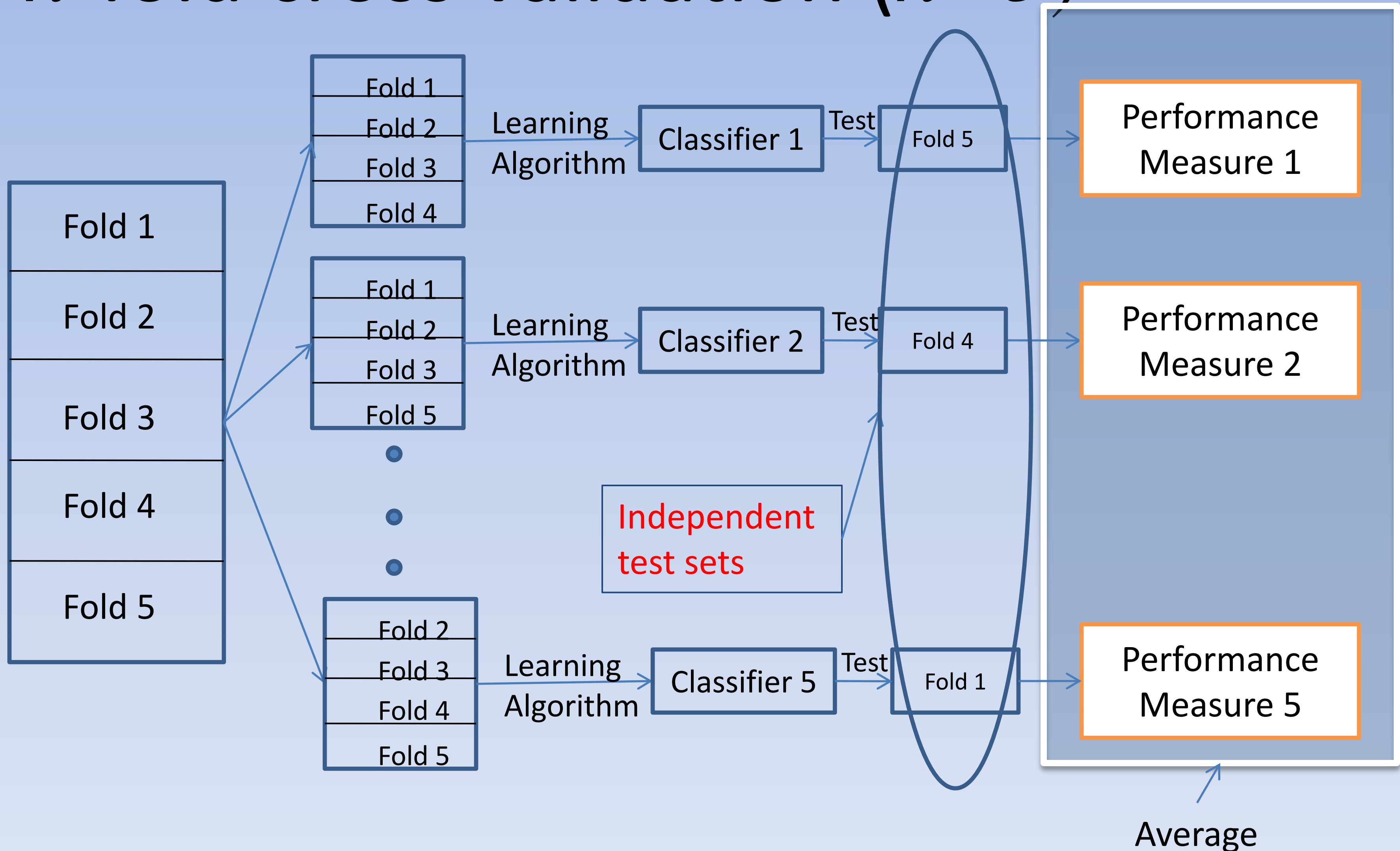
Ideal case



n-fold cross validation

- Generally, examples are limited
- For proper learning, need training sets to be as large as possible
- For good estimates of future performance, need a number of independent test sets
- Idea: partition the available examples into "folds"

n-fold cross validation ($n=5$)



Metrics for Classification

Contingency Table

Class according to Learned Classifier (Predicted Answer)	Class according to Target Concept (Correct Answer): Positive	Class according to Target Concept (Correct Answer): Negative
Positive	True Positives (TP)	False Positives (FP) (Type I error)
Negative	False Negatives (FN) (Type II error)	True Negatives (TN)

Example

Does Learned Classifier predict patient has disease?	Does patient have disease? Positive (Disease)	Negative
Positive (Disease)	Patient has disease Classifier predicts patient has disease	Patient does **not** have disease Classifier predicts patient has disease
Negative	Patient has disease Classifier predicts patient **does not** have disease	Patient does not have disease Classifier predicts patient does not have disease

Accuracy

- Most commonly used measure for comparing classification algorithms

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Weaknesses of Accuracy

- Does not account for:
  - Skewed class distributions
  - Differential misclassification costs
  - Confidence estimates from learning algorithms

Weighted Accuracy

- Corrects for skewed class distributions

$$WAcc = \frac{1}{2}\left(\frac{TP}{Allpos} + \frac{TN}{Allneg}\right)$$

$$= \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

Summary

- Perceptrons and ANNs
- Overfitting
- Restriction and preference biases
- Evaluation of algorithms
  - Contingency Table
  - Accuracy, weighted accuracy