Data Analysis and Machine Learning using Python

Lecture 6: Boosted Decision Trees and Multi-Layer Perceptrons;
intro to scikit-learn
April 20 2024

Today:

- Quick review of mid-term quiz
- Reminder: Basic ideas of supervised machine learning
- Boosted Decision Trees
- Perceptrons and Neural Networks
- How to use scikit-learn

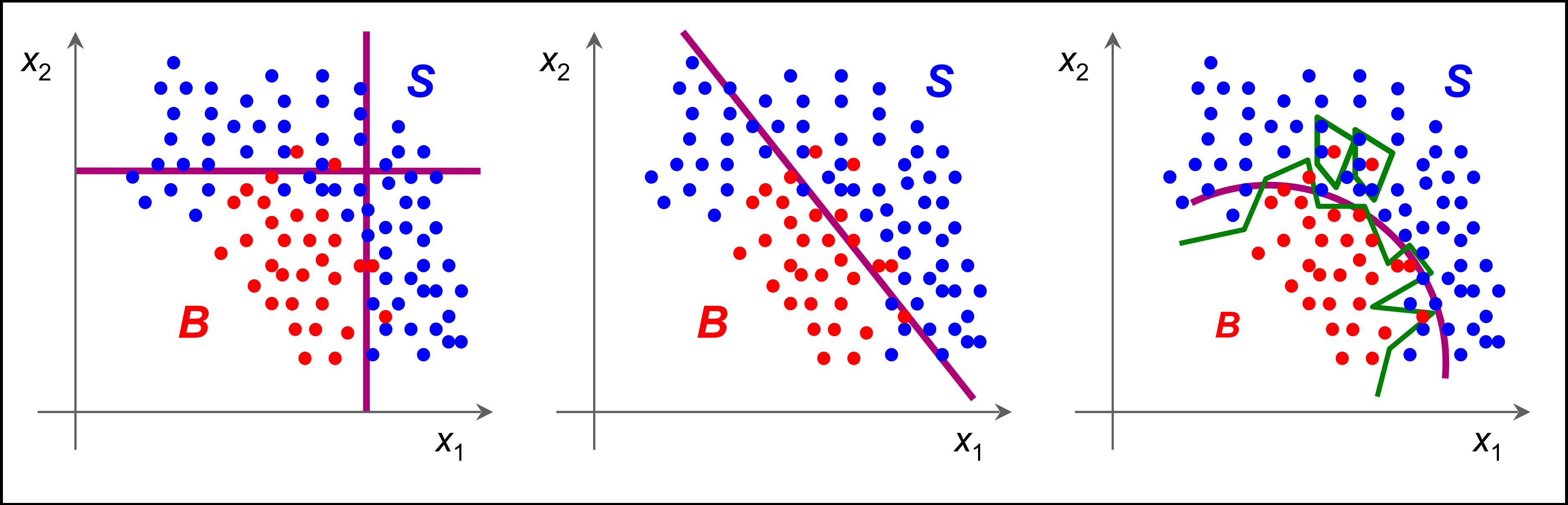
Multivariate Analysis: Classification and Regression

- Related, common problems in data analysis
- **Multivariate analysis:** Multiple features (input variables)
- **Classification**: Which *class* does a particular object belong to, based on features?
- **Regression**: Predict output for a set of features (e.g., interpolation/extrapolation)

Supervised machine learning

- **Machine learning (ML):** Build ***model*** for classification or regression automatically
- **Supervised ML**: Model “learns” from training sample for which correct answer is known
- **Model**: “function” determined by ML algorithm and used for classification or regression
  - Model provides output value for any valid input
  - Model has to generalize from the training sample to test sample

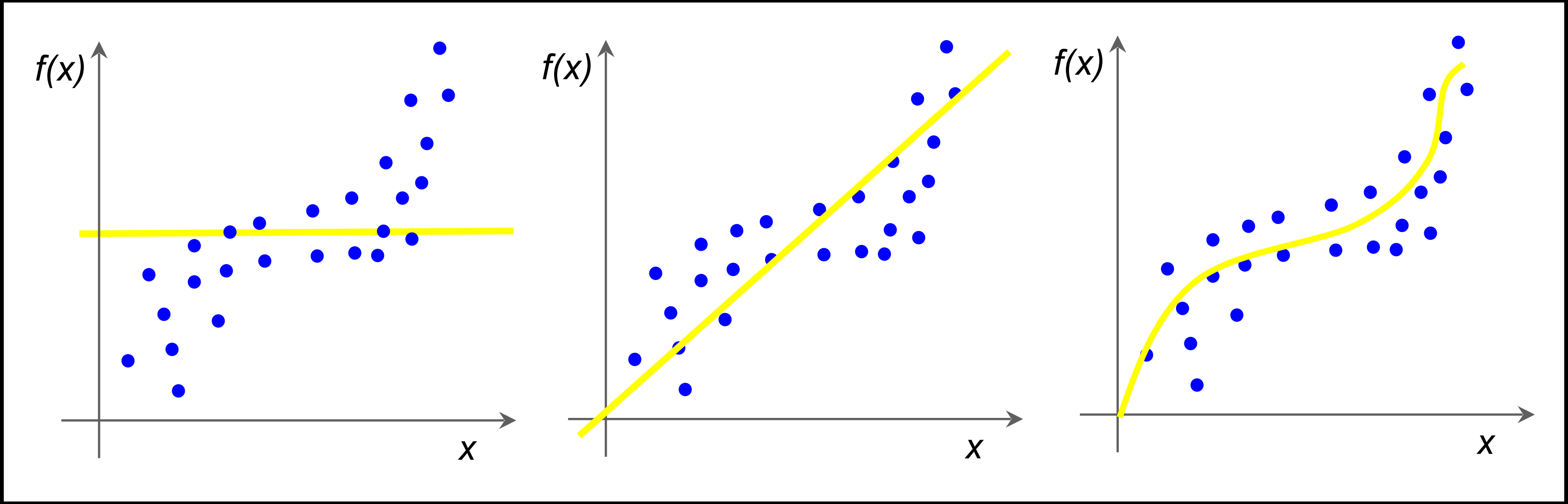
Classification



- Classification: Find “decision boundary” to separate different event classes (in physics, often “signal” and “background”)

- Boundary: D-1 dimensional hypersurface in D-dimensional feature space

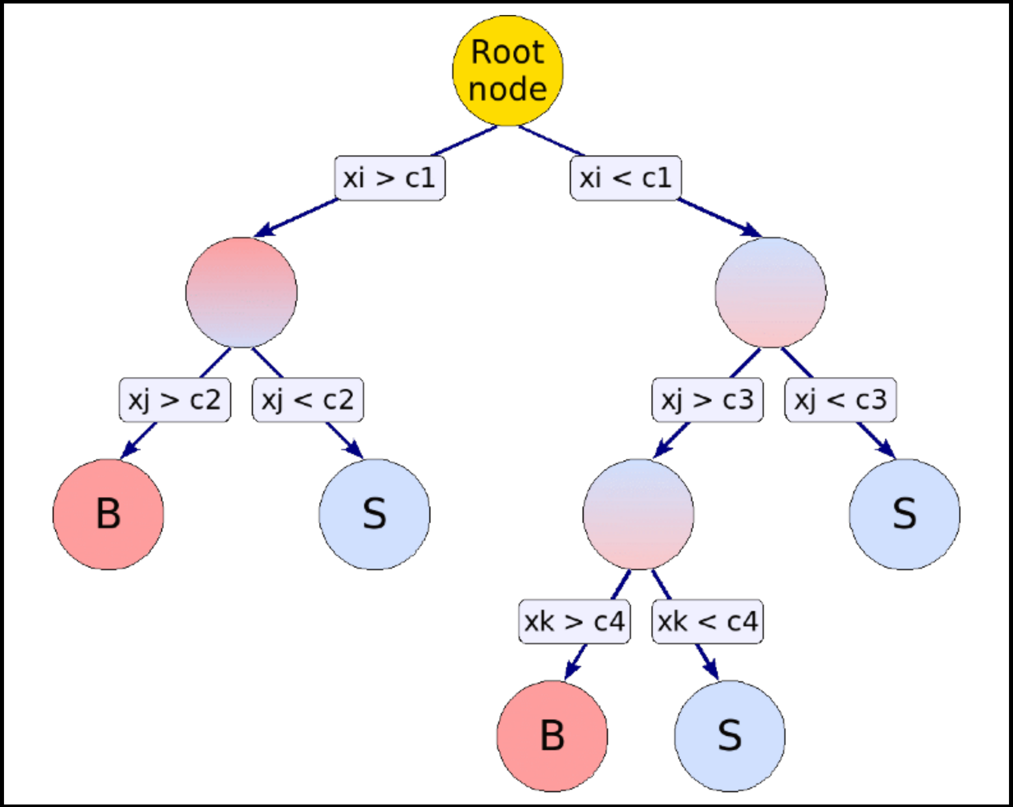
Regression



- Regression: Find “model” to predict f(**x**) for every point x in feature space
    - in general, x is D-dimensional vector
- Model: D-dimensional “hypersurface” in (D+1)-dimensional space (**x**,f(**x**))

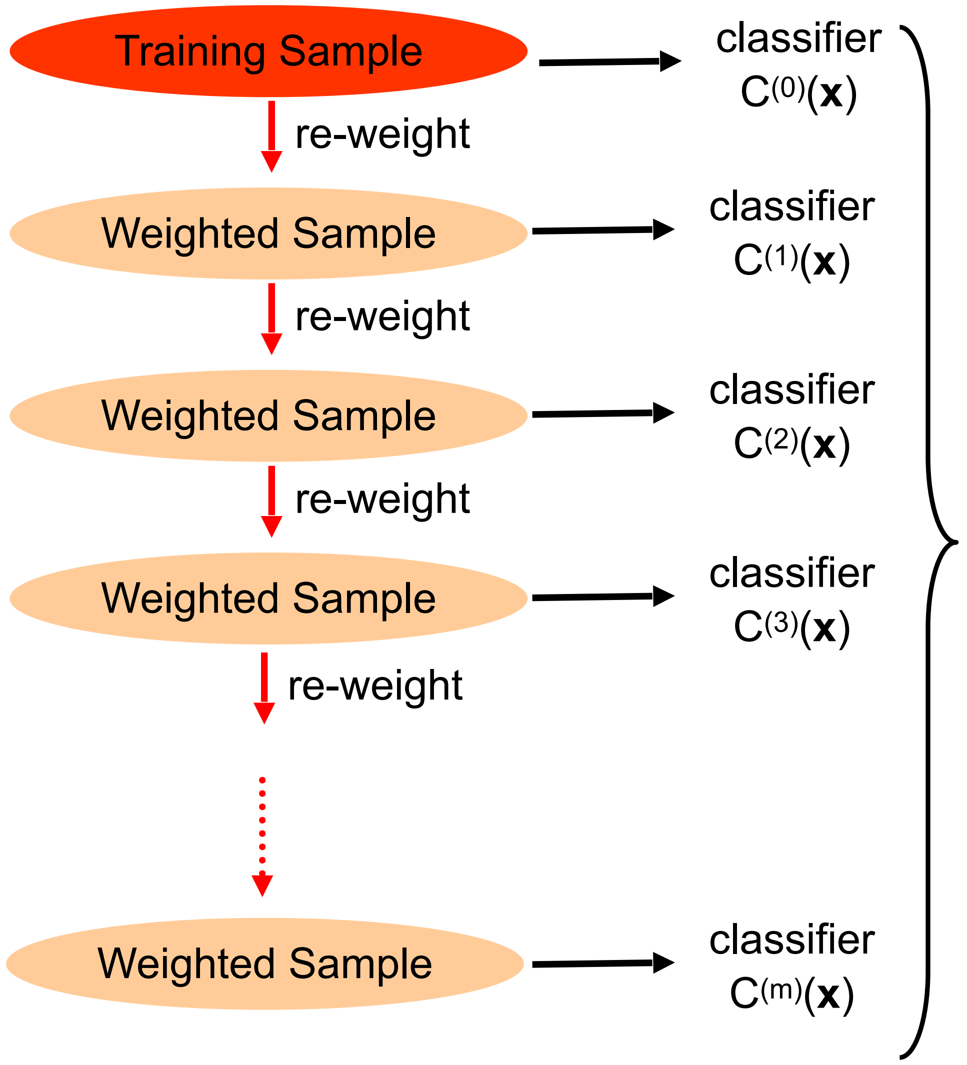
Boosted Decision Trees

Decision Trees



- Sequential application of cuts
- Each cut on different variable
- Final cut in each branch splits data into signal and background

Boosting

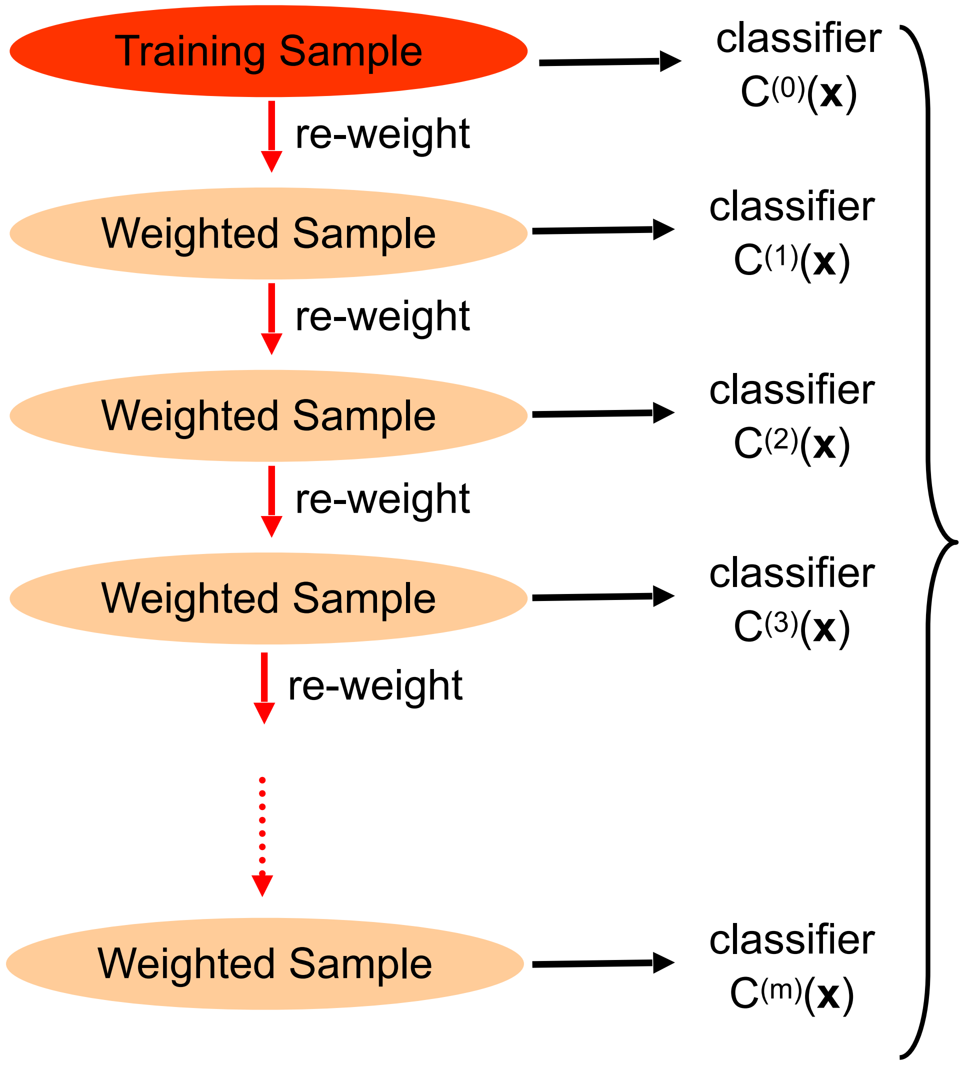


Training Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
classifier $C^{(0)}(\mathbf{x})$
classifier $C^{(1)}(\mathbf{x})$
classifier $C^{(2)}(\mathbf{x})$
classifier $C^{(3)}(\mathbf{x})$
classifier $C^{(m)}(\mathbf{x})$

- <u>Boosted Decision Trees:</u>
- combine many decision trees, derived from the same sample
  - overcomes overfitting problem
  - increases performance

$$y(x) = \sum_{i}^{N_{\text{Classifier}}} w_i C^{(i)}(x)$$

Adaptive Boosting (AdaBoost)


Training Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
classifier $C^{(0)}(\mathbf{x})$
classifier $C^{(1)}(\mathbf{x})$
classifier $C^{(2)}(\mathbf{x})$
classifier $C^{(3)}(\mathbf{x})$
classifier $C^{(m)}(\mathbf{x})$

- AdaBoost re-weights events misclassified by previous classifier by:

$$\frac{1-f_{err}}{f_{err}} \quad \text{with:}$$

$$f_{err} = \frac{\text{misclassified events}}{\text{all events}}$$

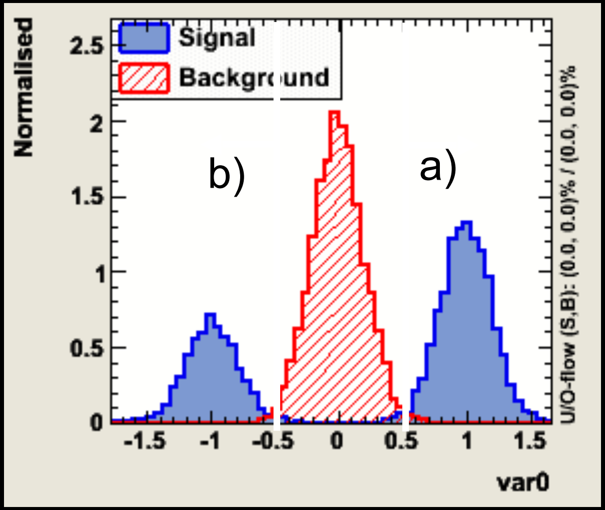
- AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

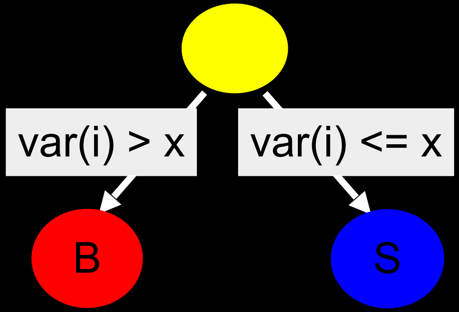
$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

AdaBoost: A simple demonstration

Example:

- Data file with three "bumps"
- Weak classifier (i.e. one single simple "cut" ↔ decision tree stumps)





a) Var0 > 0.5 → ε_{sig}=66% $\varepsilon_{bkg} \approx$ 0% misclassified events in total 16.5%

or

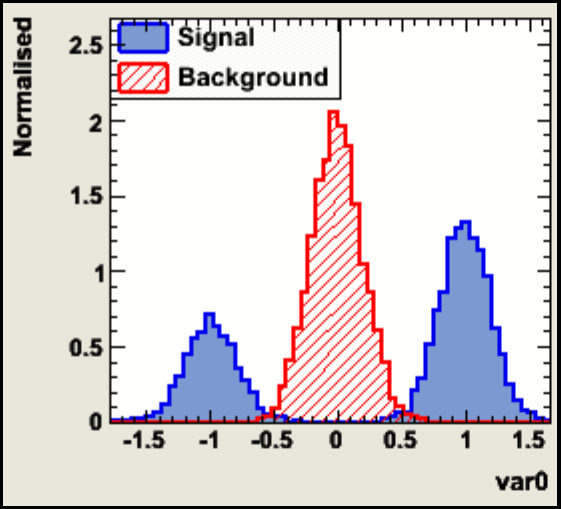
b) Var0 < -0.5 → ε_{sig}=33% $\varepsilon_{bkg} \approx$ 0% misclassified events in total 33%

the training of a single decision tree stump will find "cut a)"

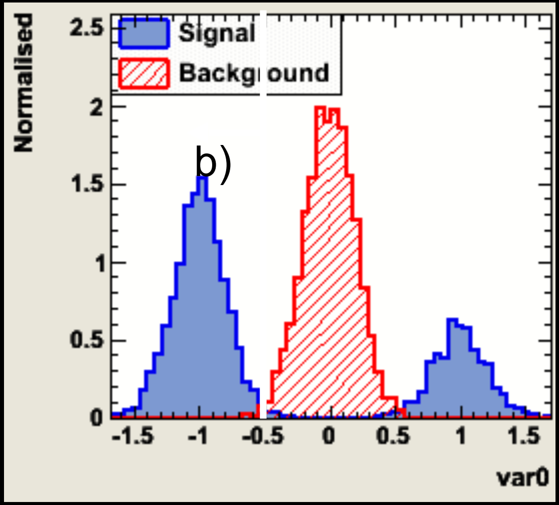
AdaBoost: A simple demonstration

➔ before building the next “tree”: weight wrong classified training events by (1-err/err)) ≈ 5

➔ the next “tree” sees essentially the following data sample:

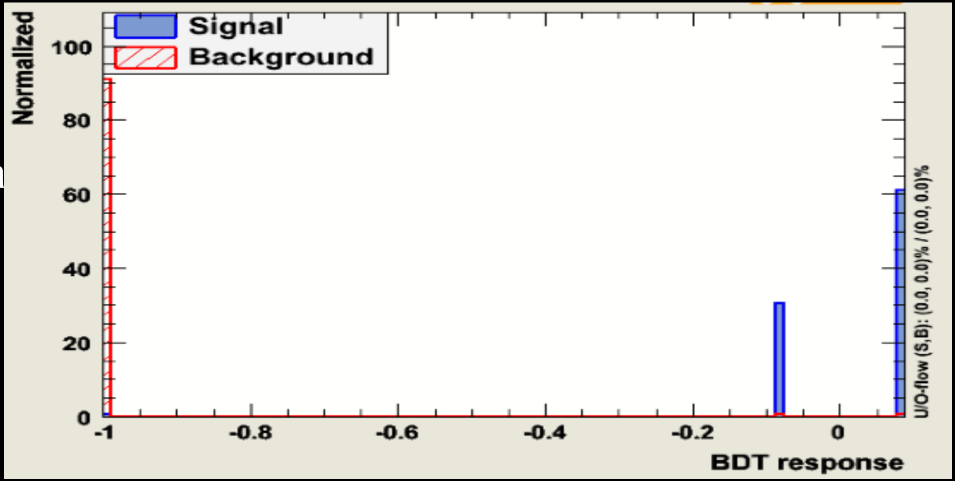


re-weight

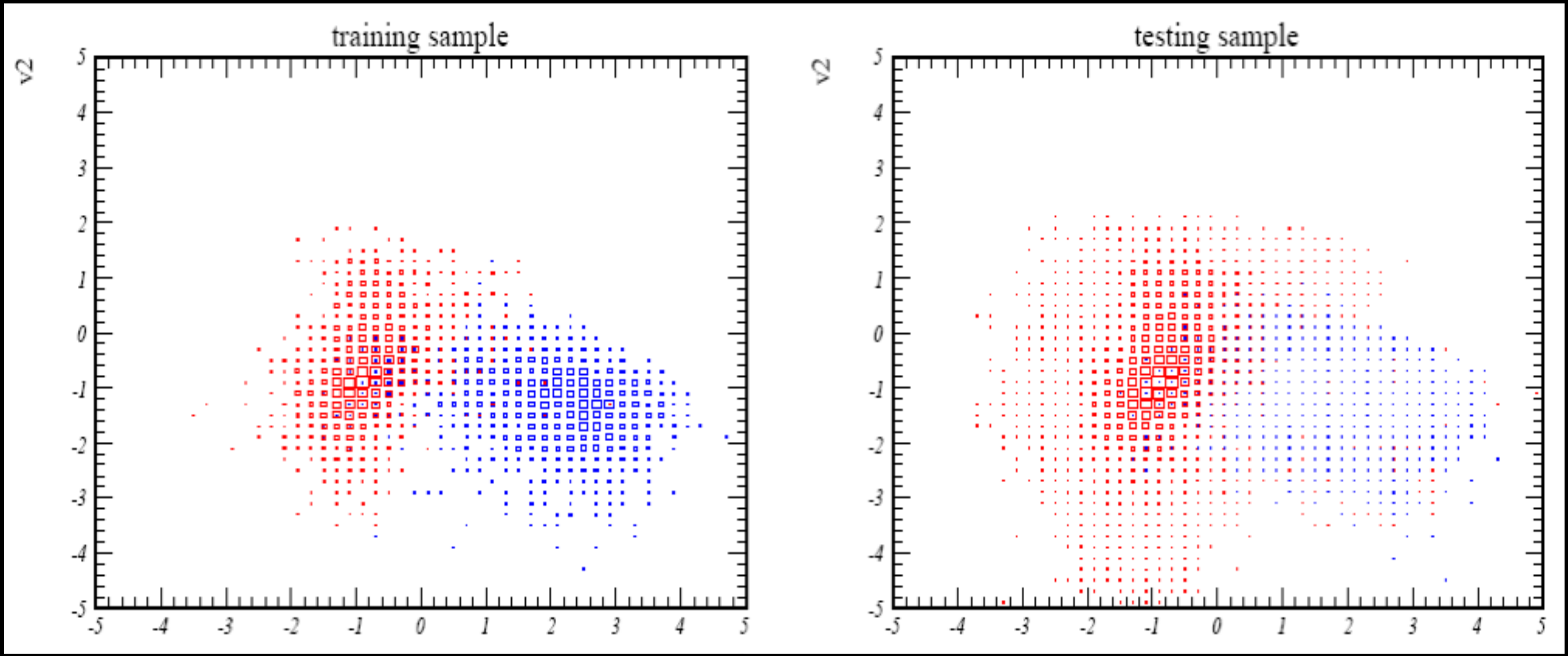


.. and hence will chose: “cut b)”: Var0 < -0.5

The combined classifier: Tree1 + Tree2 the (weighted) average of the response to a test event from both trees is able to separate signal from background as good as one would expect from the most powerful classifier



Training and test sample



- training and test sample do not have to be identical
- Can train on different ratios of signal and background events

General Advice for (MVA) Analyses

There is no magic in MVA-Methods:

- you typically still need to make careful tuning and do some “hard work”
- no “artificial intelligence” … just “fitting decision boundaries” in a given model

The most important thing at the start is finding good observables

- good separation power between S and B
- little correlations amongst each other
- no correlation with the parameters you try to measure in your signal sample!

Think also about possible combination of variables

- this may allow you to eliminate correlations
  - rem.: you are MUCH more intelligent than your computer

Apply pure preselection cuts and let the MVA only do the difficult part.

“Sharp features should be avoided” → numerical problems, loss of information when binning is applied

- simple variable transformations (i.e. log(variable)) can often smooth out these areas and allow signal and background differences to appear in a clearer way

Treat regions in the detector that have different features “independent”

- can introduce correlations where otherwise the variables would be uncorrelated!

Let’s look at some BDT examples in Scikit learn

Artificial Neural Networks

Artificial Neural Networks

Human brain

- Neuron switching time ~.001 second
- Number of neurons ~10^{10}
- Connections per neuron ~10^{4-5}
- Scene recognition time ~.1 second
- Requires a lot of training
- Lots of parallel computation!

Artificial neural networks (ANNs):

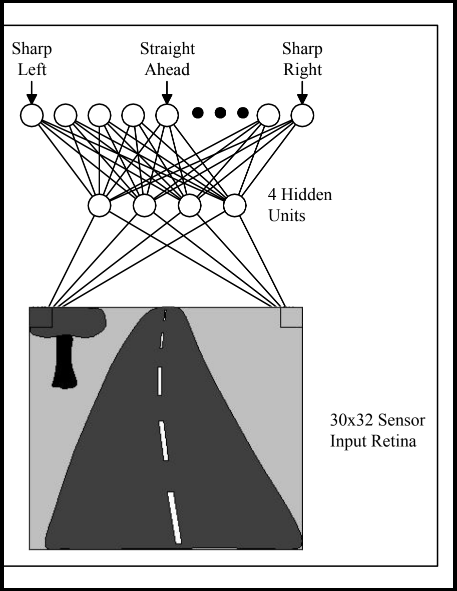
- Many neuron-like threshold switching units

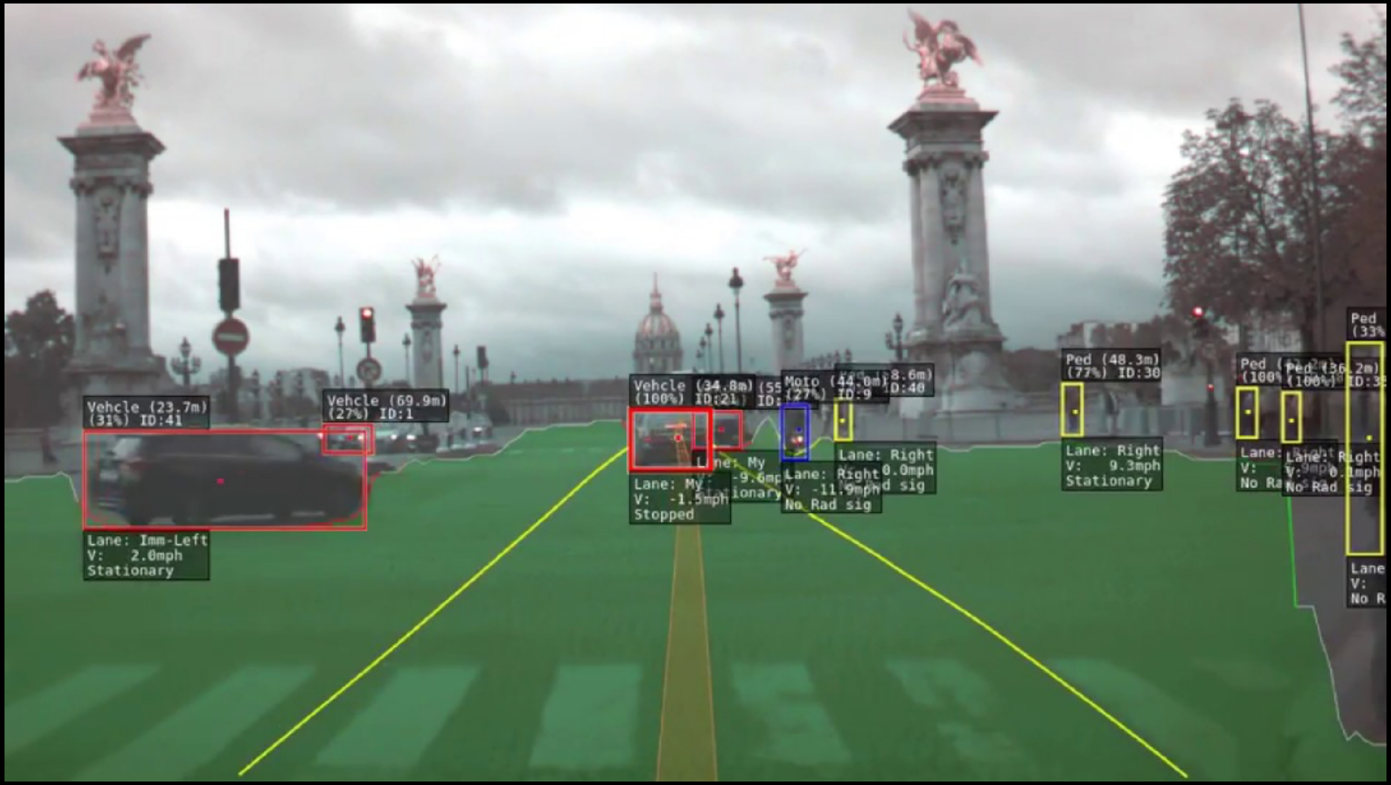
When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g., combining information from different measurements or detectors)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is *unimportant*

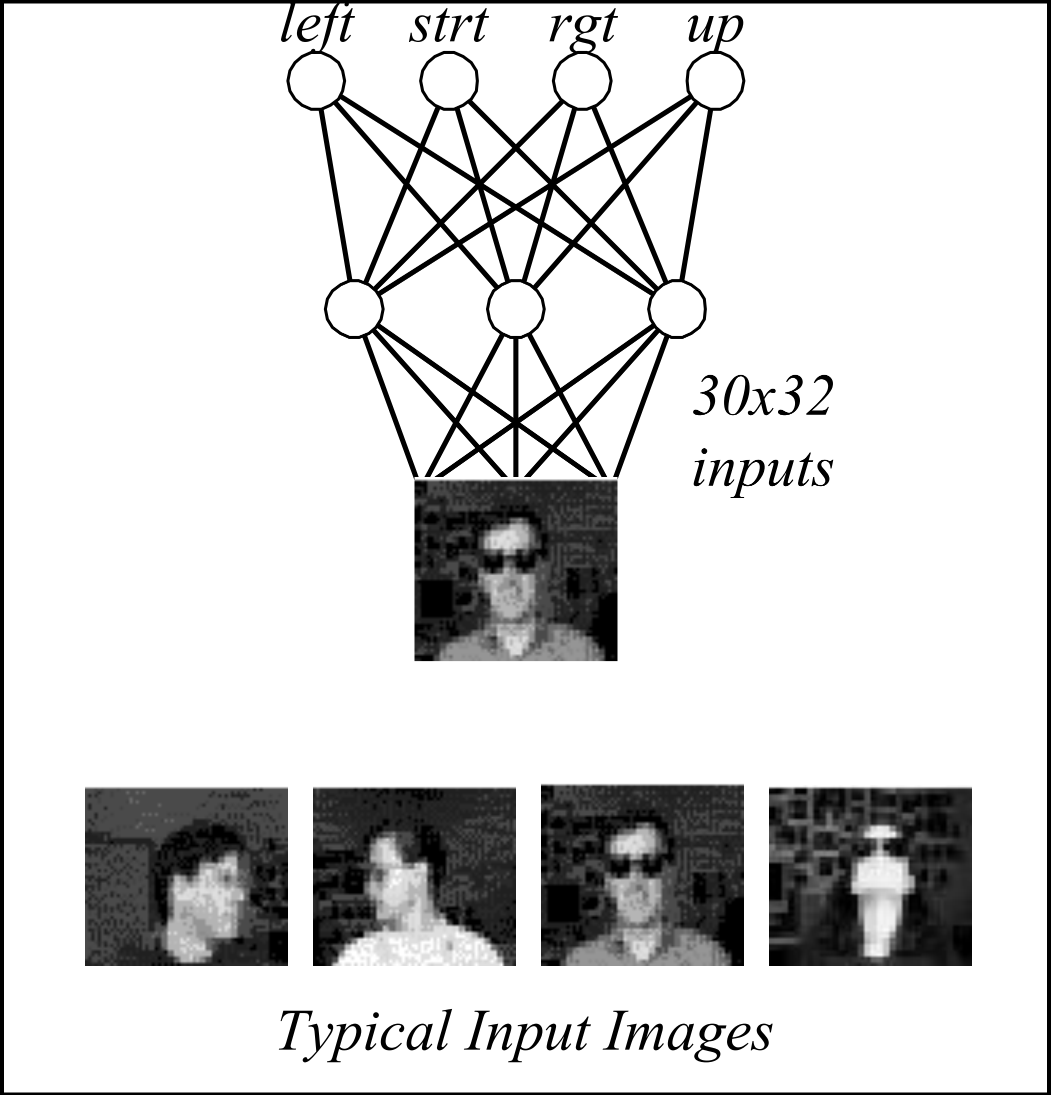
Examples:

ANN drives through Paris



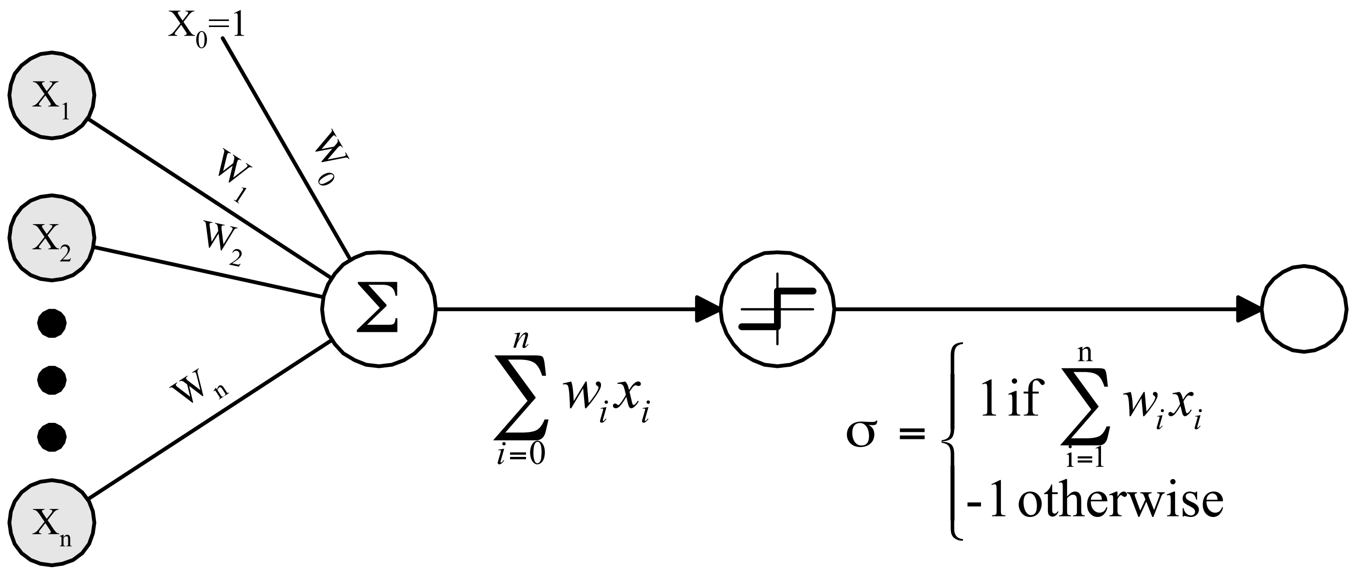

Vehcle (23.7m) (31%) ID:41
Lane: Imm-Left V: 2.0mph Stationary
Vehcle (69.9m) (27%) ID:1
Vehcle (34.8m) (100%) ID:21
Lane: My V: -1.5mph Stopped
Moto (44.0m) (27%) ID:9
Lane: Right V: -11.9mph No Rad sig
Ped (48.3m) (77%) ID:30
Lane: Right V: 9.3mph Stationary

ANN recognizes faces



Typical Input Images

Basic unit: Perceptron



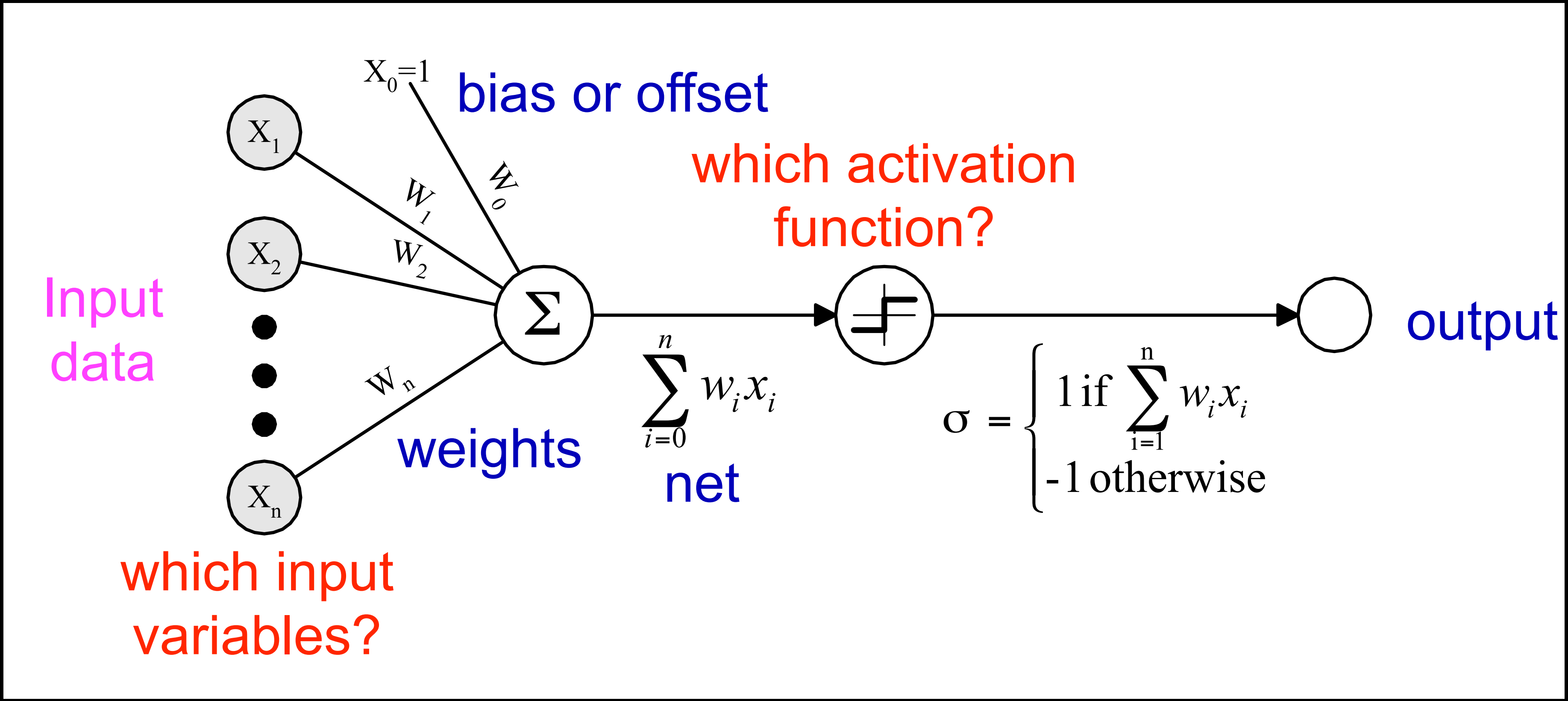
$X_0=1$
X_1
X_2
X_n
W_0
W_1
W_2
W_n
Σ
$\sum_{i=0}^{n} w_i x_i$
$\sigma = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i x_i \\ -1 \text{ otherwise} \end{cases}$

$$o(x_1,\ldots,x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

Sometimes we will use simpler vector notation :

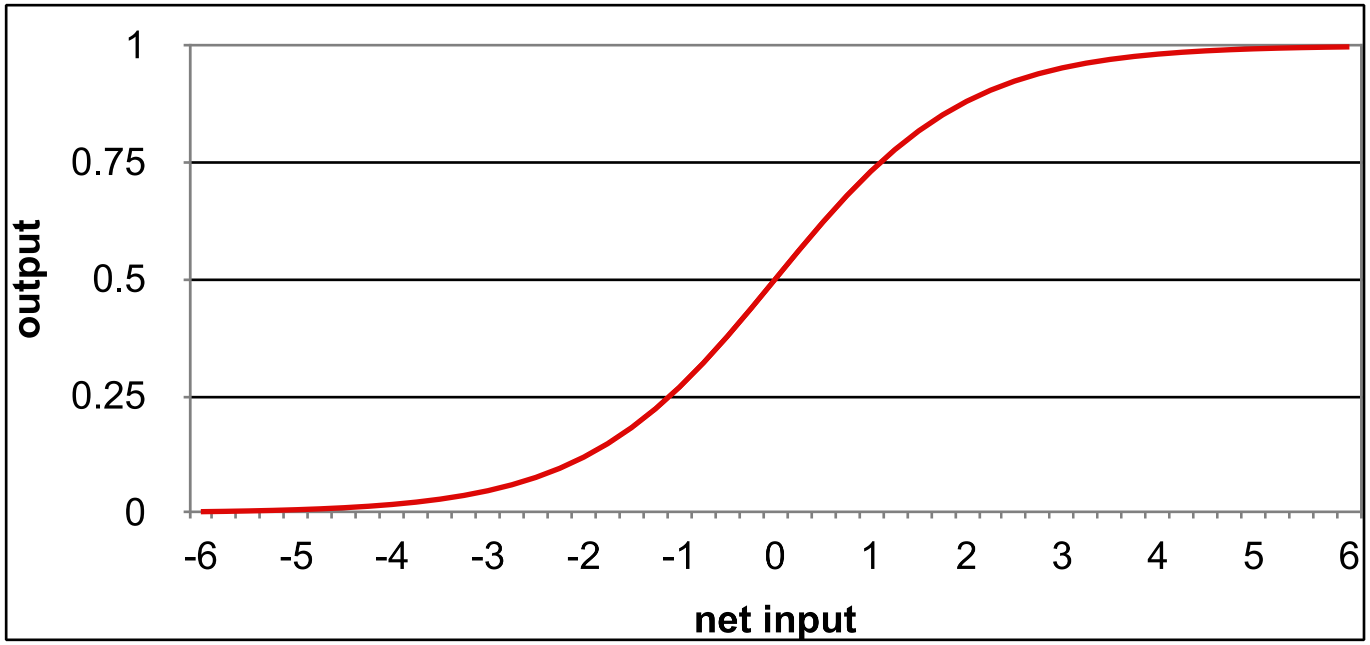
$$o(\vec{x}) = \begin{cases} 1 \text{ if } \vec{w} \cdot \vec{x} > 0 \\ -1 \text{ otherwise} \end{cases}$$

Basic unit: Perceptron



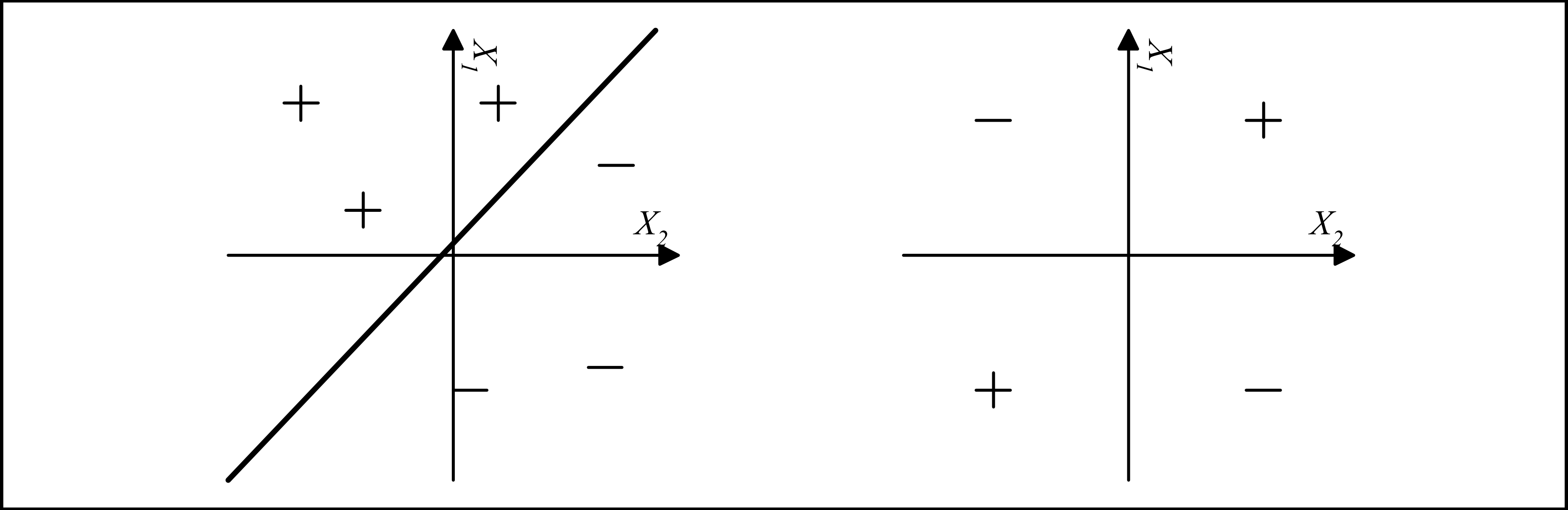
The Sigmoid Function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- “rounded” step function
- Unlike step function, can take derivative → helpful for learning

Perceptron decision boundaries



Represents some useful functions

- What weights represent $g(x_1,x_2) = AND(x_1,x_2)$?

But some functions not representable

- e.g., not linearly separable
- therefore, we will want networks of perceptrons

How to train your Perceptron

$$w_i \leftarrow w_i + \Delta w_i$$

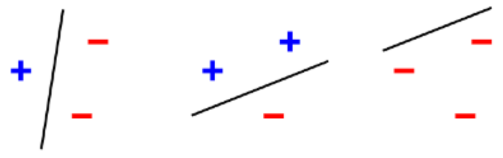
where

$$\Delta w_i = \eta\,(t - o)x_i$$

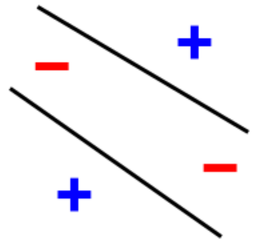
- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called learning rate

Can prove it will converge

- If training data is linearly separable
- and η is sufficiently small



linearly separable



not l.s.

Again: Loss function

Now we just need to fix the parameters (weights) → Minimize Loss function E(**w**):

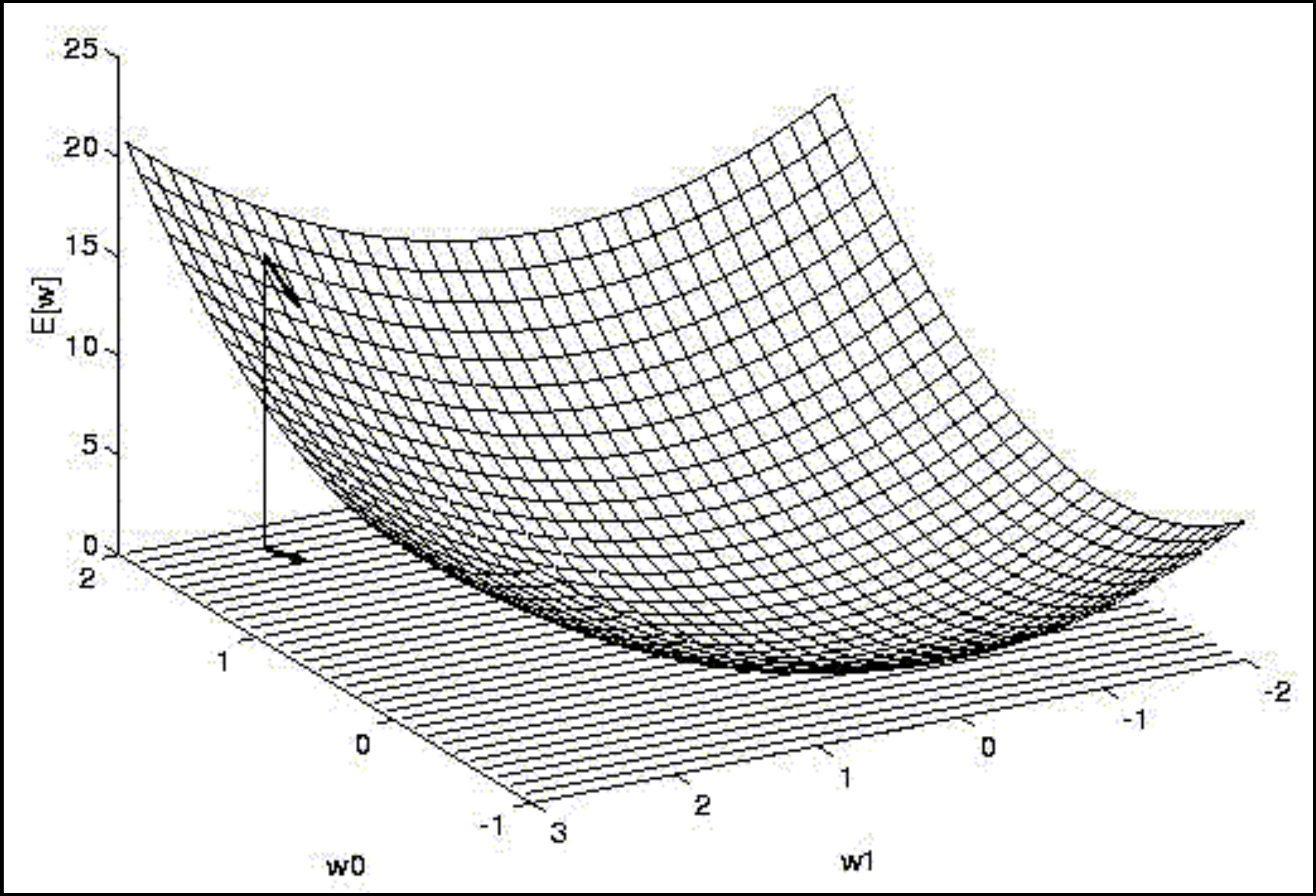
$$E(\mathbf{w}) = \sum_{i}^{events} \Big(\underbrace{y_i^{train}}_{\text{true}} - \underbrace{y(x_i)}_{\text{predicted}} \Big)^2$$

← i.e. like usual χ^2 in fitting

How to find weights w that minimize loss function?

- One global fit will usually not work
  - noisy data → many local minima
- Solution: Gradient descent optimization

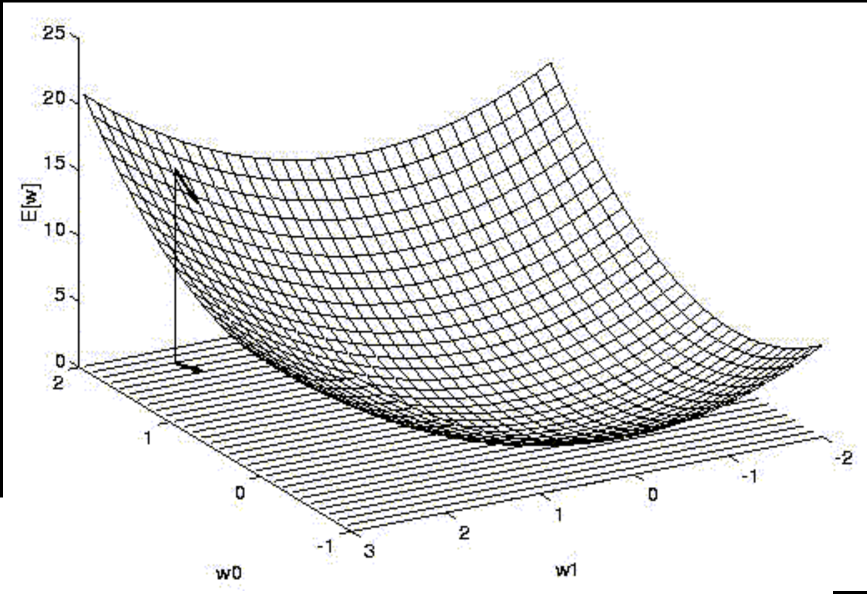
Gradient Descent



Make steps in the direction of the steepest slope of E($\mathbf{w}$)

Gradient Descent

Finding the optimal vector of weights **w**



Gradient $$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$ Slope of E(w) surface

Training rule: $\Delta w_i = -\eta \, \nabla E[\vec{w}]$ small step η down slope ∇E

i.e., $$\Delta w_i = -\eta \, \frac{\partial E}{\partial w_i}$$

Gradient Descent Training

```
GRADIENT – DESCENT($training\_examples$, $\eta$)
 Each training examples is a pair of the form $<\vec{x}, t>$, where $\vec{x}$ is the vector of
 input values and t is the target output value. $\eta$ is the learning rate (e.g., .05).
• Initialize each $w_i$ to some small random value
• Until the termination condition is met, do
    - Initialize each $\Delta w_i$ to zero.
    - For each $<\vec{x}, t>$ in $training\_examples$, do
        * Input the instance $\vec{x}$ and compute output $o$
        * For each linear unit weight $w_i$, do
              $\Delta w_i \leftarrow \Delta w_i + \eta\,(t - o)x_i$
    - For each linear unit weight w, do
        $w_i \leftarrow w_i + \Delta w_i$
```

→ Obtain set of weights for which output is closest to target, as measured by Loss Function

Gradient Descent

Differentiate using chain rule

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

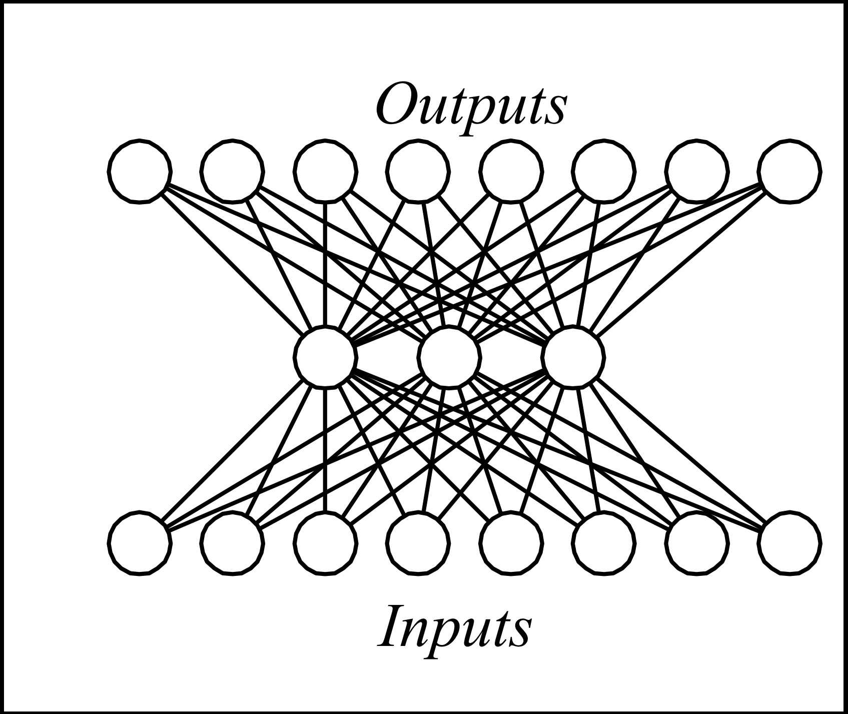
$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$

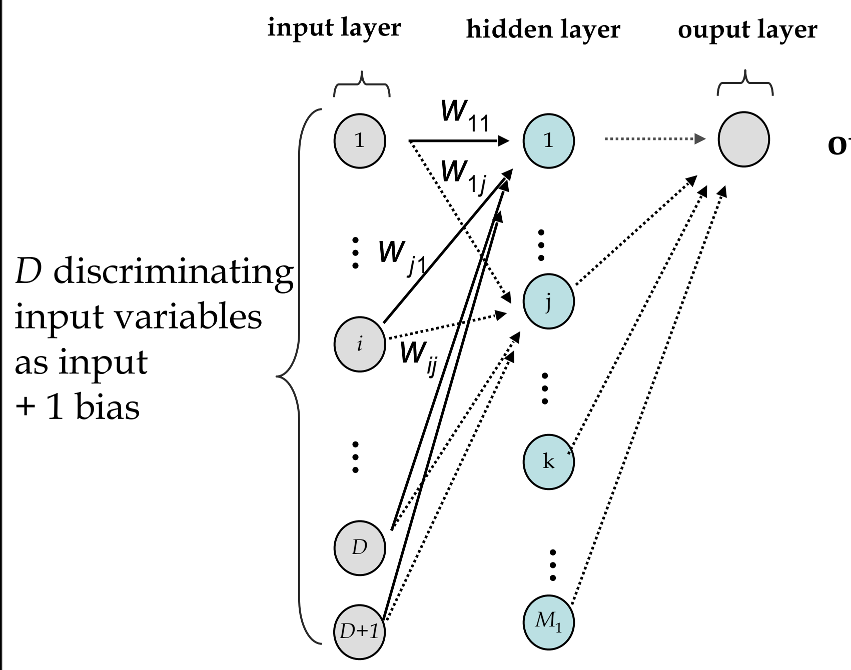
$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

Networks of perceptrons

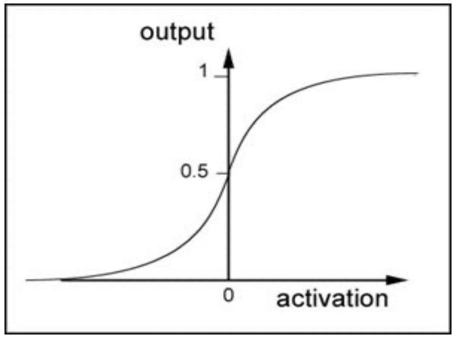
- Linear units correspond to *hyperplanes* as decision boundary
- How to approximate arbitrary hyper surfaces?
- → Multi-layer perceptron (MLP)



Multilayer Perceptron - MLP



output: $$y(x)=\sum_{i}^{M} w_{0i} A\left(w_{i0} + \sum_{j=1}^{D} w_{ij} \cdot x_j \right)$$



"Activation" function e.g. sigmoid:

$$A(x) = \left(1+e^{-x}\right)^{-1}$$

or tanh
or …

- Nodes in hidden layer have "activation functions" whose arguments are linear combinations of input variables → non-linear response to the input
- The output is a linear combination of the output of the activation functions at the internal nodes
- Input to the layers from preceding nodes only → feed forward network (no backward loops)
- It is straightforward to extend this to additional layers

Backpropagation

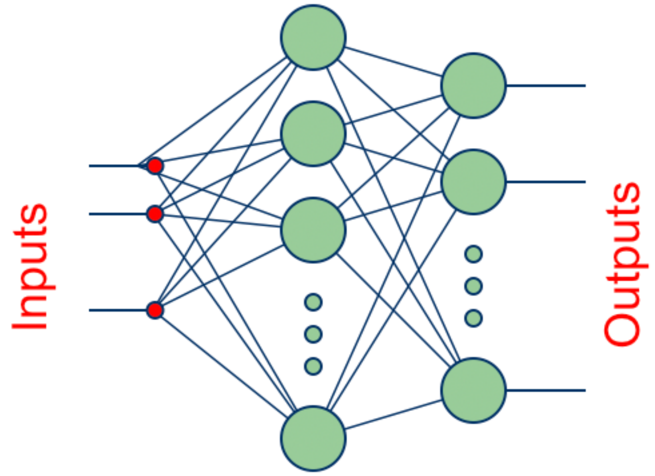
- How to change weights for hidden layers?
- Can’t take derivative of E wrt hidden layer weights directly
- Use same idea (gradient descent), but recursively
- Start with output layer, calculate update to weights from hidden layer
- Then update hidden layer weights
  - continue if multiple hidden layers
  - repeat until satisfied with network performance

Backpropagation

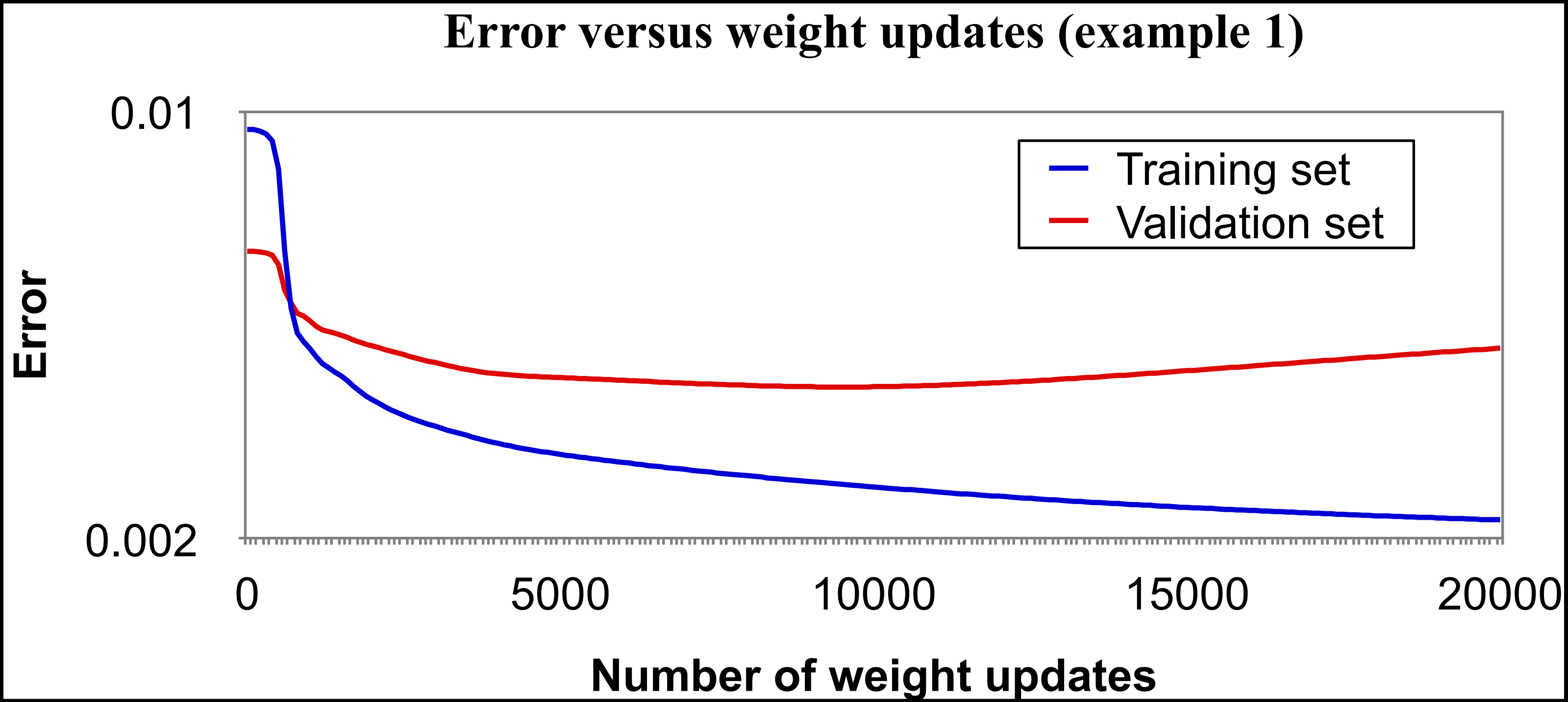
Initialize all weights to small random numbers. Until satisfied, do

- For each training example, do

```
1. Input the training example and compute the outputs
2. For each output unit $k$
       $\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$
3. For each hidden unit $h$
       $\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{h,k}\delta_k$
4. Update each network weight $w_{i,j}$
       $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$
   where
       $\Delta w_{i,j} = \eta\,\delta_j x_{i,j}$
```



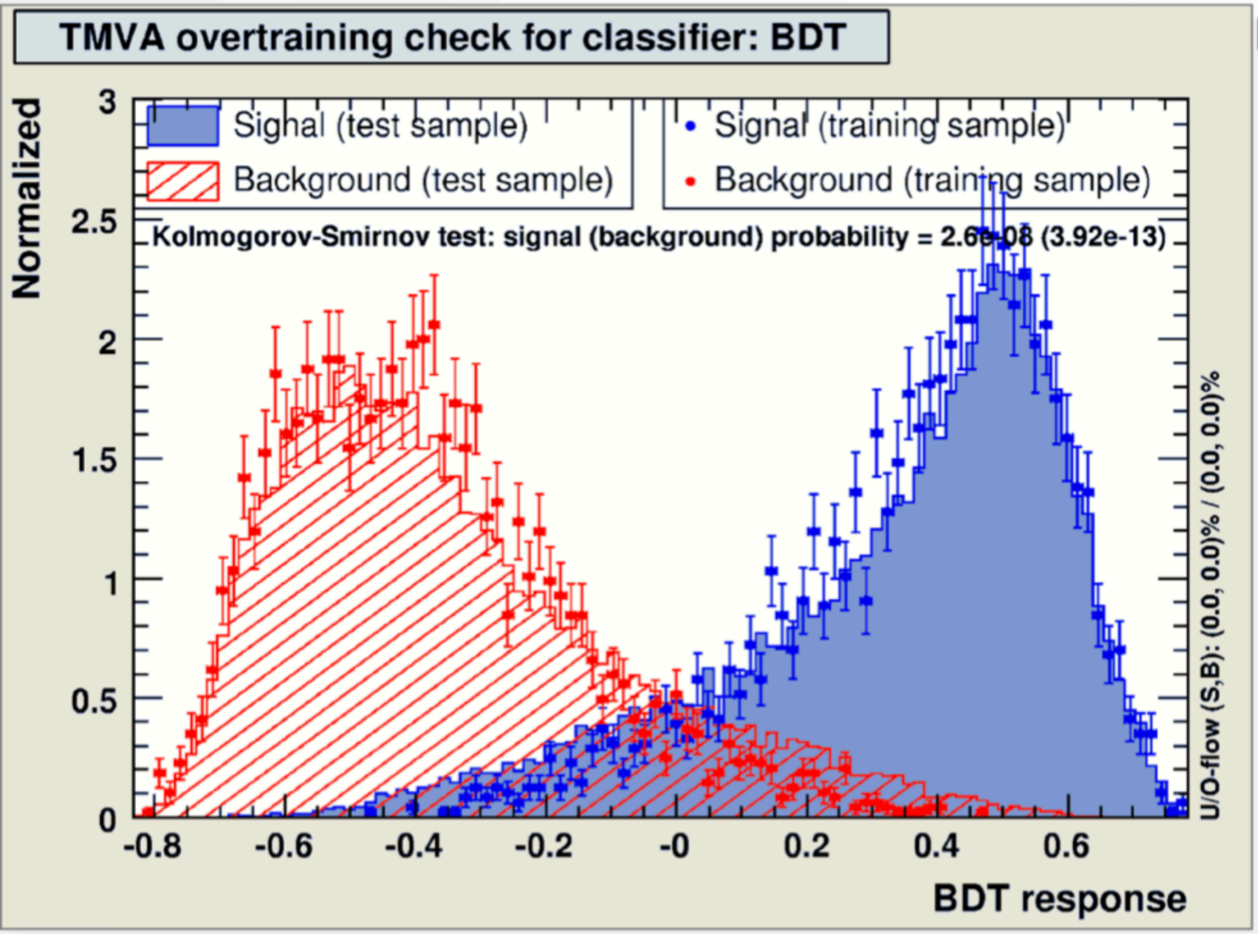
Convergence/Overfitting in ANNs



Train on ***Training set***, check results
on independent ***Validation set*** *(or Test set)*

Overtraining MVAs

- Check for overtraining: classifier output for test *and* training samples



- Remark on **overtraining**
  - Occurs when classifier training has too few degrees of freedom because the classifier has too many adjustable parameters for too few training events
  - Sensitivity to overtraining depends on classifier: *e.g.*, Fisher weak, BDT strong
  - Compare performance between training and test sample to detect overtraining
  - Actively counteract overtraining: *e.g.*, smooth likelihood PDFs, prune decision trees

Let’s look at some MLP examples in Scikit learn