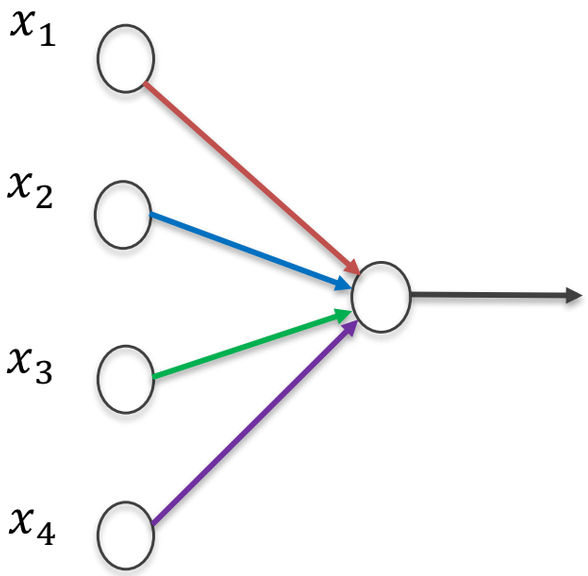
Stochastic Gradient Descent, The Backprop Algorithm, and Applications in Astronomy

Rafael Martínez-Galarza
Harvard-Smithsonian Center for Astrophysics

What is learning?

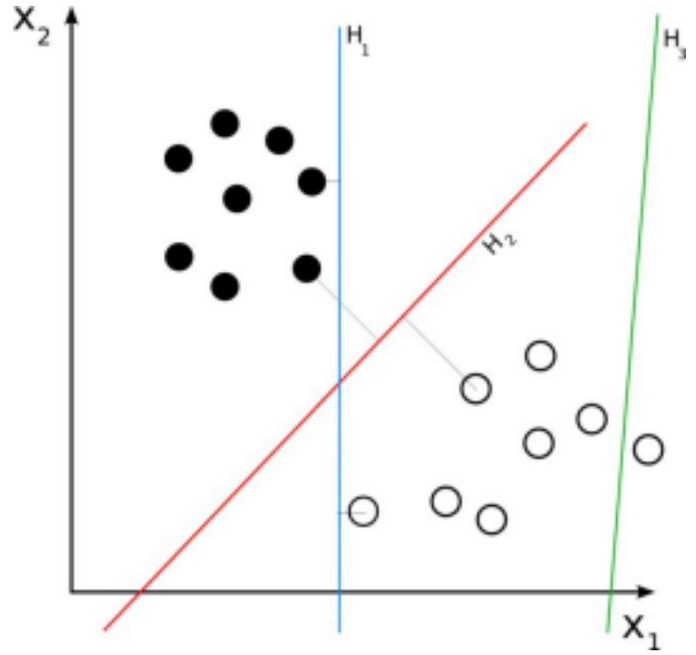
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



$$Y = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4)$$

Intercept or **Bias**

Coefficients or **Weights**



$$o = g(\vec{w} \cdot \vec{x} + b).$$

- Learning is adjusting the weights and bias to find the best possible decision boundary.
- Learning is performed on the TRAINING DATA.
- Another way to see it: adjust the weights and bias so that g(w*x +b) -the probability- is high if dot is black and low otherwise.

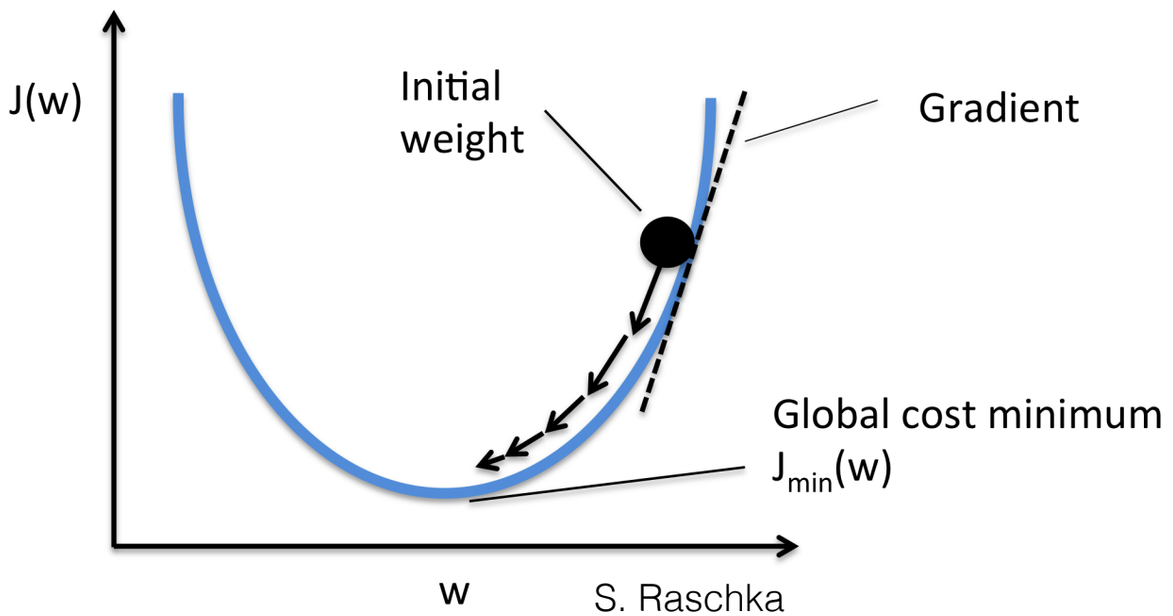
Error Function

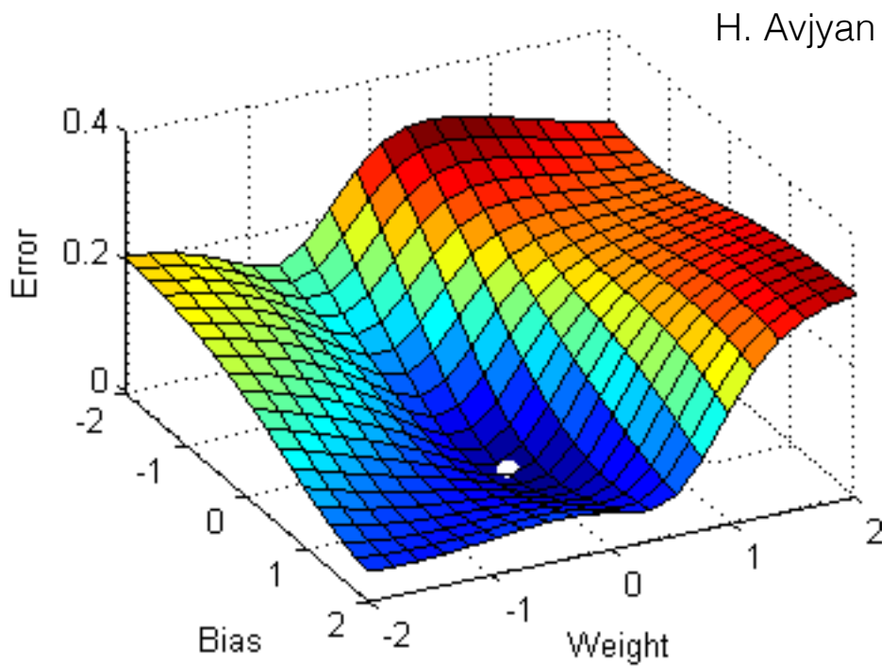
- How do we adjust the weights?
- We need to quantify the difference between the perceptron output and the true value y for an input **x**, over a set of input-output pairs.
- The mean square error (MSE) is a natural choice:

$$E(X) = \frac{1}{2N}\sum_{i=1}^{N}(o_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(g(\vec{w}\cdot\overrightarrow{x_i} + b) - y_i\right)^2$$

- Note that E(X) = 0 when $o_i = y_i$ for all input-output pairs.
- Therefore, we want to MINIMIZE E(X)
- How do we do that?

Gradient Descent





- Start with random bias and weights
- Use gradient of error function w.r.t. the bias and weights at each point to update those values.
- Iterate until certain convergence criterium is met.

$$\overrightarrow{w_{i+1}} = \overrightarrow{w_i} - \alpha \frac{\partial E(X)}{\partial \overrightarrow{w_i}}$$

$$b_{i+1} = b_i - \alpha \frac{\partial E(X)}{\partial b_i}$$

Learning rate!

Parenthesis: Stochastic GD

- Remember that the error function is a sum over all examples in the training set:

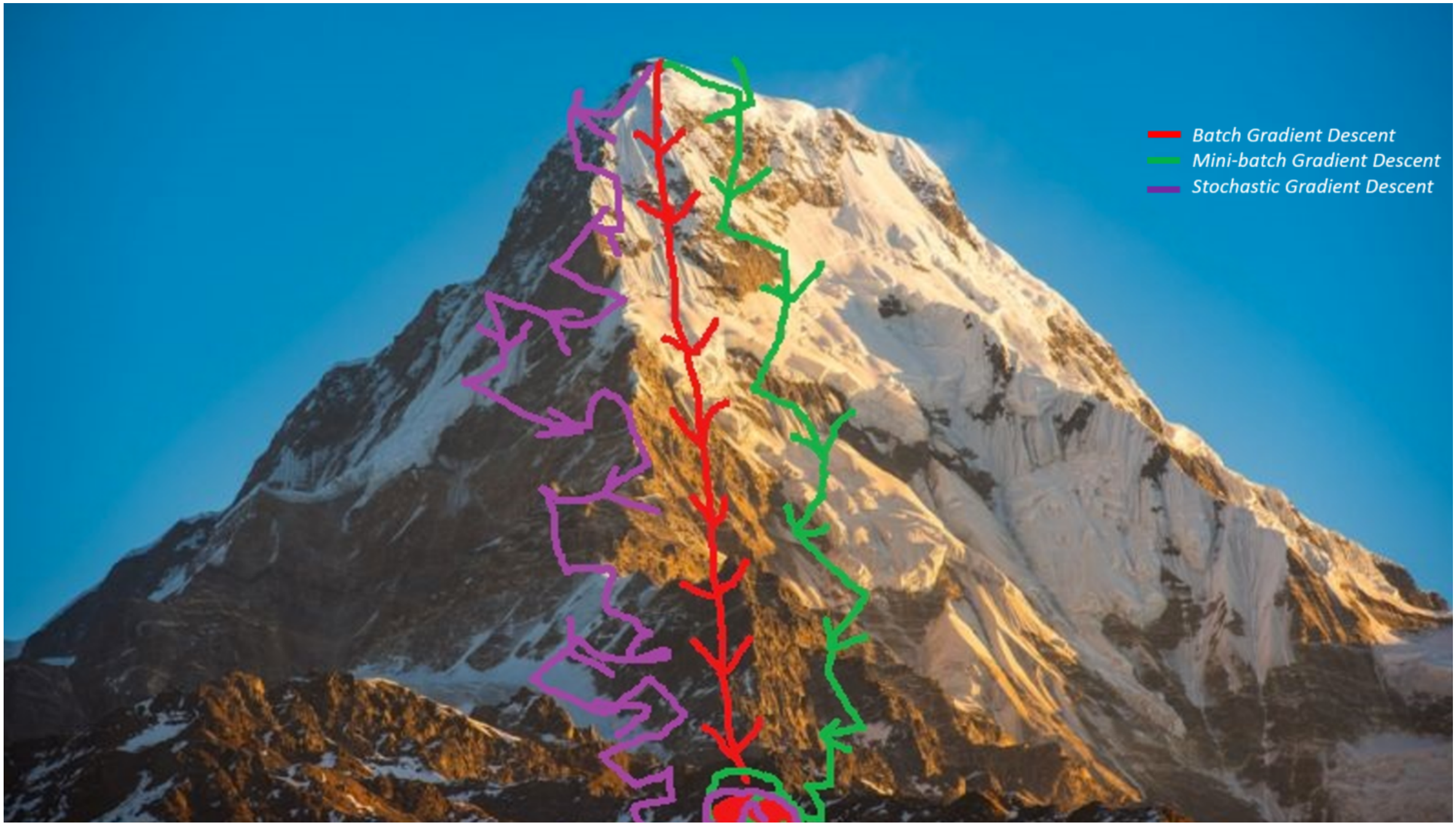
$$E(X) = \frac{1}{2N}\sum_{i=1}^{N}(o_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(g(\vec{w}\cdot\overrightarrow{x_i} + b) - y_i\right)^2$$

- For a single update of weights, I need to go over the entire data set. Not efficient.

- Solution 1: approximate the gradient using a single random example. Not too accurate.

- Solution 2: Perform an update using a mini-batch of examples.

Parenthesis: Stochastic GD



I. Dabbura

Relevant tensors

DEFINITION

w_{ij}^k : weight for node j in layer l_k for incoming node i

b_i^k : bias for node i in layer l_k

a_i^k : product sum plus bias (activation) for node i in layer l_k

o_i^k : output for node i in layer l_k

r_k : number of nodes in layer l_k

---

g : activation function for the hidden layer nodes

g_o : activation function for the output layer nodes

Training the MLP: Backpropagation

- Just as before, iteratively update the values of w and b until E(X) is minimized.

$$E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2$$

- So we want to know how the change in E(X) depends on changes of each weight. **Problem is that intermediate layer neurons have no target output.**
- But E(X) is a sum over examples, and so it's derivative with respect to each weight is a sum of derivatives:

$$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial}{\partial w_{ij}^k}\left(\frac{1}{2}\left(\hat{y}_d - y_d\right)^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ij}^k}$$

- So all we care about is the error for a single example:

$$E = \frac{1}{2}\left(\hat{y} - y\right)^2$$

Training the MLP: Backpropagation

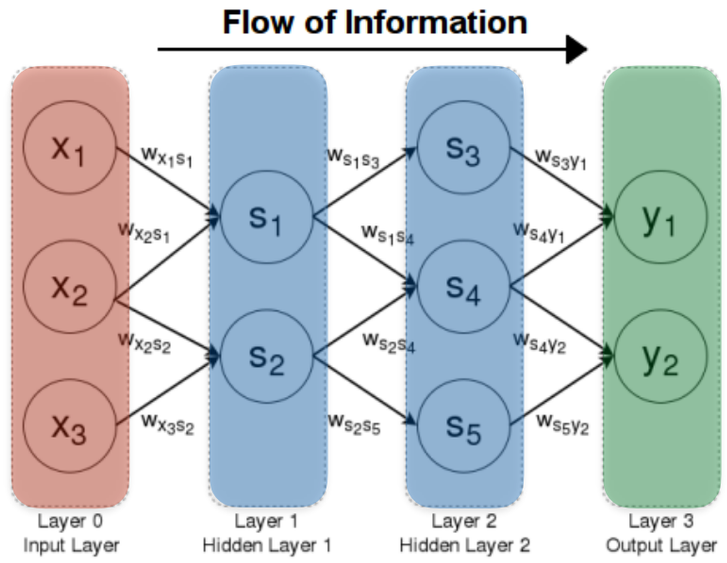
- Now, we can make use of the chain rule:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k}$$

$$\delta_j^k \equiv \frac{\partial E}{\partial a_j^k}$$

- But:

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k} \left(\sum_{l=0}^{r_{k-1}} w_{lj}^k o_l^{k-1} \right) = o_i^{k-1}$$



- And therefore:

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}$$

- **The partial derivative of the error with respect to the weight for neuron *j* in layer *k*, for incoming neuron *i*, is the product of its derivative with respect to the linear activation in that neuron and the output of the incoming neuron in layer *k-1*.**

Training the MLP: Backpropagation

- Hey, but we have not calculated $\delta_j^k \equiv \frac{\partial E}{\partial a_j^k}$
- Spoiler: it depends on the values of error terms in the next layer - that's why we need to go backwards.
- **Part 1**: Output layer. Let's find δ_1^m first. The error function for this layer is:

$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}\left(g_o(a_1^m) - y\right)^2$$

- Lets apply the chain rule:

$$\delta_1^m = \left(g_0(a_1^m) - y\right) g_o'(a_1^m) = (\hat{y} - y) g_o'(a_1^m)$$

- Therefore:

$$\frac{\partial E}{\partial w_{i1}^m} = \delta_1^m o_i^{m-1} = (\hat{y} - y) g_o'(a_1^m)\, o_i^{m-1}$$

- Easy¡

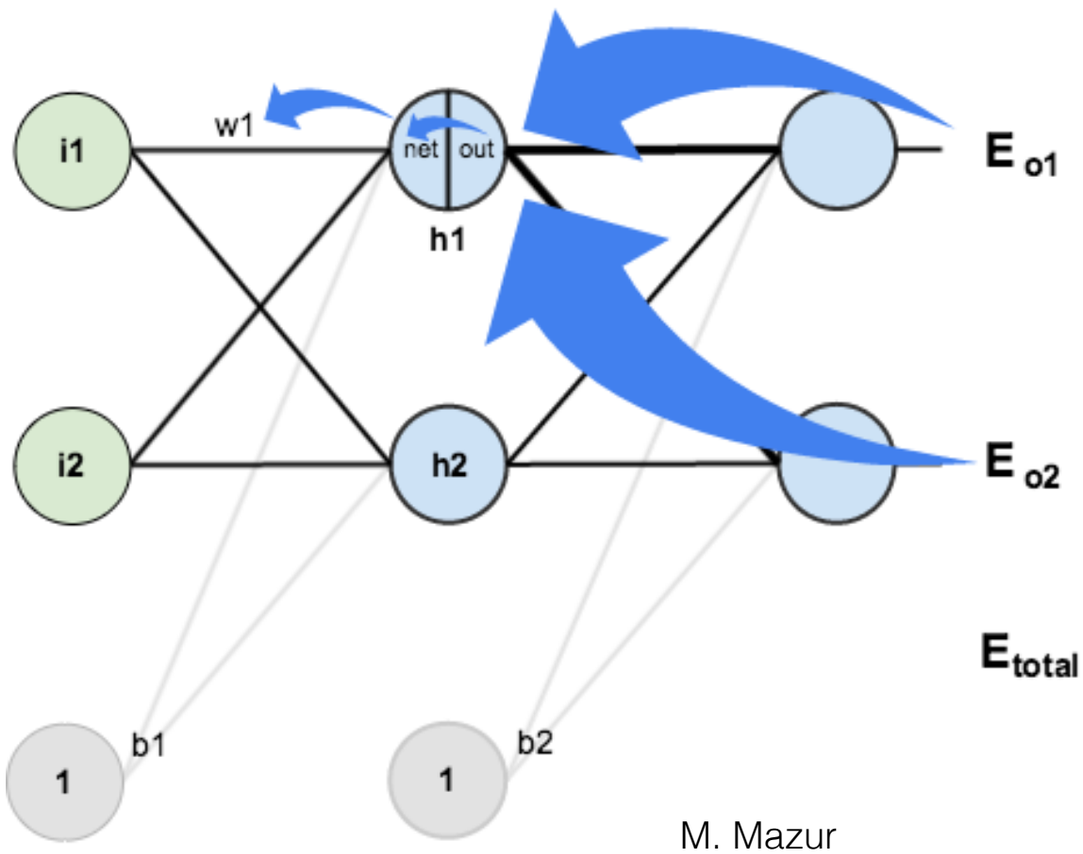
Training the MLP: Backpropagation

- **Part 2**: Hidden layers. For the output layer we have a target. Not so for the hidden layers. What to do?? Worry not, as the chain rule comes to the rescue:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$E_{total} = E_{o1} + E_{o2}$$

- Note that it depends on the values or error terms in the next layer - that's why we need to go backwards. Graphically:

M. Mazur

Training the MLP: Backpropagation

- Let's plug in some things we know:

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}$$

$$a_l^{k+1} = \sum_{j=1}^{r^k} w_{jl}^{k+1} g\left(a_j^k\right) \longrightarrow \frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'\left(a_j^k\right)$$

- Plugging this in the definition of the error term gives the **backpropagation formula:**

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'\left(a_j^k\right) = g'\left(a_j^k\right) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}$$

- And therefore the derivative of the total error w.r.t. the weight in a hidden layer is:

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = g'\left(a_j^k\right) o_i^{k-1} \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}$$

The general algorithm

1. For each example in the training set, calculate the forward phase starting with the input layer. Get output.

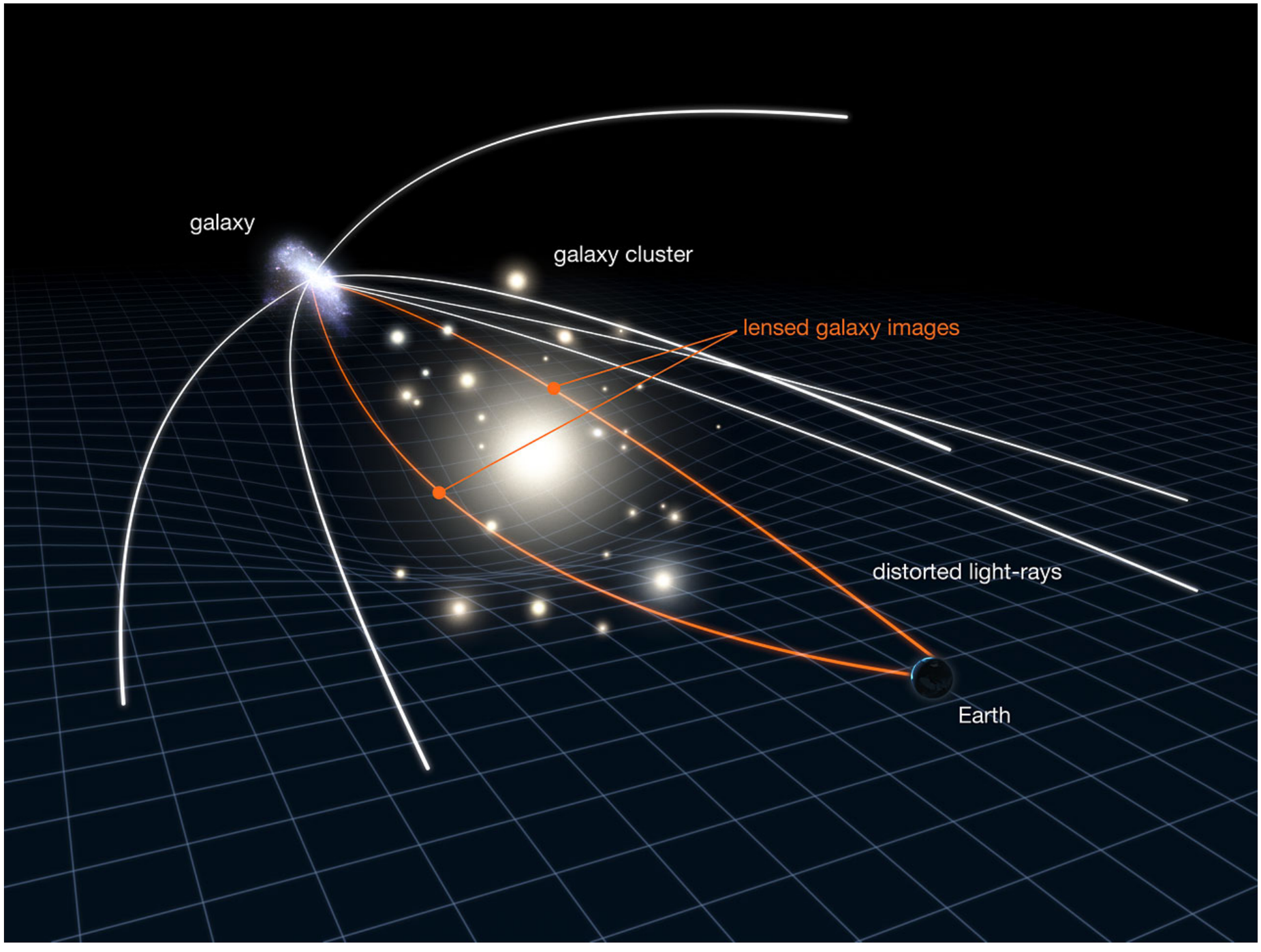
2. For each example, calculate the backward phase, starting from the error in the output layer, and propagating the error backwards:

    - Evaluate error term in the output layer.
    - Back propagate the error in the hidden layers using the formulas derived.
    - Evaluate the partial derivative of the total error wrt each weight

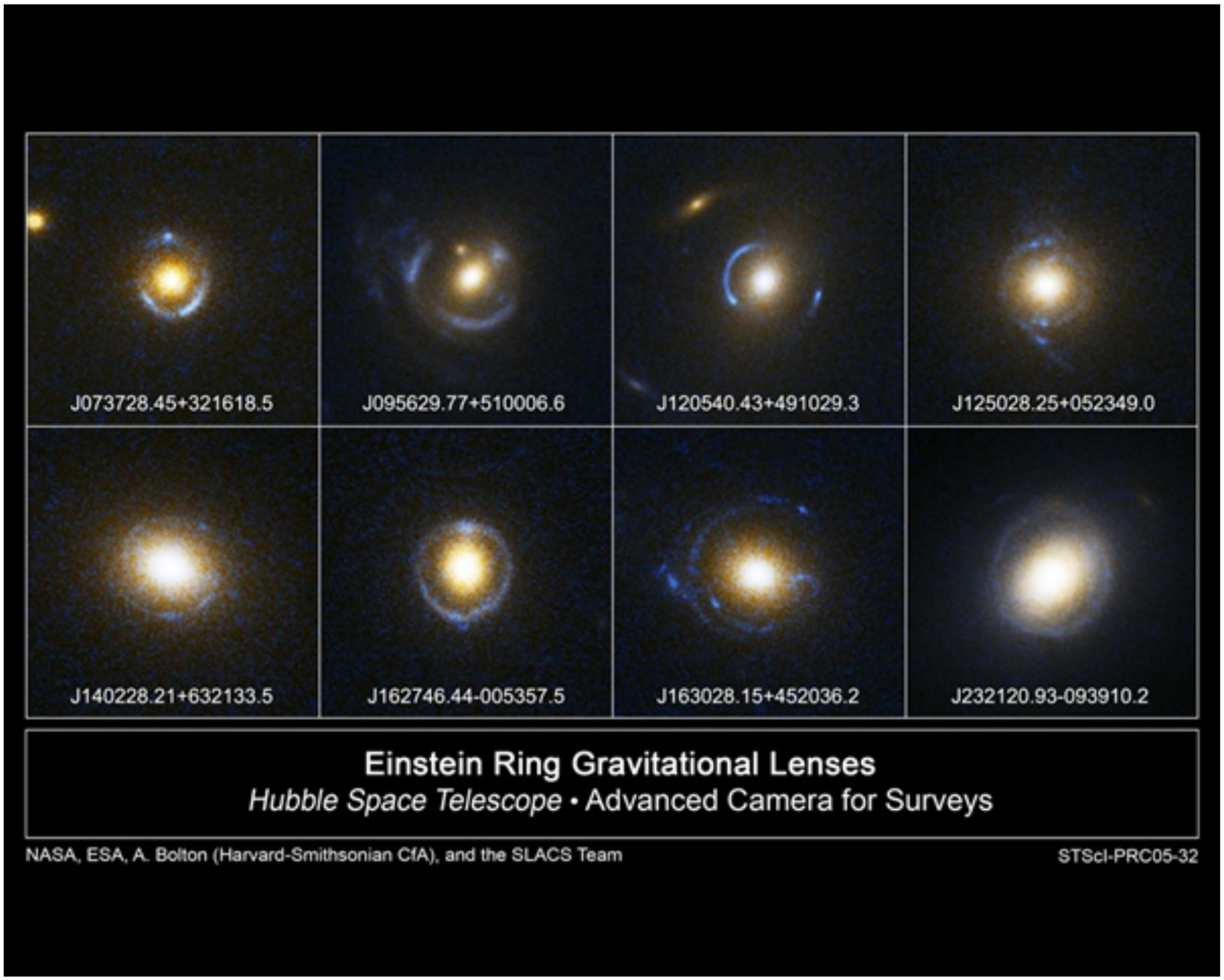
3. Combine the individual gradients by averaging them.

4. Update the weights with a certain learning rate: $w1+ = w1 - \eta * dE/dw1$

Gravitational Lensing

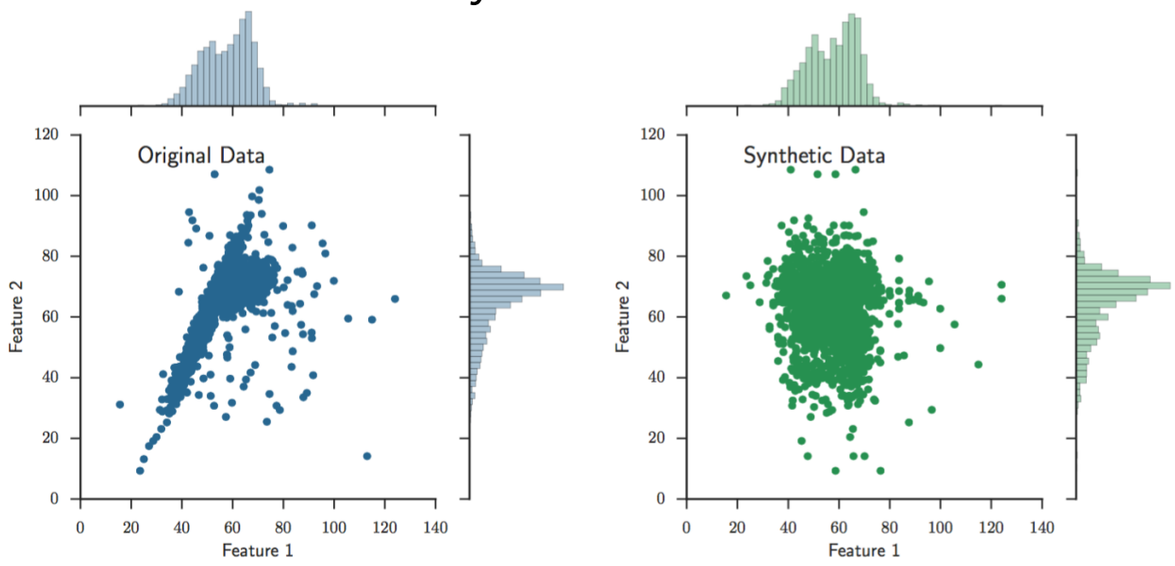


Gravitational lensing

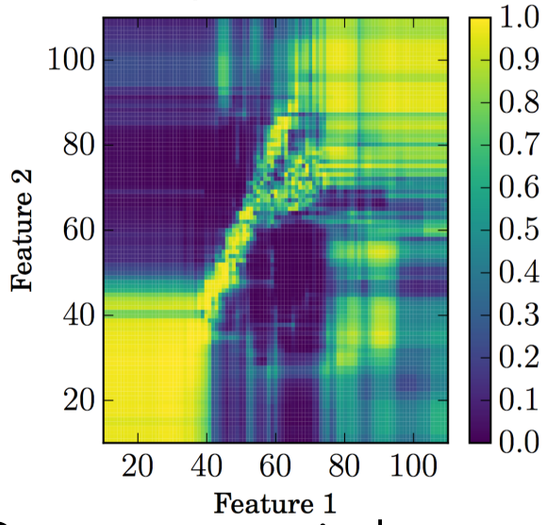


Unsupervised Random Forest

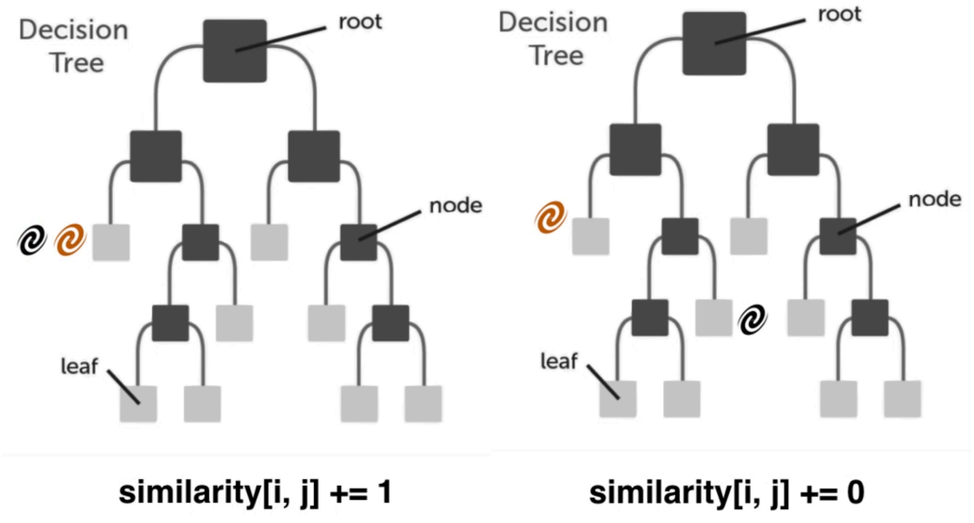
1. Generate synthetic dataset from



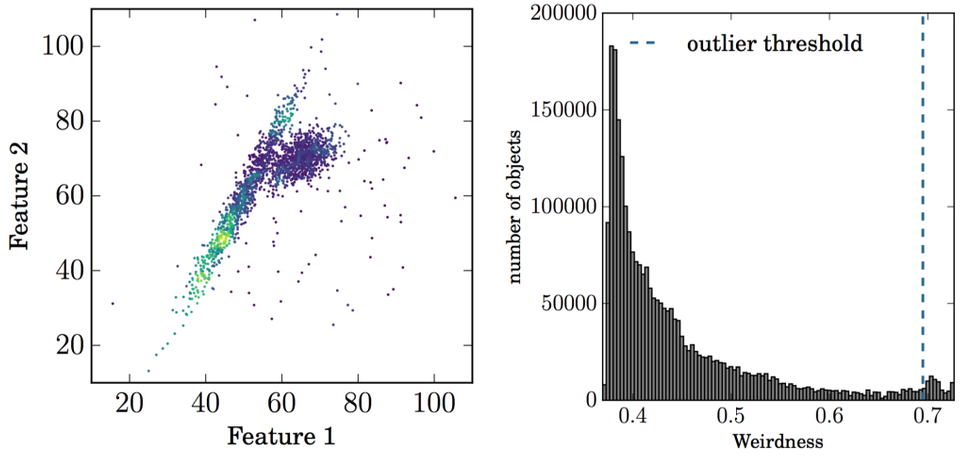
2. Train RF classifier with real+synthetic data



3. Propagate the real data



4. Compute weirdness



Baron & Poznanski , 2017