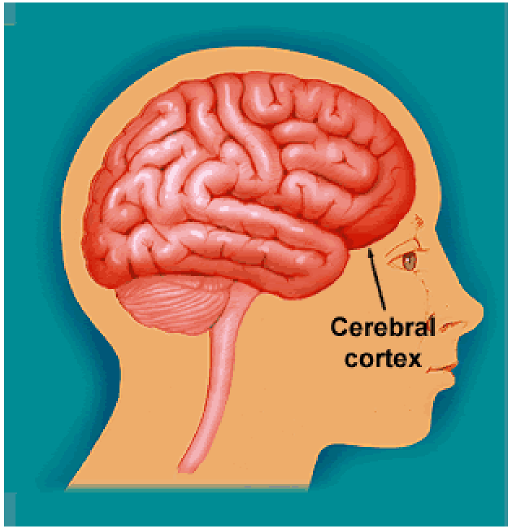
ErSE222: Machine learning in Geoscience

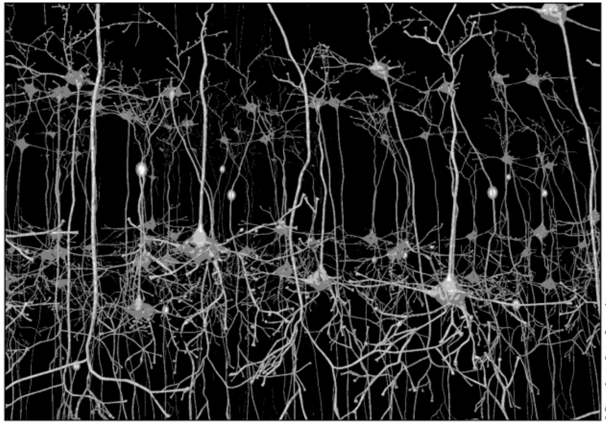
Feb. 9th, 2025

Tariq Alkhalifah and Omar Saad

Neural Network

- **Neuron**: the fundamental computational units
- **Synapses**: the connections between neurons
- **Layer**: neurons are organized into layers
- ***Extremely complex***: around 10^{11} neurons in the brain, each with 10^4 connections



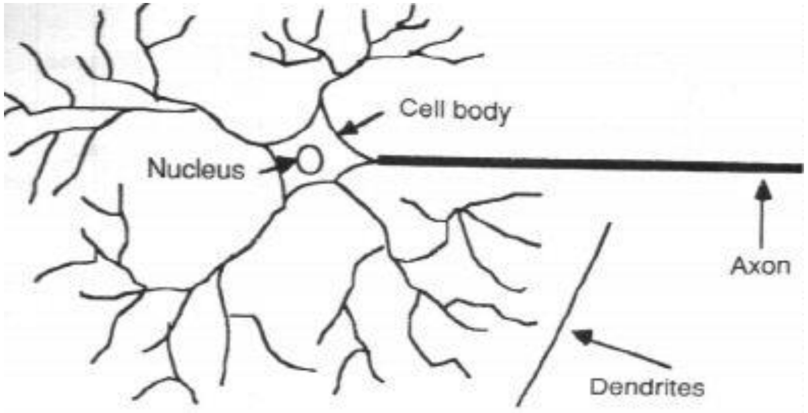


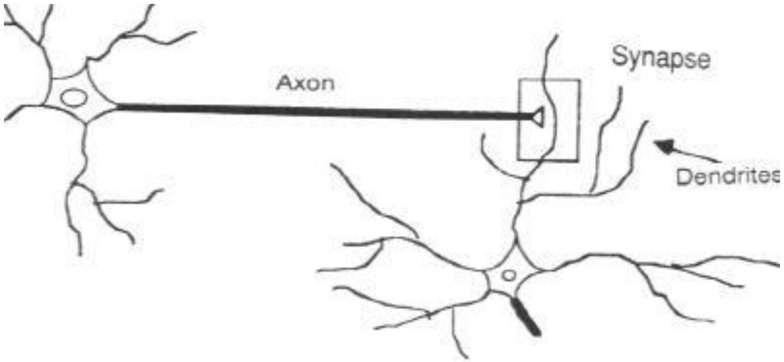
DIGITAL STUDIO SA

CG image of the vertical organization of neurons in the primary visual cortex (V1).
Smooth stellate and spiny stellate cells relay visual information coming out from the retina to pyramidal cells, themselves doing a first basic computation of visual motion perception.
version of July 2000

The Neuron - A Biological Information Processor

- *dendrites* - the receivers
- *soma* - neuron cell body (sums input signals)
- *axon* - the transmitter
- *synapse* - point of transmission



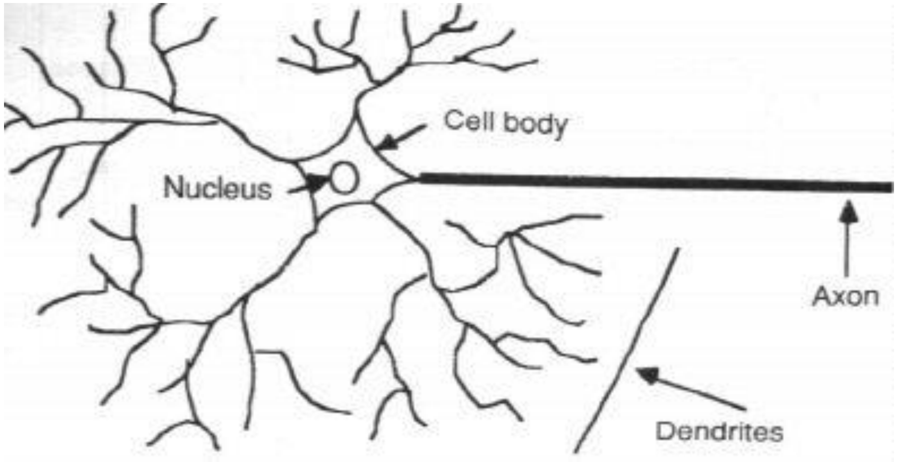


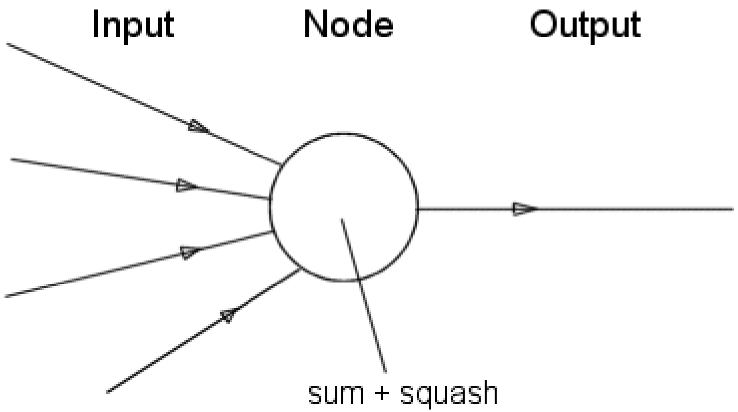
Neural Network

- 200 billion neurons, 32 trillion synapses
- Element size: 10^{-6} m
- Energy use: 25W
- Parallel, Distributed

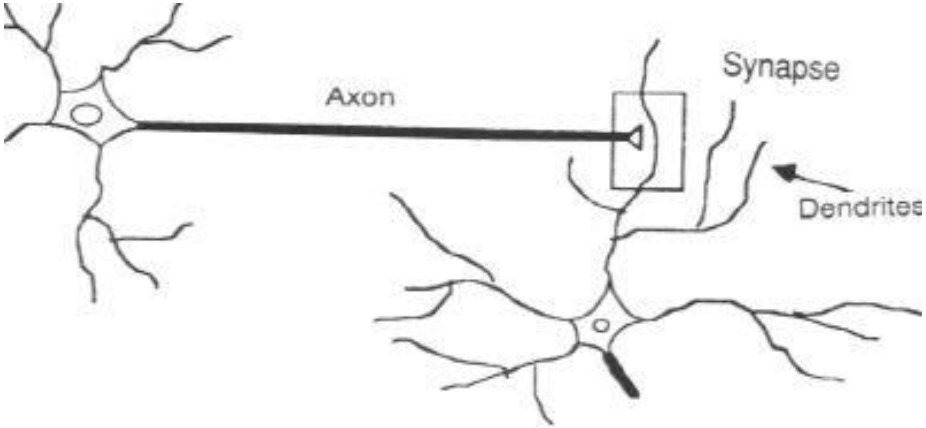
- 1 billion bytes RAM but trillions of bytes on disk
- Element size: 10^{-9} m
- Energy watt: 30-90W (CPU)
- Serial, Centralized

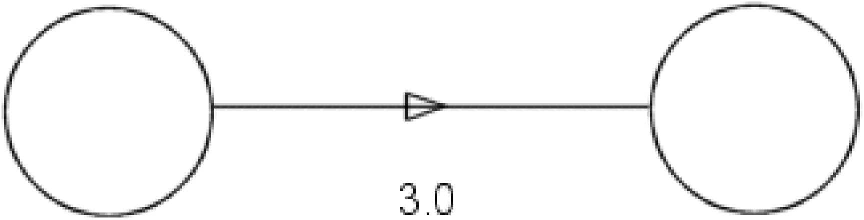
Neural Network





Neural Network





Learning algorithms

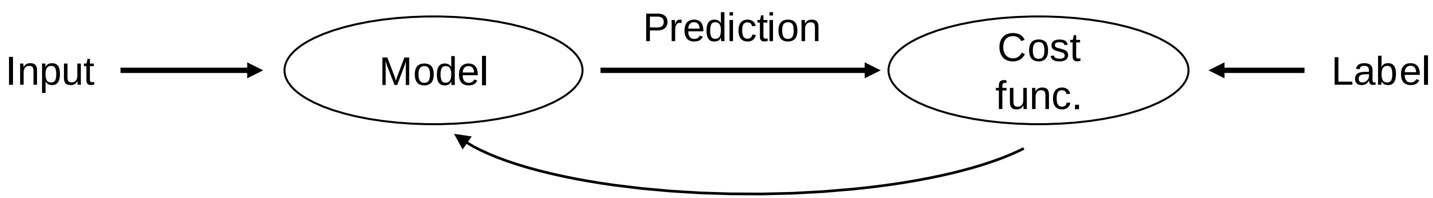
- Dataset: $\mathbf{x} = [x_1, x_2, \ldots, x_{N_f}]^T \quad \mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N_s)}]$ Inputs / features

  $\mathbf{y} = [y_1, y_2, \ldots, y_{N_t}]^T \quad \mathbf{Y} = [\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(N_s)}]$ Outputs / labels

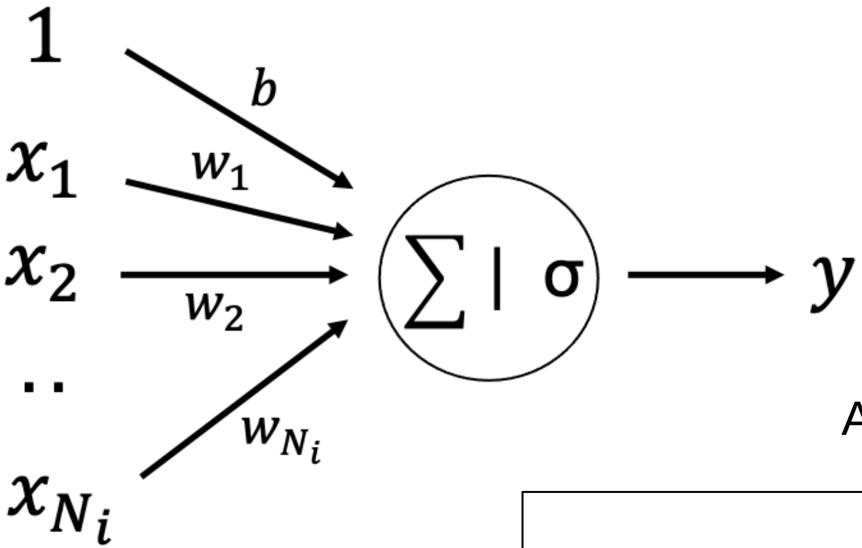
- Model: $\mathbf{y} = f_\theta(\mathbf{x})$

- Cost function: $J_\theta = \frac{1}{N_s} \sum_{j=1}^{N_s} \mathscr{L}(\mathbf{y}^{(j)}, f_\theta(\mathbf{x}^{(j)}))$

- Optimization algorithm: $\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, J_\theta$



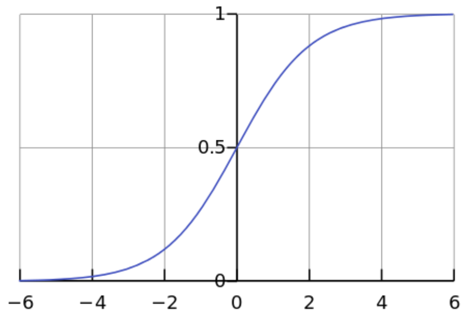
Perceptron

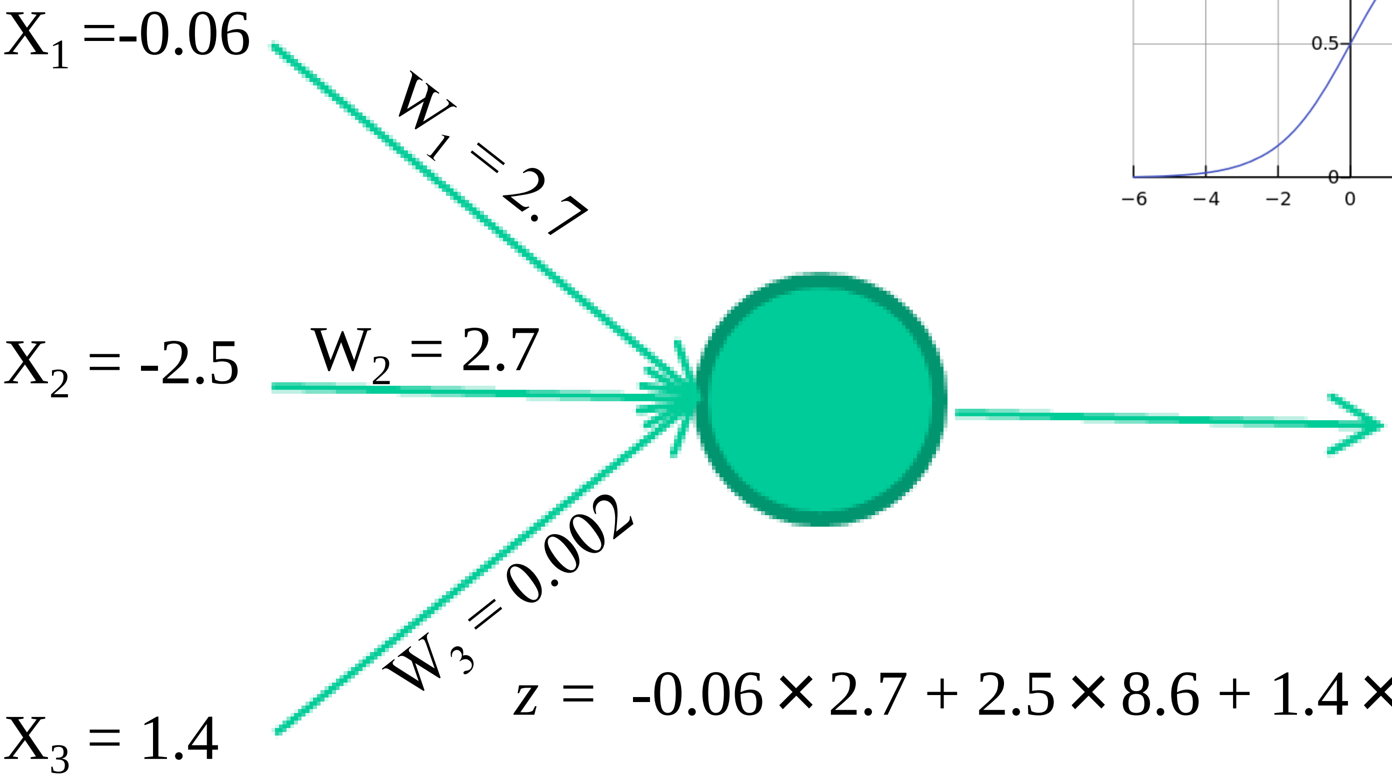


Activation function Weights Bias

$$y = \sigma\left(\sum_i w_i x_i + b\right) = \sigma\left(\sum_i \mathbf{w}^T \mathbf{x} + b\right)$$

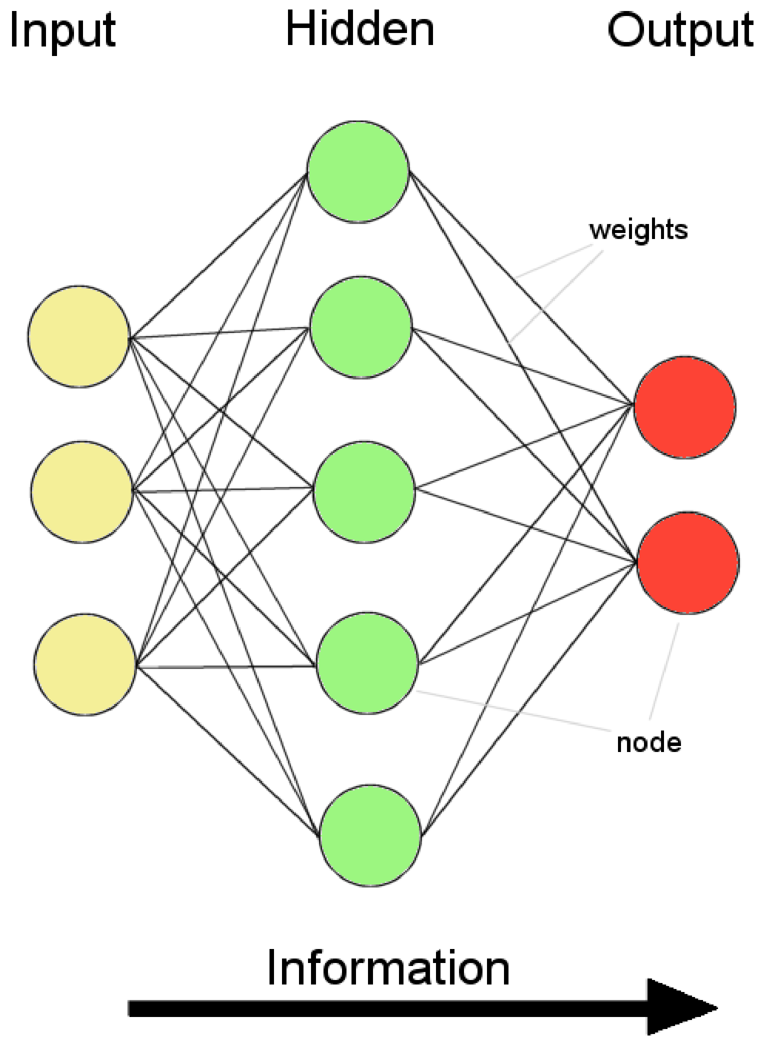
Perceptron



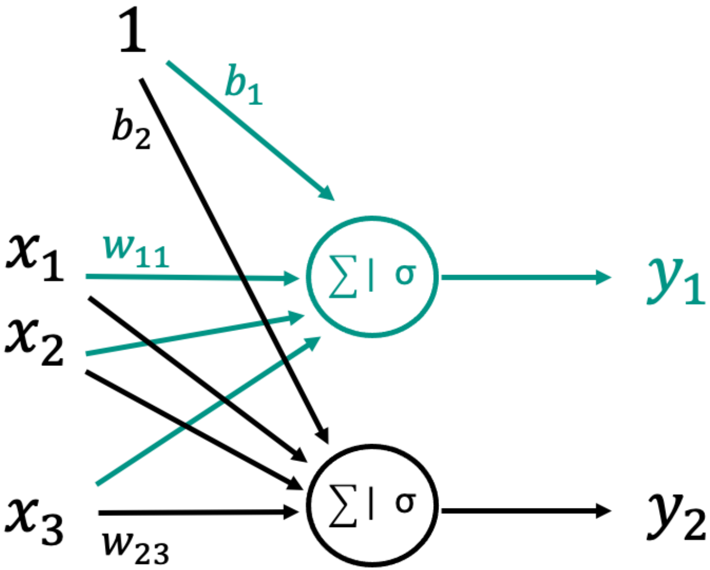


$$z = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$$

MLP



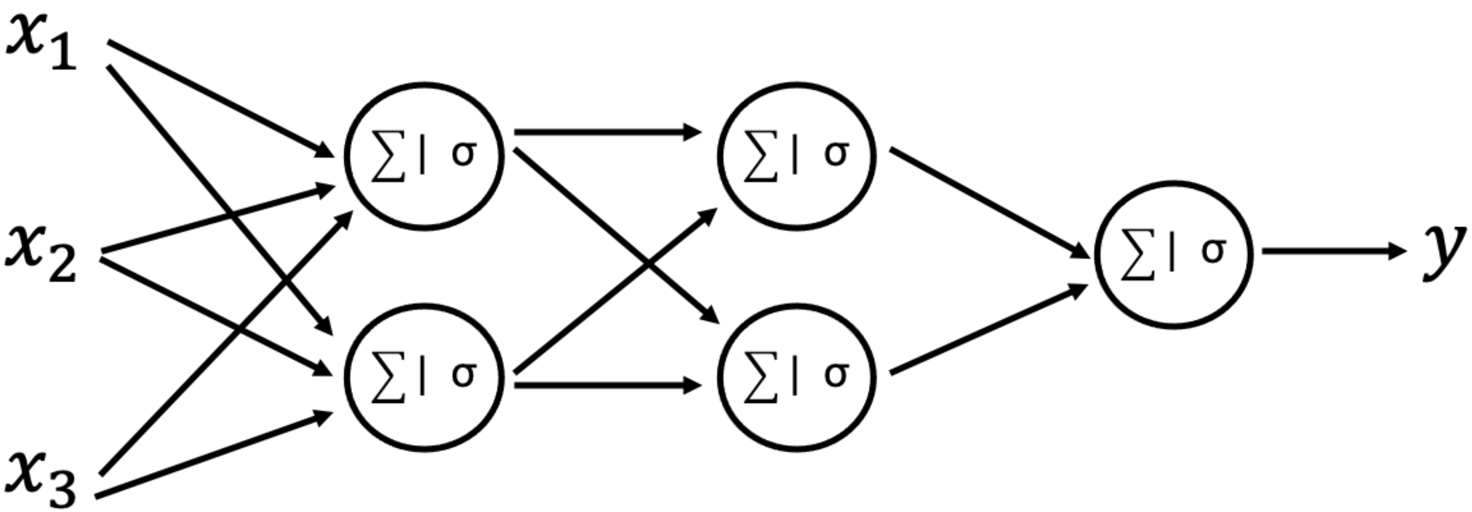
MLP



$$\mathbf{W} \in \mathbb{R}^{N_o \times N_i} \quad \mathbf{b} \in \mathbb{R}^{N_o}$$

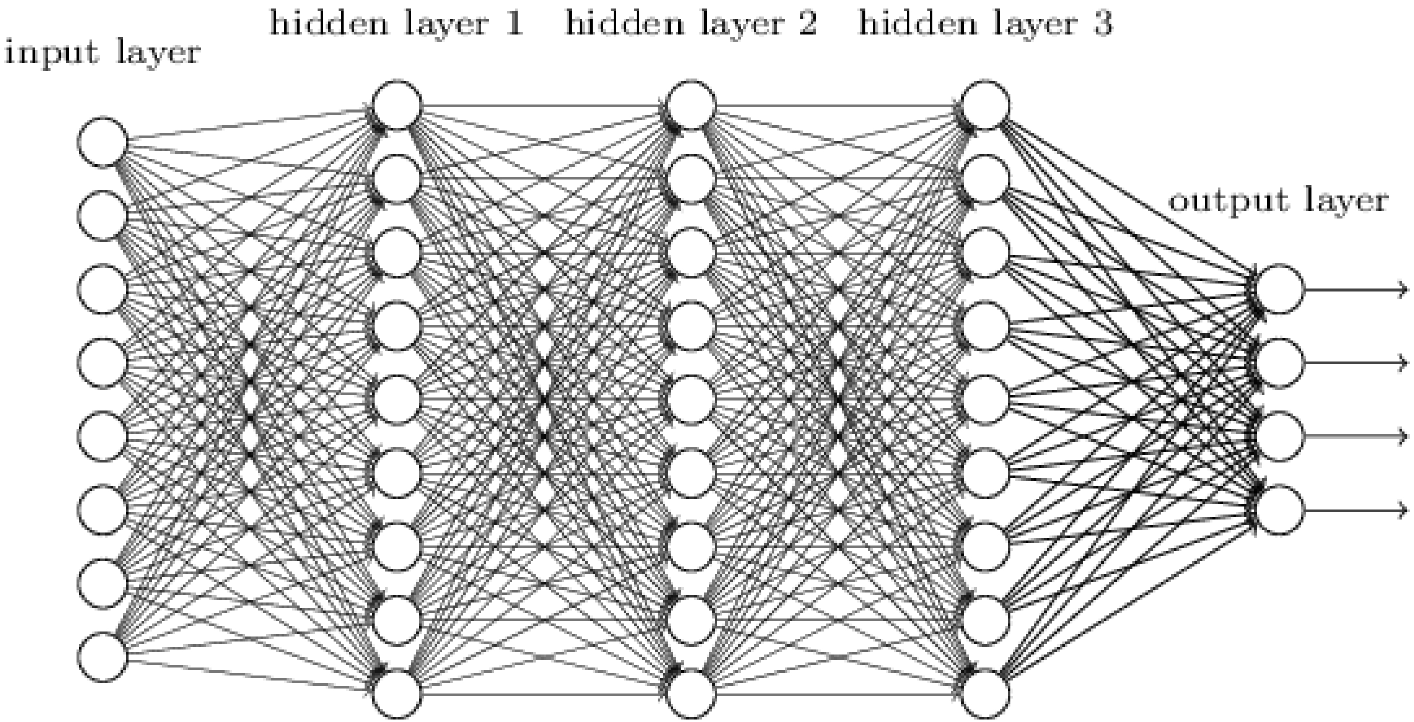
$$y_j = \sigma(\sum_i w_{ji} x_i + b), \quad \mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Deep Network (3-layers)



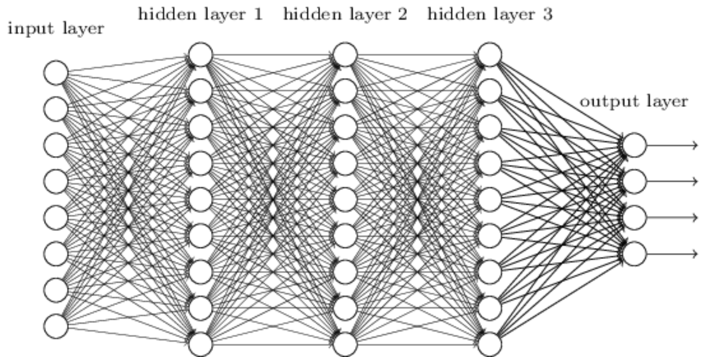
$\mathbf{x}$ $\mathbf{a}^{[1]}$ $\mathbf{a}^{[2]}$ $\mathbf{a}^{[3]} = \mathrm{y}$

Deep Network (3-layers)



Deep Network (3-layers)

- *Input layer*: first layer taking the input vector $\mathbf{x}$ as input and returning an intermediate representation $\mathbf{z}^{[1]}$;
- *Hidden layers*: second to penultimate layers taking as input the previous representation $\mathbf{z}^{[i-1]}$ and returning a new representation $\mathbf{z}^{[i]}$;
- *Ouput layer*: last layer producing the output of the network $\mathbf{y}$;
- *Depth*: number of hidden layers (plus output layer);
- *Width*: number of units in each hidden layer.



Activation Functions - motivation

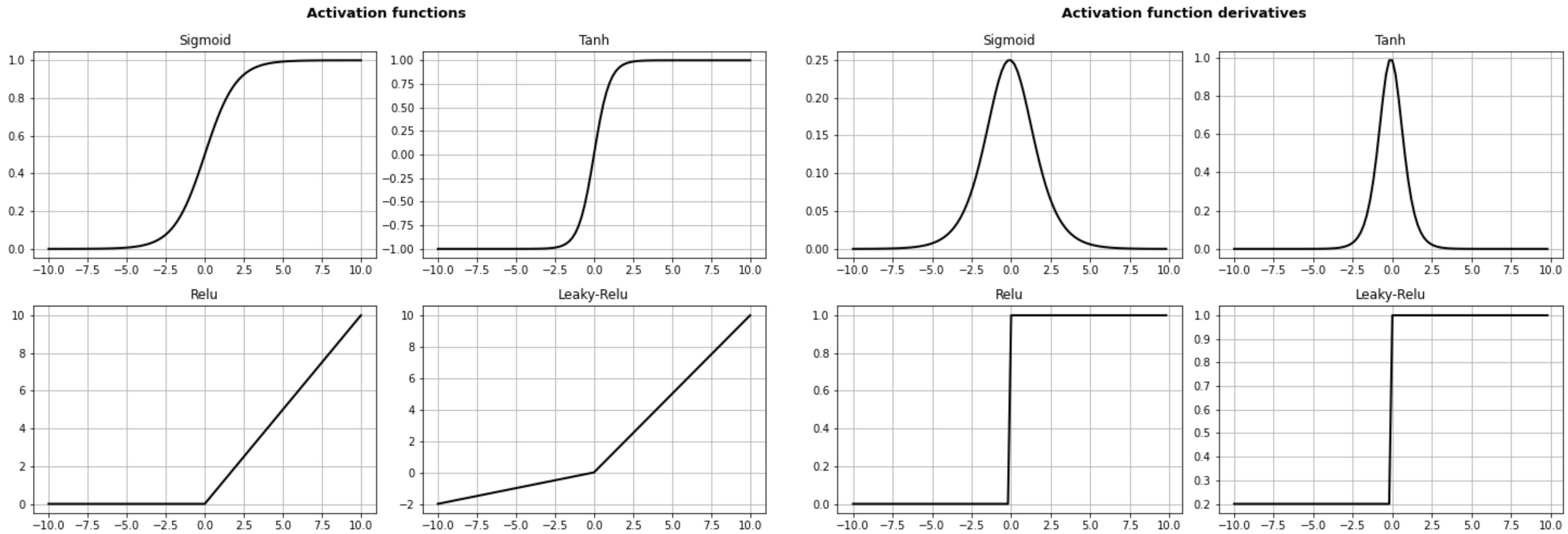
The need for nonlinearities comes from the recognition that:

$$\mathbf{y} = \sigma(\mathbf{W}^{[3]}\sigma(\mathbf{W}^{[2]}\sigma(\mathbf{W}^{[1]}\mathbf{x}))) = \mathbf{W}^{[3]}\mathbf{W}^{[2]}\mathbf{W}^{[1]}\mathbf{x} = \mathbf{W}\mathbf{x}$$

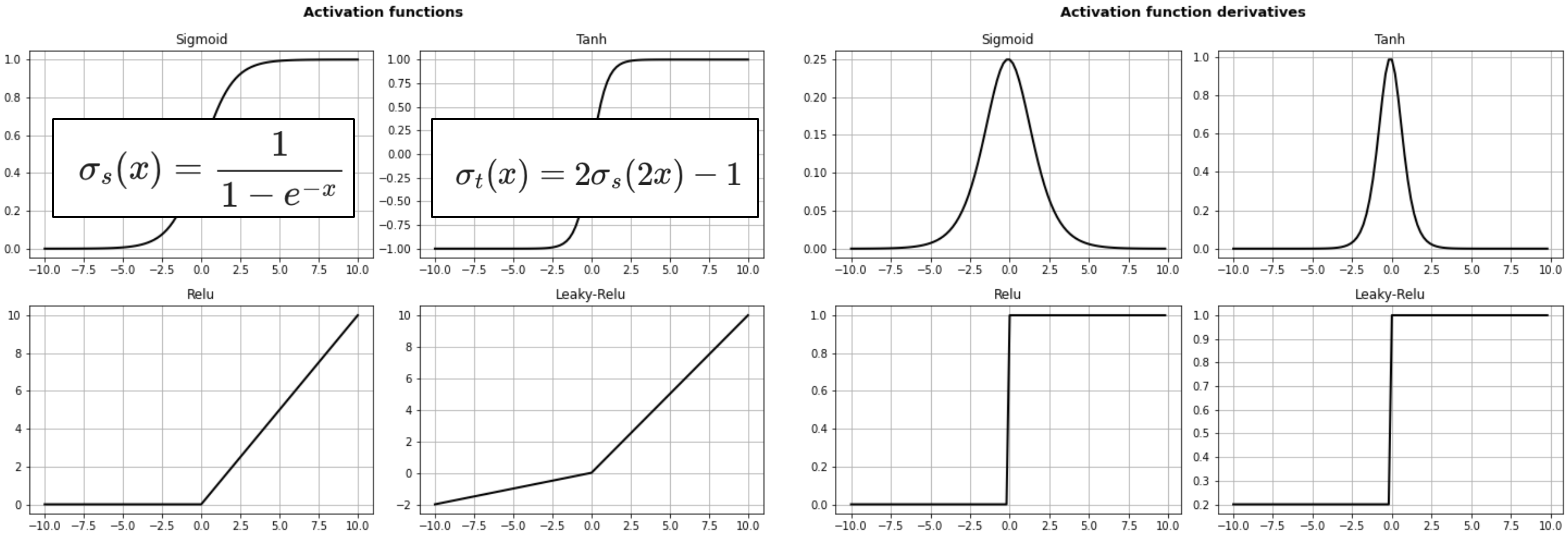
$$\sigma(\mathbf{x}) = \mathbf{I}\mathbf{x} = \mathbf{x},$$

One single linear model!

Activation Functions



Activation Functions

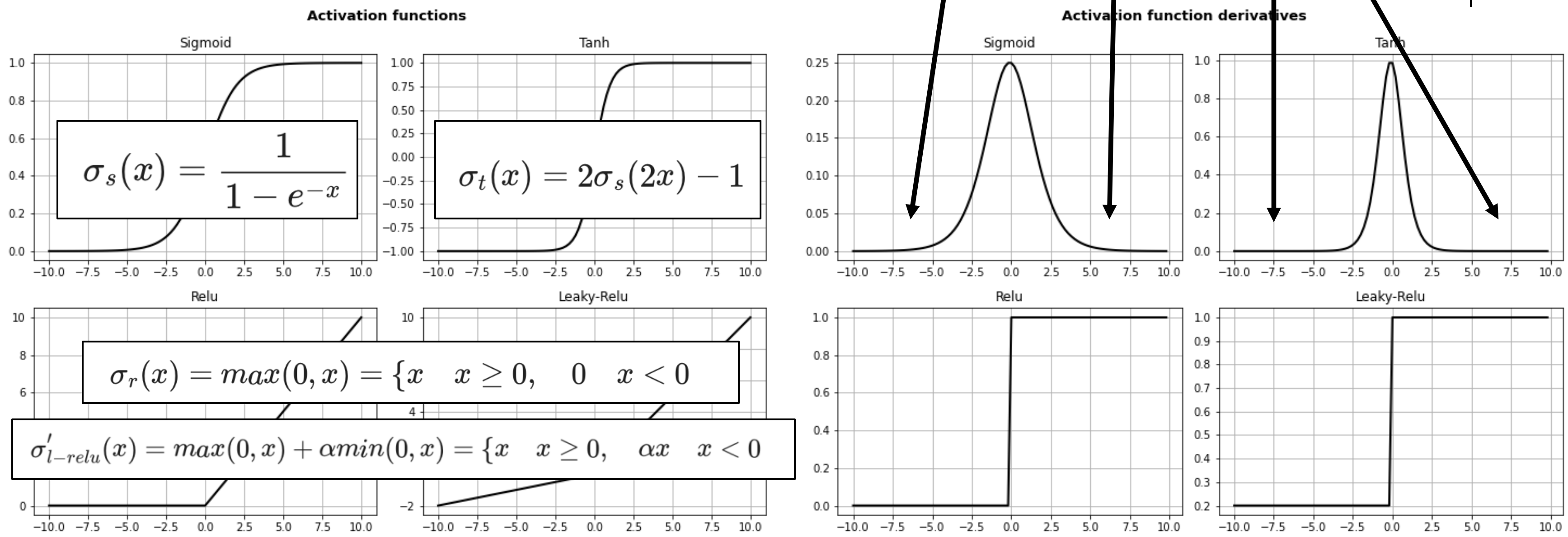


$$\sigma_s(x) = \frac{1}{1 - e^{-x}}$$

$$\sigma_t(x) = 2\sigma_s(2x) - 1$$

Activation Functions

Differentiable, but saturate away from 0



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

$$\sigma_t(x) = 2\sigma_s(2x) - 1$$

$$\sigma_r(x) = max(0, x) = \{x \quad x \geq 0, \quad 0 \quad x < 0$$

$$\sigma'_{l-relu}(x) = max(0, x) + \alpha min(0, x) = \{x \quad x \geq 0, \quad \alpha x \quad x < 0$$

Non-differentiable, but does not saturate

Loss Function (Regression)

The most commonly used cost function is:

$$J_\theta = \underset{\substack{\uparrow \\ \text{Mean squared error}}}{MSE}(\mathbf{y}_{train}, \hat{\mathbf{y}}_{train}) = \frac{1}{N_s}||\mathbf{y}_{train} - \hat{\mathbf{y}}_{train}||_2^2 = \frac{1}{N_s}\sum_i^{N_s}\underbrace{(y_{train}^{(i)} - \hat{y}_{train}^{(i)})^2}_{\mathcal{L}}$$

which is minimized as follows:

$$\hat{\theta} = min_\theta J_\theta \rightarrow \theta_{i+1} = \theta_i - \alpha \nabla J_\theta$$

Once the model is trained (= best parameters are learned) one can estimate the solution from a new input – INFERENCE

$$y_{test} = \tilde{\mathbf{x}}_{test}^T \hat{\theta}$$

Loss Function (Classification)

The loss function is called **binary cross-entropy**

$$\mathscr{L}(y^{(i)}_{train}, \hat{y}^{(i)}_{train}) = -(\underbrace{y^{(i)}_{train} log(\hat{y}^{(i)}_{train})}_{\text{for ‘true' labels (y=1)}} + \underbrace{(1 - y^{(i)}_{train}) log(1 - \hat{y}^{(i)}_{train})}_{\text{for ‘false' labels (y=0)}})$$

and the total cost function is:

$$J_\theta = \frac{1}{N_s} \sum_{i}^{N_s} \mathscr{L}(y^{(i)}_{train}, \hat{y}^{(i)}_{train}) \longrightarrow \hat{\theta} = min_\theta J_\theta \rightarrow \theta_{i+1} = \theta_i - \alpha \nabla J_\theta$$

Sigmoid Activation Function $\boxed{\sigma_s(x) = \frac{1}{1 - e^{-x}}}$

Loss Function (Classification)

Categorical cross-entropy loss in multi-class classification tasks;

$$CE = -\sum_{i=1}^{i=N} y_i \cdot log(\widehat{y_i})$$

$$CE = -[y_1 \cdot log(\widehat{y_1}) + y_2 \cdot log(\widehat{y_2}) + y_3 \cdot log(\widehat{y_3})]$$

Softmax Layer

Ultimately goal of a ML model (Supervised Learning)

A model is useful if it can ***perform well on new, previously unseen data***. This property of a model is also generally referred to as *generalization*.

<u>Training</u>

$\mathbf{x}^{(1)}$

$\mathbf{x}^{(2)}$

...

$\mathbf{x}^{(N_{train})}$

$\longrightarrow$ ML model $\longrightarrow$

$\hat{y}^{(1)} \longleftrightarrow y^{(1)}$

$\hat{y}^{(2)} \longleftrightarrow y^{(2)}$

...

$\hat{y}^{(N_{train})} \longleftrightarrow y^{(N_{train})}$

<u>Inference</u>

$\mathbf{x}^{(new)} \longrightarrow$ ML model $\longrightarrow \hat{y}^{(new)}$

ML data ‘structuring’ (Supervised Learning)

- Training dataset: $\{\mathbf{X}_{train}, \mathbf{Y}_{train}\}$, used to train the model (e.g., learn the free-parameters $\boldsymbol{\theta}$ of a NN);
- Validation dataset: $\{\mathbf{X}_{valid}, \mathbf{Y}_{valid}\}$, used to select the hyperparameters of the model;
- Testing dataset: $\{\mathbf{X}_{test}, \mathbf{Y}_{test}\}$, used only once a model is finalized (trained and optimized) to produce an *unbiased* estimate of model performance.

ML performance

Different measures for different datasets:

- **Training error** (or performance): overall error (or performance) computed over the training dataset;
- **Validation error** (or performance): overall error (or performance) computed over the validation dataset.
- **Test/Generalization error** (or performance): overall error (or performance) computed over the testing dataset.

Classification evaluation metrics (Confusion Matrix)

		True Labels	
		Positive (P)	Negative (N)
Predicted Labels	Positive	Correct / True Positive (TP)	Type 1 Error / False Positive (FP)
	Negative	Type 2 Error / False Negative (FN)	Correct / True Negative (TN)

Classification evaluation metrics

- Precision

$$Pr = \frac{TP}{TP+FP}$$

Appropriate when minimizing false positives is the focus..

- Recall:

$$Rc = \frac{TP}{TP+FN} = \frac{TP}{P}$$

Appropriate when minimizing false negatives is the focus..

- Accuracy:

$$Ac = \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{P+N}$$

Percentage of correct predictions over the total number of cases. Combines both error types (in the denominator), it is therefore a more global measure of the quality of the model.

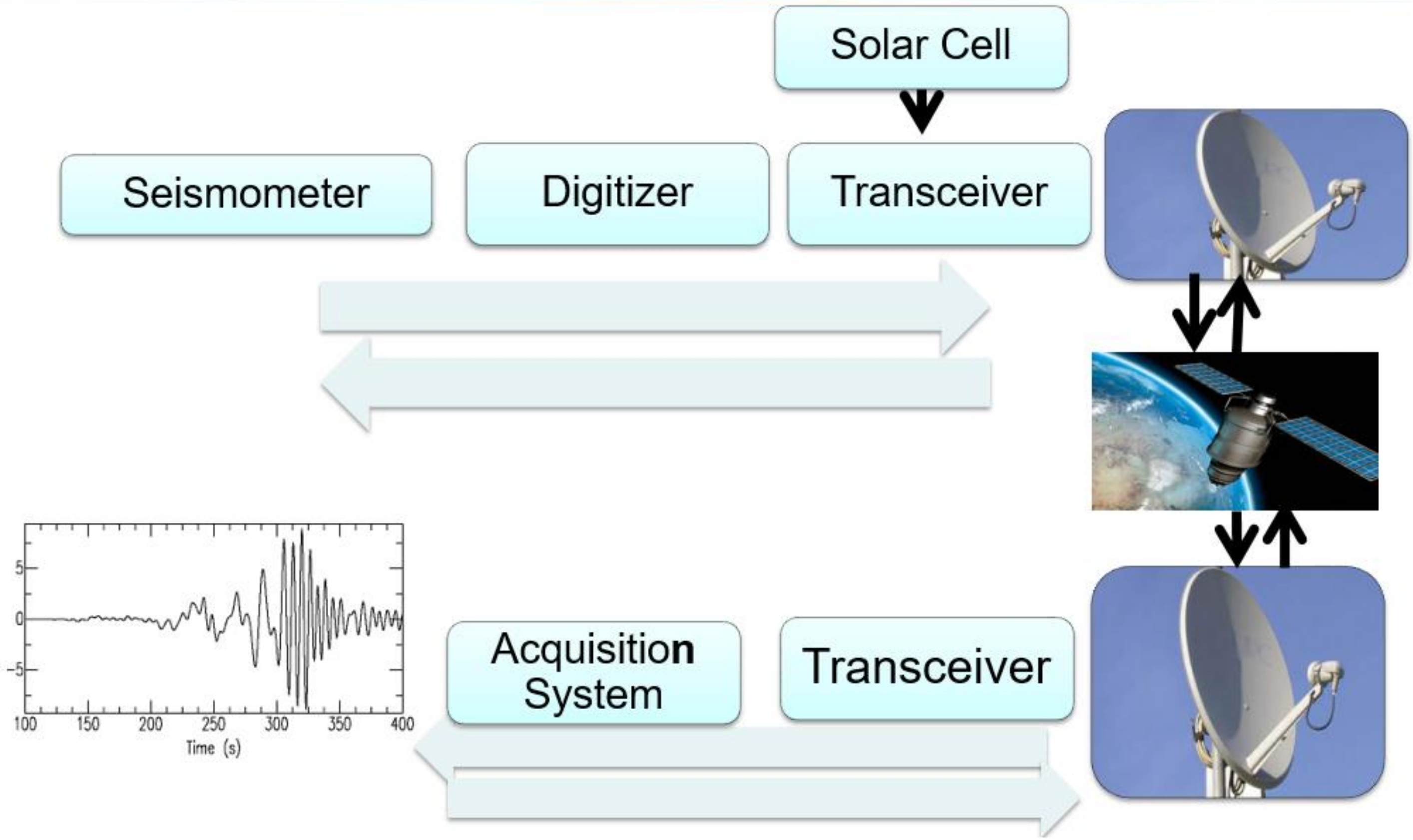
- F1-score:

$$2\frac{Pr \cdot Rc}{Pr+Rc}$$

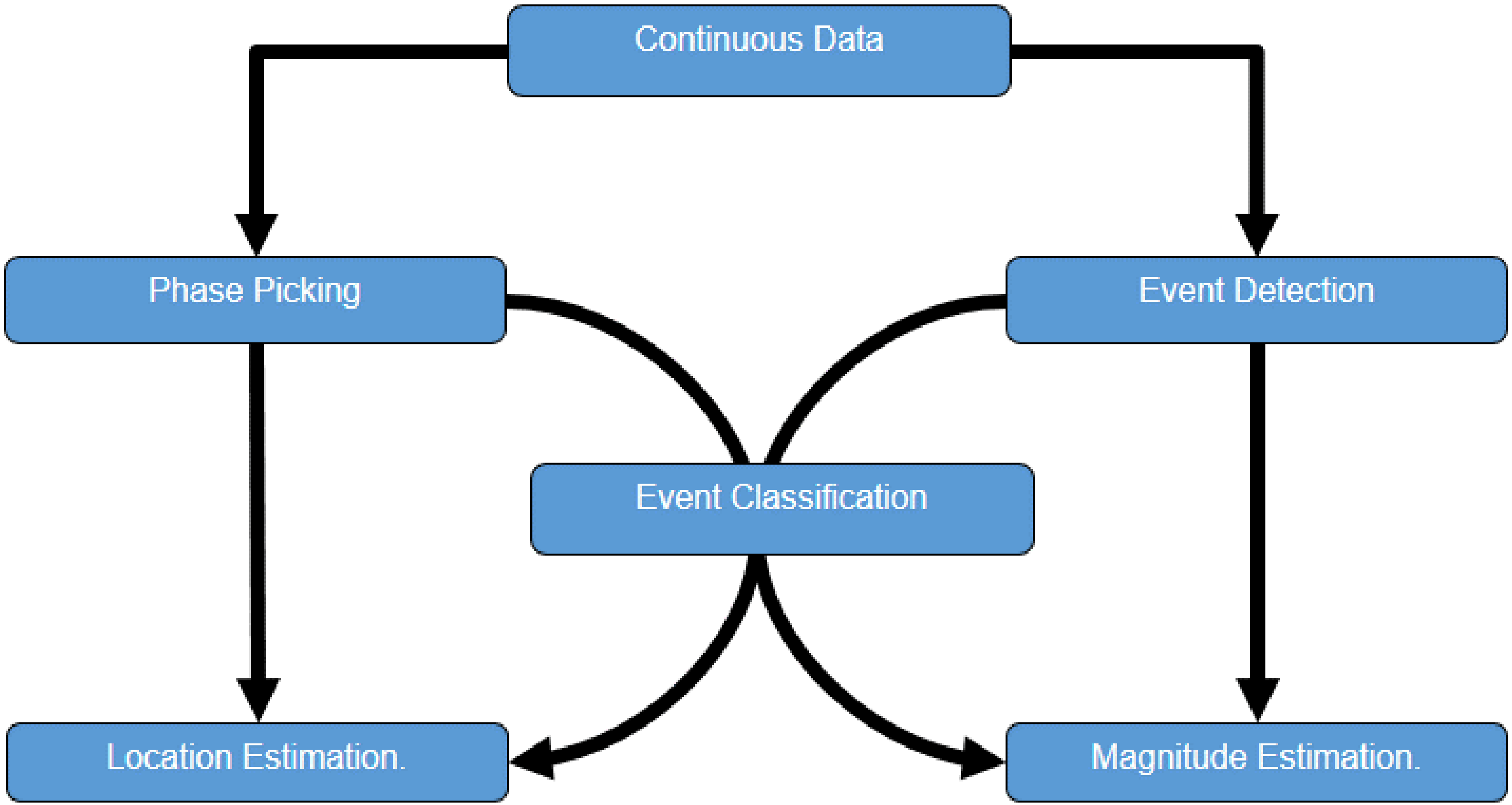
Combines precision and recall into a single measure

Application

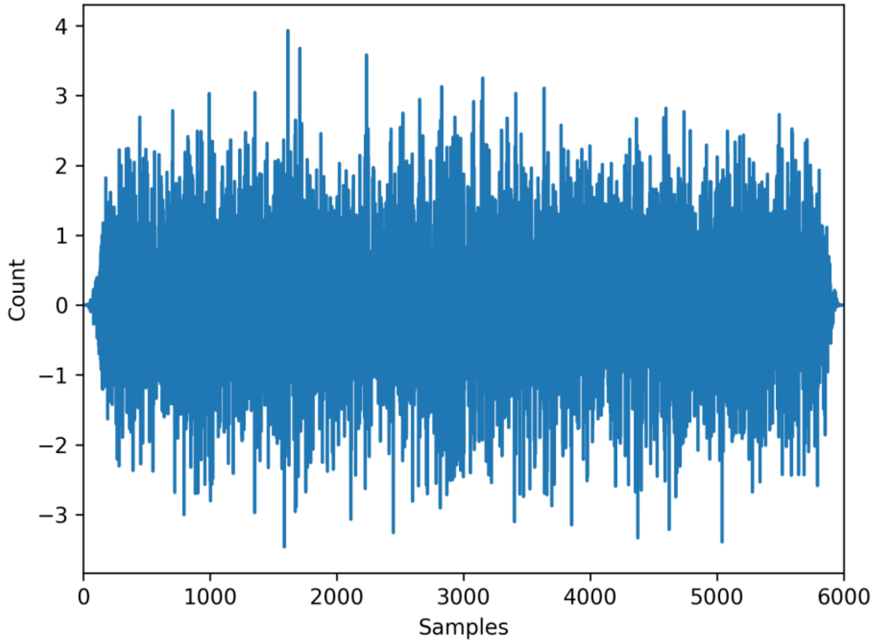
Earthquake Monitoring System

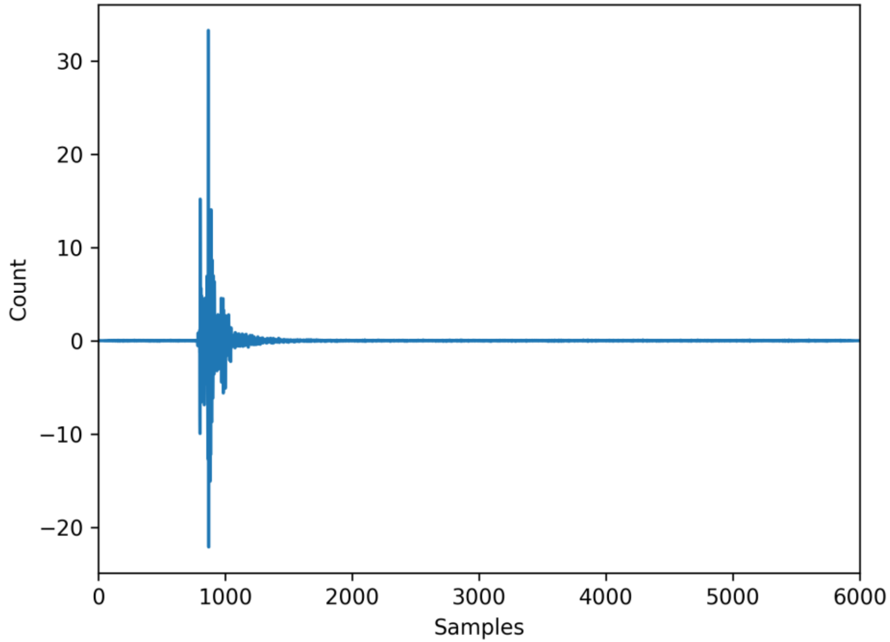


Earthquake Monitoring System



Seismic Event Classification






Thank You
30 17:50