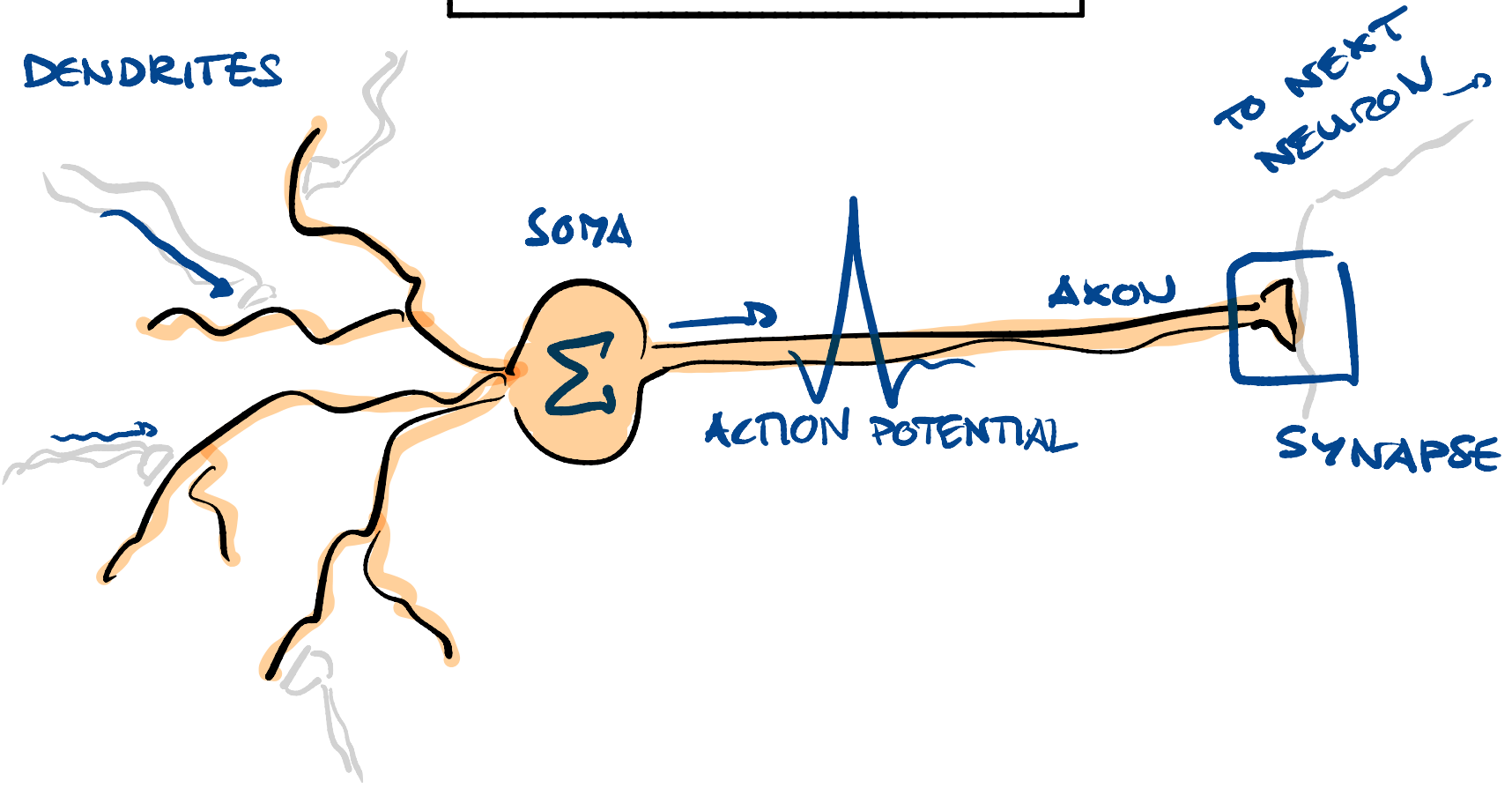
# Deep Learning for PDE

Applied Math Seminar

Dominique Zosso

9/17/2020

# INTEGRATE & FIRE



# THE PERCEPTRON

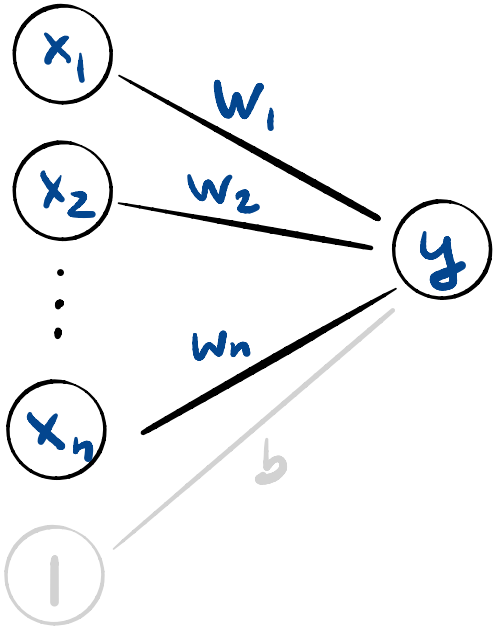
$$y : \mathbb{R}^n \longrightarrow \mathbb{R}$$

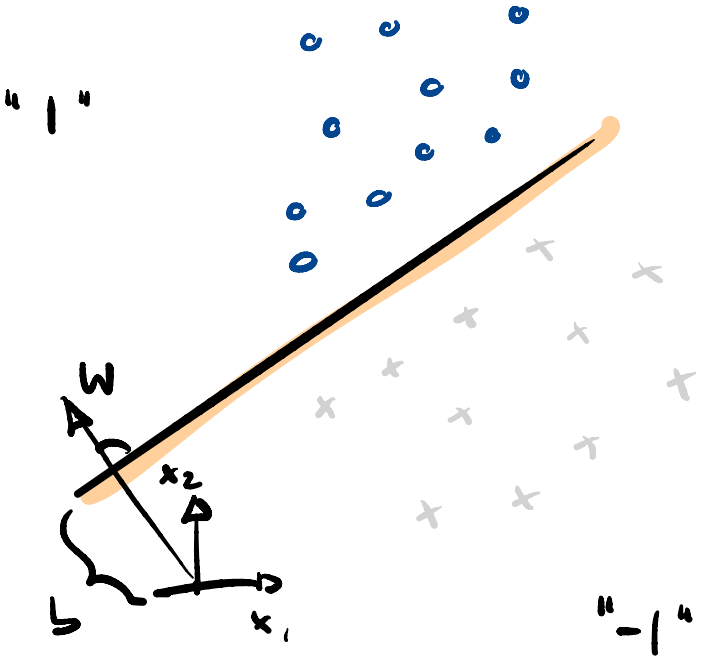
$$y(x) = f\left( w^T x + b \right)$$

$$f(\cdot) := \begin{cases} +1, & \cdot > 0 \\ -1, & \cdot < 0 \end{cases}$$

(OR: TANH, SIGMOID...)

$$y(x) = f(w^T x + b)$$

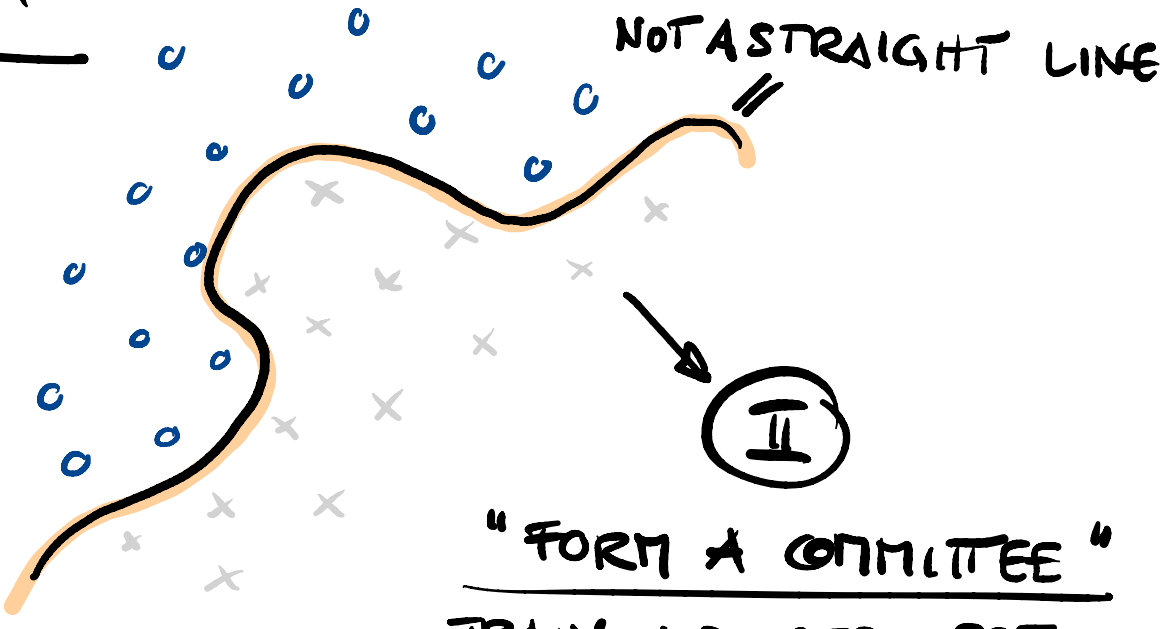



⇒ "WEAK LEARNER"

(PERFECT CLASSIFICATION ON LINEARLY SEPARABLE DATA)

"ACTUAL" DATA:




INTRODUCE:

$$\phi : \mathbb{R}^n \to \mathbb{R}^m$$

THEN:

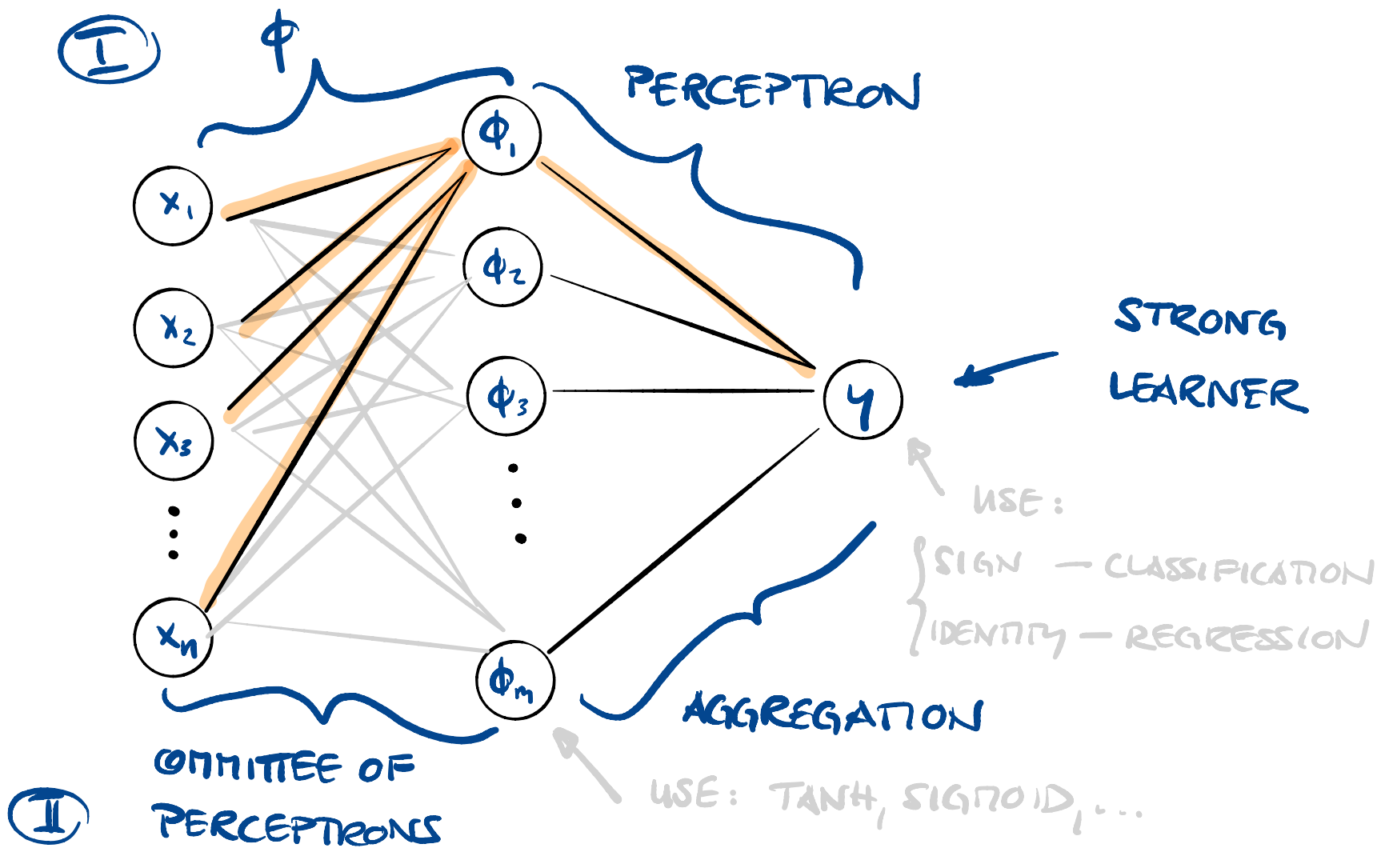
$$y(x) = f(w^T \phi(x) + b)$$

⇒ KERNEL METHODS

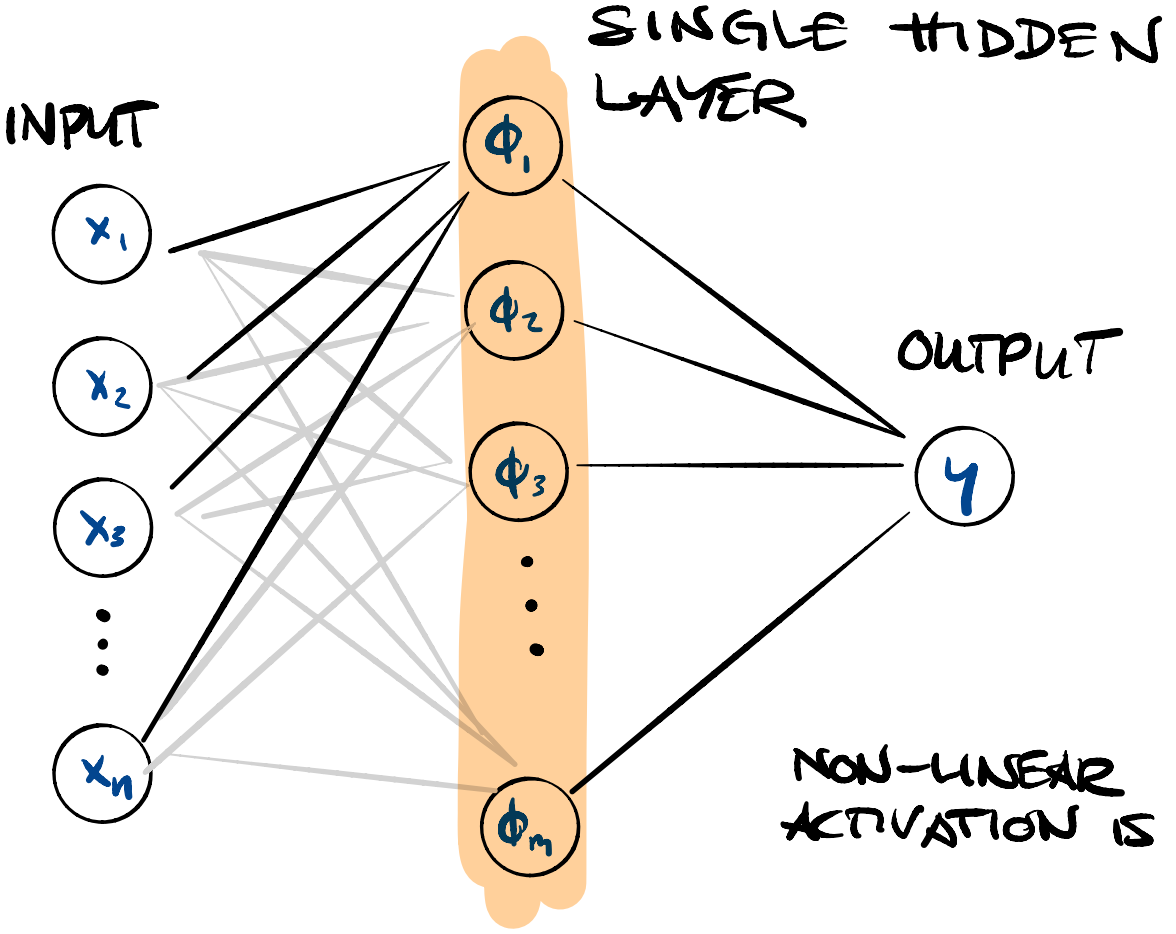
"FORM A COMMITTEE"

TRAIN A DIVERSE SET OF WEAK LEARNERS AND AGGREGATE INTO A STRONG LEARNER

⇒ ENSEMBLE METHODS


I
$\phi$
PERCEPTRON
$\phi_1$
$\phi_2$
$\phi_3$
$\phi_m$
$x_1$
$x_2$
$x_3$
$x_n$
$y$
STRONG LEARNER
USE:
SIGN — CLASSIFICATION
IDENTITY — REGRESSION
AGGREGATION
USE: TANH, SIGMOID, ...
COMMITTEE OF PERCEPTRONS
II

# FEED FORWARD ARCHITECTURE:




NON-LINEAR ACTIVATION IS KEY!

# UNIVERSAL REPRESENTATION:

ARBITRARY WIDTH

- CYBENKO (1989): $f = \sigma$ (SIGMOID: $\frac{1}{1-e^{-\cdot}}$)

  $\forall g \in C([0,1]^n)$, $\varepsilon > 0$ : $\exists m, w$ : $\sup(g(x) - y(x)) < \varepsilon$

- HORNIK (1991): MORE GENERIC ACTIVATIONS
- LESHNO + (1993), PINKUS (1999): $\Leftrightarrow$ $f$ NON-POLYNOMIAL

DEPTH

- VARIOUS LIMITED WIDTHS: $m > n$ ; $m = n+3$ ; $m \leq n+4$

  BUT ARBITRARY # OF LAYERS

## So far:

$y(x, w)$ is a non-linear function in $x$.

① **Hyperparameters:**
- network architecture
- activation functions
- (regularizer)

② **Parameters:**
- edge weights $w$

$\Rightarrow$ Fix ①, learn ②, validate

**TRAINING DATA:**

$$\begin{cases} X \in \mathbb{R}^{n \times N} = [\vec{x}_1 \dots \vec{x}_N] \\ t \in \mathbb{R}^N \quad \text{(TARGET VALUES)} \end{cases}$$

WANT: $y(\vec{x}_i) = t_i$

$$\Rightarrow \min_w \frac{1}{2} \sum_{i=1}^{N} \left( y(\vec{x}_i) - t_i \right)^2 \quad \text{(LOSS)}$$

**ALGORITHM:** STOCHASTIC GRADIENT DESCENT

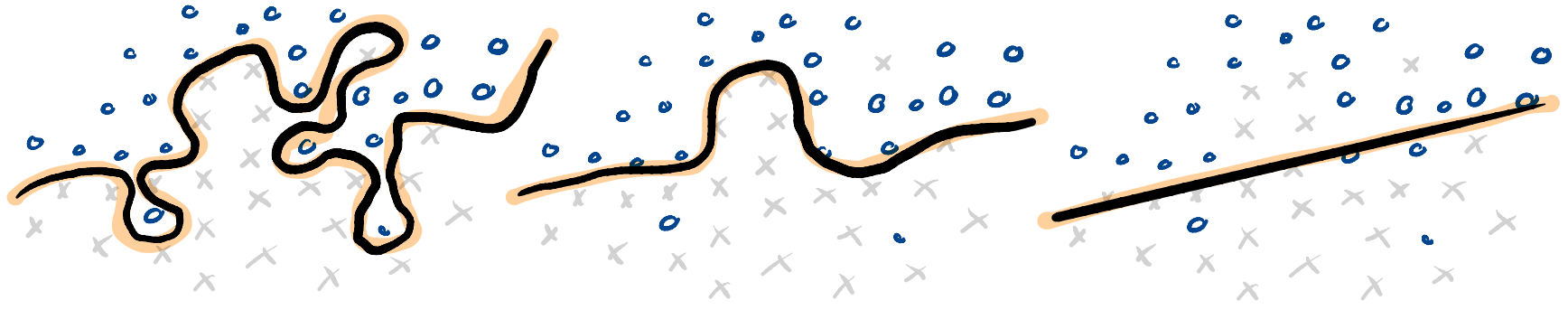
(OPERATE ON MINI-BATCHES OF DATA)

**NOTE:**

- CHAIN-RULE → BACKPROPAGATION
- NEED LOTS OF TRAINING DATA

# A WORD ON OVERFITTING:

- WHY REGULARIZE ?
- WHY VALIDATE ?



OVERFIT
(MODEL NOISE)

UNDERFIT
(OVER SIMPLIFY)

$$\rightarrow \min_{w} \frac{1}{2} \sum_{i=1}^{N} \left( y(\vec{x}_i) - t_i \right)^2 + \frac{\alpha}{2} \|w\|^2$$

# MATLAB*

* IF WEBEX / IPAD / SAFARI / MATLAB WILL LET ME

THUS FAR: ARTIFICIAL NEURAL NETWORKS
= ALL-PURPOSE BLACK BOXES.

TWO DATA-TYPES STAND OUT:

**TIME SERIES:**

TRADITIONAL TOOLS:

- AUTO-REGRESSION
- DYNAMICAL SYSTEMS

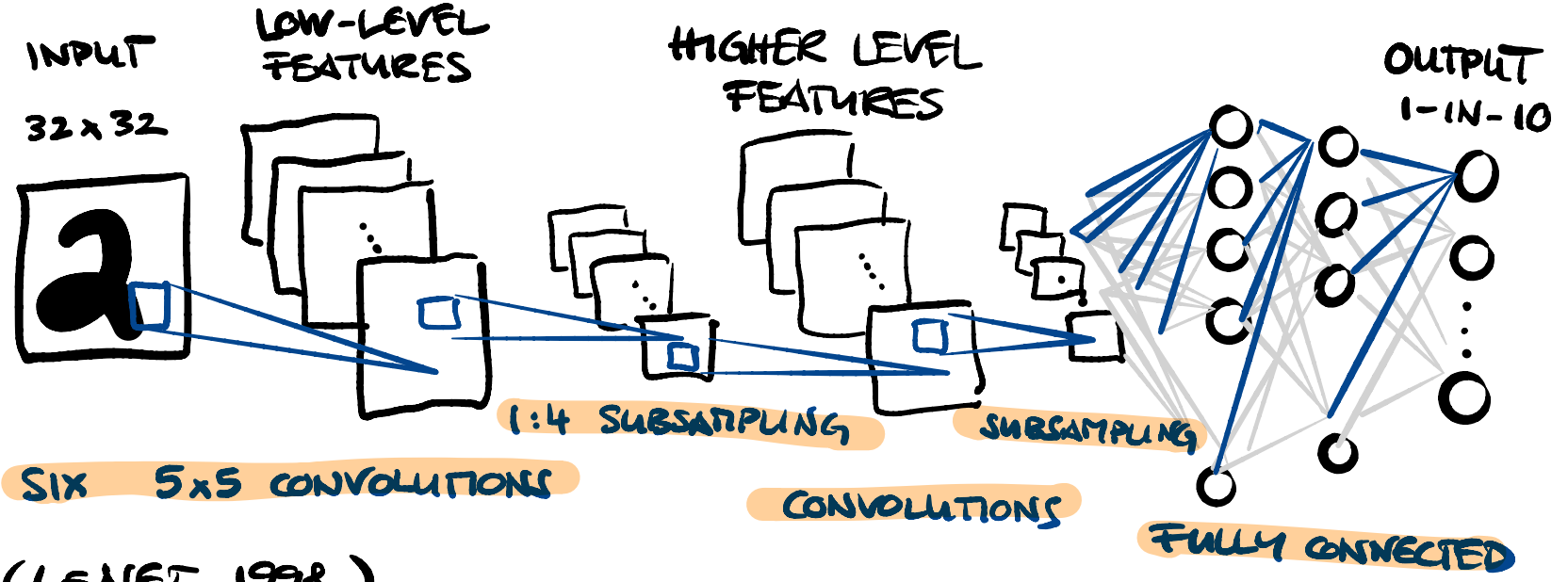
**IMAGES:**

- CONVOLUTIONS

⇒ RECURRENT NETWORKS (INTRODUCE LOOPS, DELAYS)

# CONVOLUTIONAL NEURAL NETWORK:

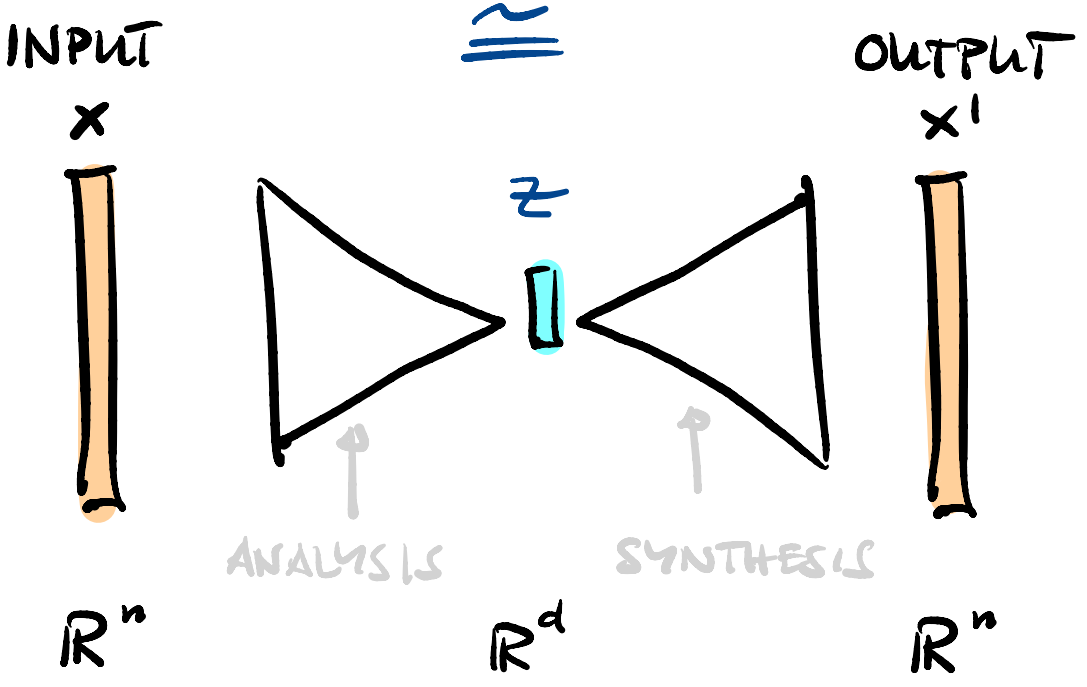
- FUNDAMENTAL IDEA: IMAGE FEATURES ARE TRANSLATION-INVARIANT — SO SHOULD BE WEIGHTS.




(LENET, 1998)

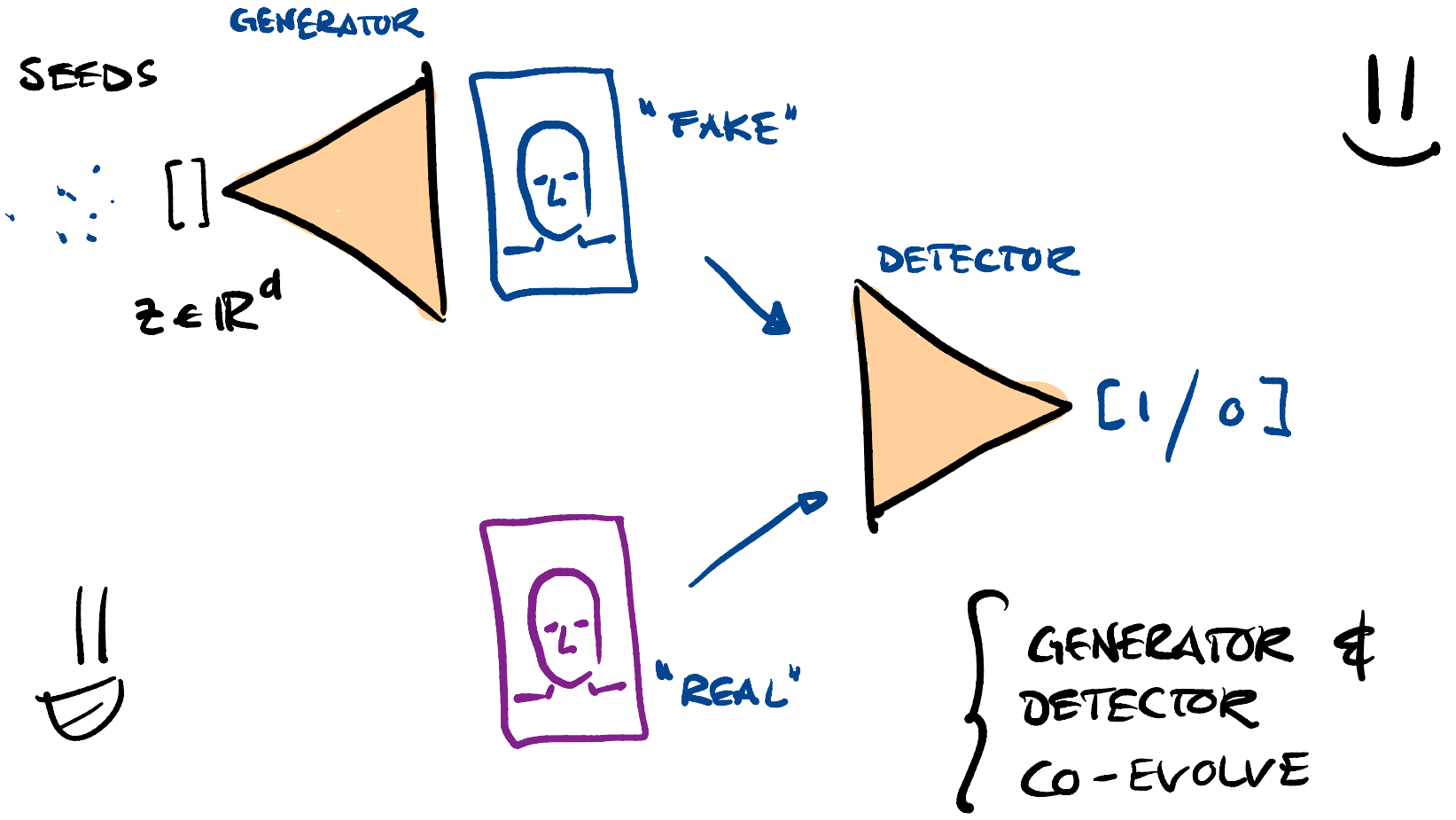
# AUTOENCODER

GOAL: LEARN OPTIMAL LOW-DIM REPRESENTATION.




SELF-SUPERVISED : TRAINING INPUT = TRAINING OUTPUT

# GENERATIVE ADVERSARIAL NETWORK: (GAN)

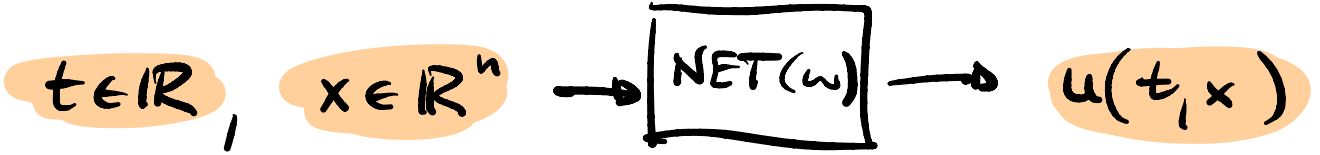


# DEEP GALERKIN METHOD

(SIRIGNANO & SPILIOPOULOS, J. COMP. PHYS. 375, 2018)

⇒ REDUCED FORM SOLUTION TO PDE

PARAMETRIZED ~~AS~~ NEURAL NET:



# MOTIVATION:

- FINITE DIFFERENCE METHODS STRUGGLE WITH "HIGH" DIMENSIONS.

- GALERKIN: LINEAR COMBINATIONS OF BASIS FUNCTIONS, INSTEAD

- HERE: COMBINE MESH-FREE GALERKIN W/ THE POWER OF "MACHINE LEARNING"

NETWORK "$u(x, w)$" IS TRAINED TO SATISFY

- DIFFERENTIAL OPERATOR
- INITIAL CONDITION
- BOUNDARY CONDITIONS

USING STOCHASTIC GRADIENT DESCENT

AT RANDOMLY * SAMPLED SPATIAL POINTS

## EXAMPLE: LINEAR WAVE EQUATION

$$\frac{\partial u}{\partial t} + c \cdot \frac{\partial u}{\partial x} = 0, \qquad (t,x) \in [0,1]^2$$

I.C. $\quad u(0,x) = f(x), \qquad x \in [0,1]$

B.C. $\quad u(t,0) = u(t,1), \qquad t \in [0,1]$

**STEP 1**: TURN PDE INTO A **LOSS** ($L^2$-ERROR)

$$J[y] := \left\| \frac{\partial y}{\partial t} + c \cdot \frac{\partial y}{\partial x} \right\|^2_{2,\,[0,1]^2}$$

$$+ \left\| y(0,\cdot) - f(\cdot) \right\|^2_{2,\,[0,1]}$$

$$+ \left\| y(\cdot,0) - y(\cdot,1) \right\|^2_{2,\,[0,1]}$$

**STEP 2**: FIND $\min_{y} J[y]$ (IDEALLY: $J[y^*] = 0$)

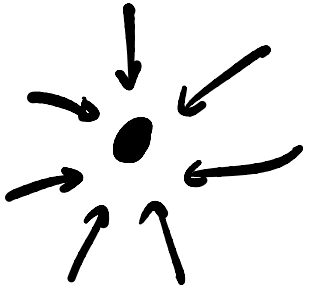
**THM (S & S):**

LET $y^n$ BE A SINGLE-HIDDEN-LAYER NN WITH $n$ NODES WHICH MINIMIZES $\mathcal{J}[y]$.

THEN (UNDER CERTAIN CONDITIONS)

$$\exists \{y^n\} \text{ S.T. } \begin{cases} \mathcal{J}(y^n) \to 0 \\ y^n \to u \end{cases} \text{ AS } n \to \infty$$

(STRONGLY IN $L^p([0,1]^2)$, $p < 2$)



: DIRECTLY MINIMIZING $\mathcal{J}[y]$ IS NOT COMPUTATIONALLY TRACTABLE.

INSTEAD: MESH-FREE APPROACH BY "ESTIMATING" ERROR TERMS FROM POINTWISE SAMPLES.

## ALGORITHM: TRAIN NETWORK $y(t, x, \theta_n)$

- GENERATE **RANDOM POINTS**

$$\begin{cases} (t_n, x_n) \text{ FROM } [0,1]^2 \\ z_n \text{ FROM } [0,1] \quad \text{(SPACE)} \\ w_n \text{ FROM } [0,1] \quad \text{(TIME)} \end{cases}$$

- **COMPUTE:** $$J(\theta_n) := \left[\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right) y(t_n, x_n, \theta_n)\right]^2$$
$$+ \left[y(0, z_n, \theta_n) - f(z_n)\right]^2 + \left[y(w_n, 0, \theta_n) - y(w_n, 1, \theta_n)\right]^2$$

- **GRADIENT DESCENT:** $\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta J(\theta_n)$

SECRETS:

- USE "MINI BATCHES" INSTEAD OF INDIVIDUAL POINTS
- $\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right) y$ CAN BE COMPUTED FROM THE NETWORK!

  (IN FACT, MATLAB WILL DO FIRST ORDER DERIVATIVES FOR US; SECOND ORDER = MONTE CARLO APPROX.)

# MATLAB*

* SAME

↳ 1 IN 6 RUNS "LOOKS" GOOD?

- "DIFFICULT" PDE
- LOSS FUNCTION HAS DIFICULT LANDSCAPE (LOCAL STATIONARY PTS.)
- GRADIENT-BASED OPTIMIZATION...
- BALANCING OF PARAMETERS (#POINTS)
- NETWORK ARCHITECTURE

## RECAP: DEEP LEARNING

- BIOLOGICALLY MOTIVATED (PERCEPTRON, MLP)
- GENERALIZES WINNING CONCEPTS (KERNELS, ENSEMBLE METHODS)

- MORE POWERFUL ARCHITECTURES
- REQUIRES LOTS OF TRAINING (DATA?)

- SELF-SUPERVISED TRAINING FOR PDE
- MESH-FREE; DERIVATIVES!
- THE DEVIL IS IN THE DETAILS

CHECK GITHUB.COM/DZOSSO/DEEPLEARNINGPDE

FOR:

- [ ] PAPER
- [ ] SLIDES
- [ ] MATLAB CODE