FeedForward Networks

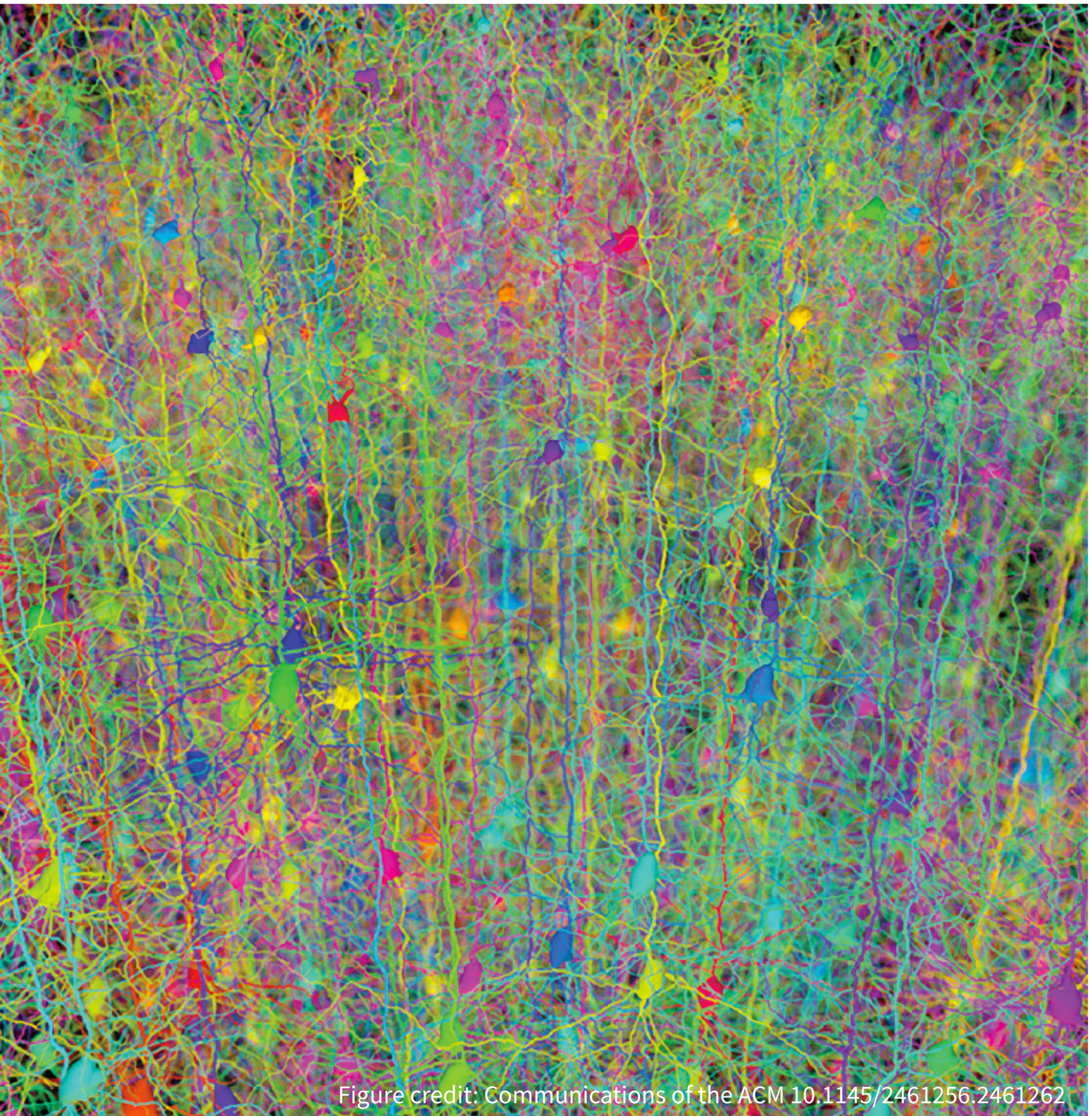


Figure credit: Communications of the ACM 10.1145/2461256.2461262

GRAHAM TAYLOR

VECTOR INSTITUTE

SCHOOL OF ENGINEERING
UNIVERSITY OF GUELPH

CANADIAN INSTITUTE
FOR ADVANCED RESEARCH

UNIVERSITY of GUELPH
CHANGING LIVES
IMPROVING LIFE

CIFAR
CANADIAN
INSTITUTE
FOR
ADVANCED
RESEARCH

Artificial Neurons

- Neuron pre-activation (also called input activation)

$$a(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b = \sum_i w_i x_i + b$$

- Neuron activation (also called output activation)

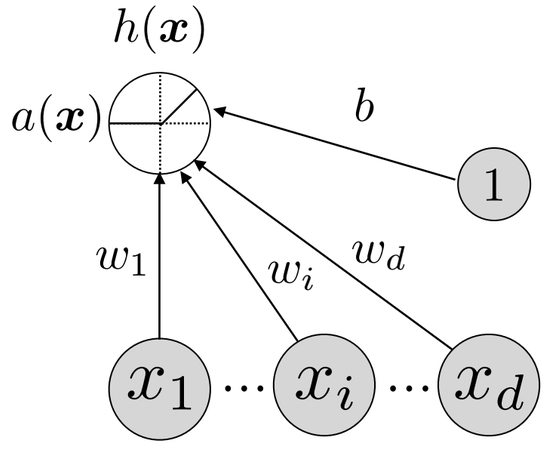
$$h(\boldsymbol{x}) = g\left(a\left(\boldsymbol{x}\right)\right) = g\left(\sum_i w_i x_i + b\right)$$

- <u>Parameters:</u>

  - $\boldsymbol{w}$ are the connection weights

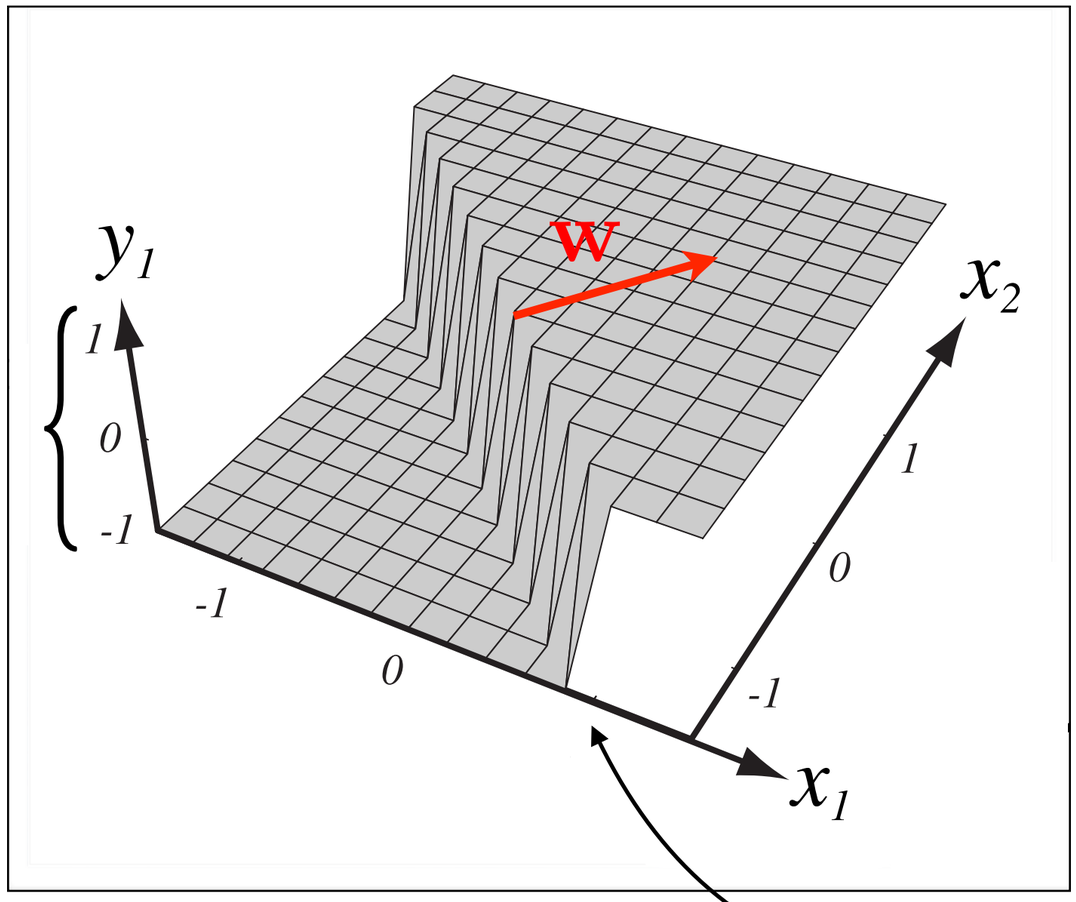
  - b is the neuron "bias"

- $g(\cdot)$ is called the activation function



Artificial Neurons - Geometry

range determined by $g(\cdot)$

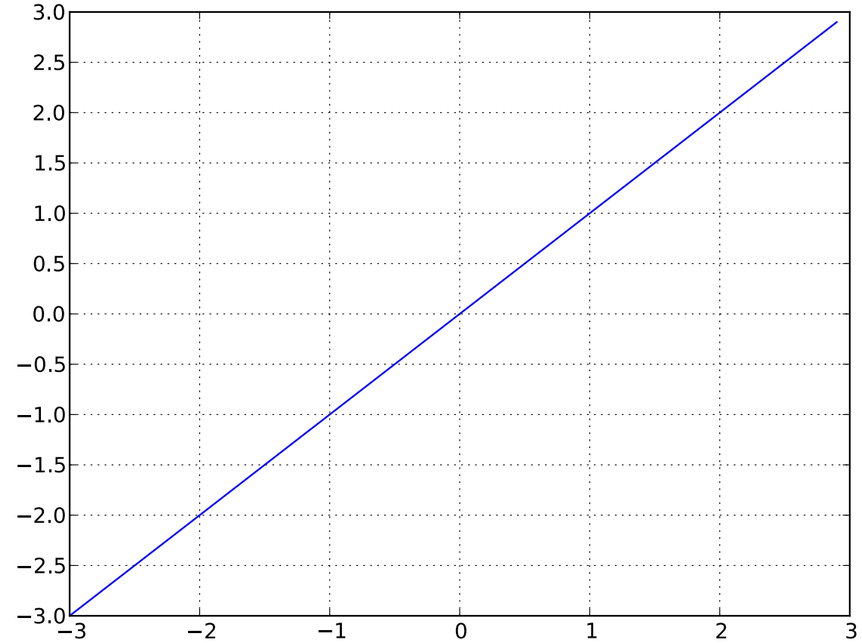


bias b changes the position of the riff

via Hugo Larochelle (reproduced from Pascal Vincent's slides)

Linear Activation Function

- Does not “squash” the input
- Not very interesting: no “warping”
- Typically used for outputs

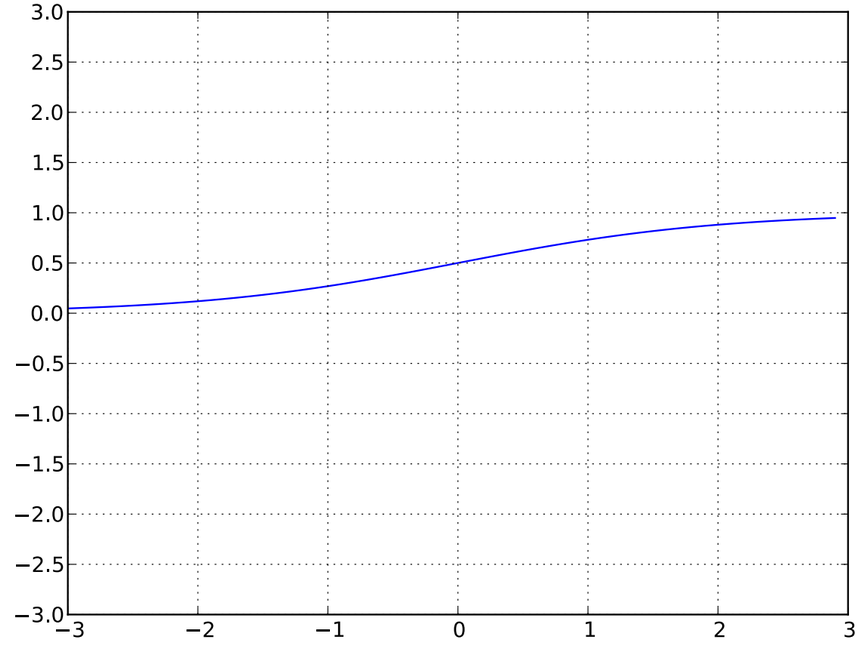


$$g(a) = a$$

Reproduced from Hugo Larochelle’s slides

Sigmoid Activation Function

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

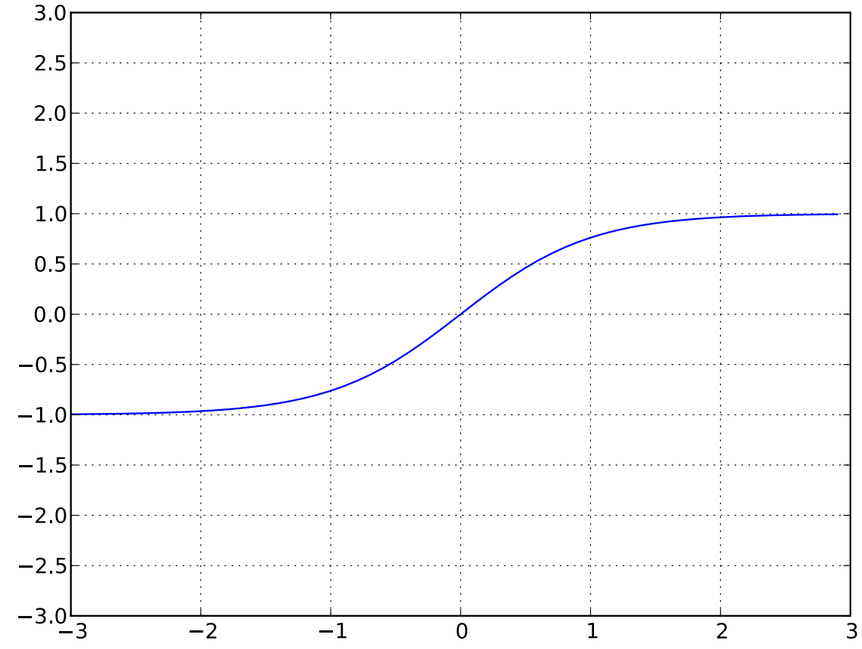


$$g(a) = \sigma(a) = \frac{1}{1 + \exp(-a)}$$

Reproduced from Hugo Larochelle's slides

Hyperbolic Tangent "tanh" Activation Function

- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

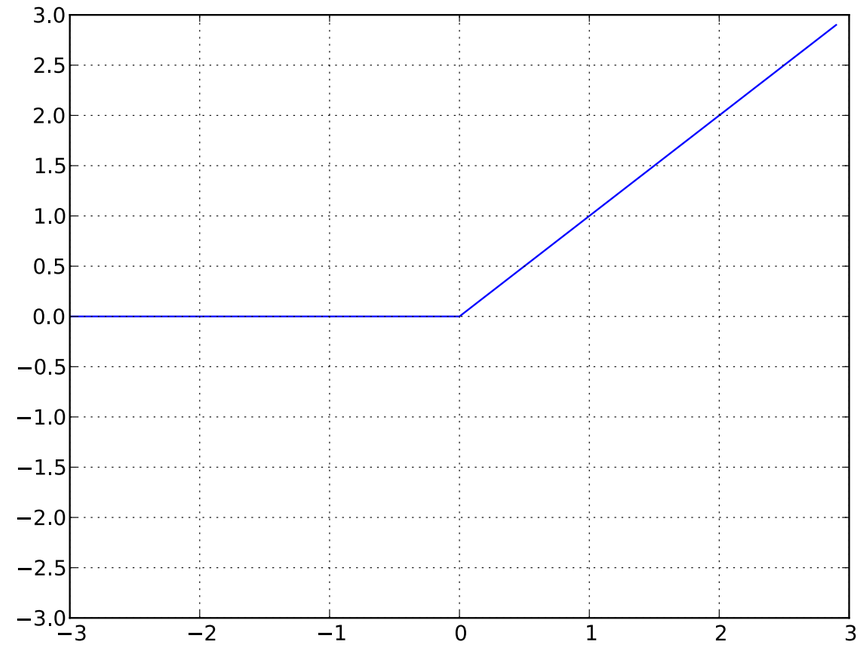


$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

Reproduced from Hugo Larochelle's slides

Rectified Linear (ReLU) Activation Function

- Bounded below by 0
  (always non-negative)
- Not upper bounded
- Strictly increasing
- Leads to neurons
  with sparse activities
- Dominant activation function
  in modern neural networks



$$g(a) = \text{relu}(a) = \max(0, a)$$

Reproduced from Hugo Larochelle's slides

ReLU Alternatives

- A more generic form of the ReLU can be formulated as follows:

$$g(a_i) = \begin{cases} a_i, & \text{if } a_i > 0 \\ p_i a_i, & \text{if } a_i \leq 0 \end{cases}$$

- This leads to several variations of standard ReLU:
- ReLU: obtained when $p_i = 0$
- PReLU: Parametric ReLU — obtained when p_i is a learnable parameter. The resultant activation function is:

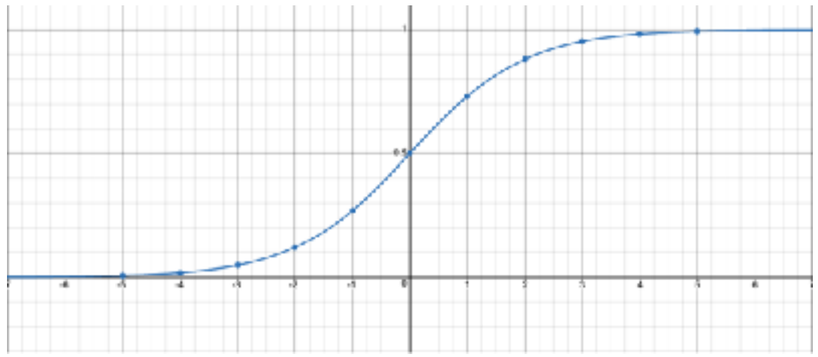
$$g(a_i) = \max(0, a_i) + p_i \min(0, a_i)$$

- LReLU: Leaky ReLU — obtained when p_i is a small and fixed value, e.g. 0.01. The resultant activation function is:

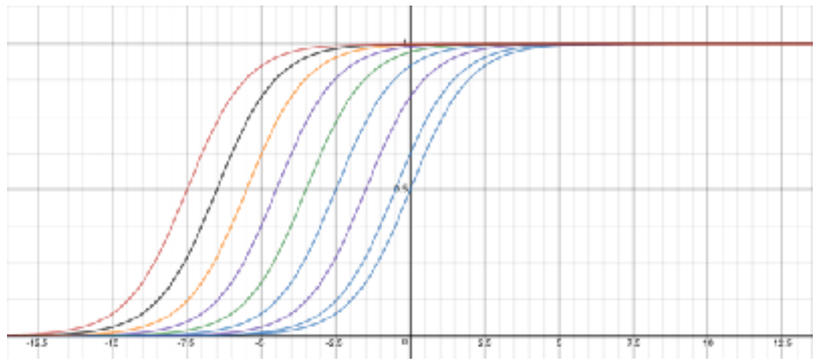
$$g(a_i) = \max(0, a_i) + 0.01 \min(0, a_i)$$

- RReLU: Randomized Leaky ReLU — obtained when p_i is a random number sampled from a uniform distribution.

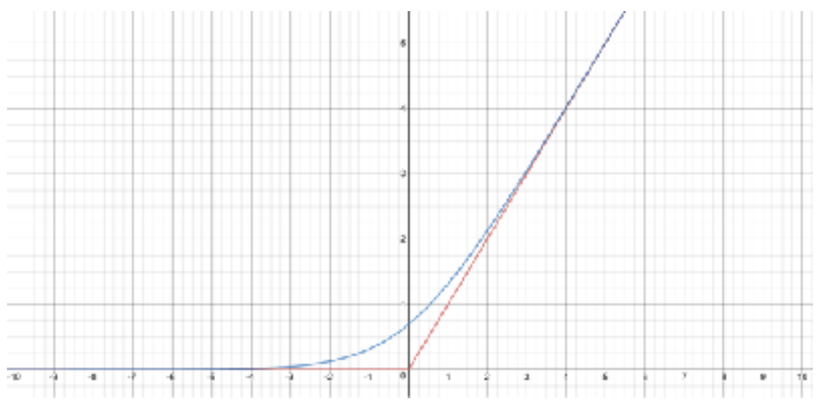
$$g(a) = \sigma(a) = 1/(1 + e^{-a})$$



$$g(a) = \sigma(a + 0.5 - n), n = 1, 2, \ldots$$



$$g(a) = \sum_{n=1}^{\infty} \sigma(a + 0.5 - n) \approx \log(1 + e^a)$$



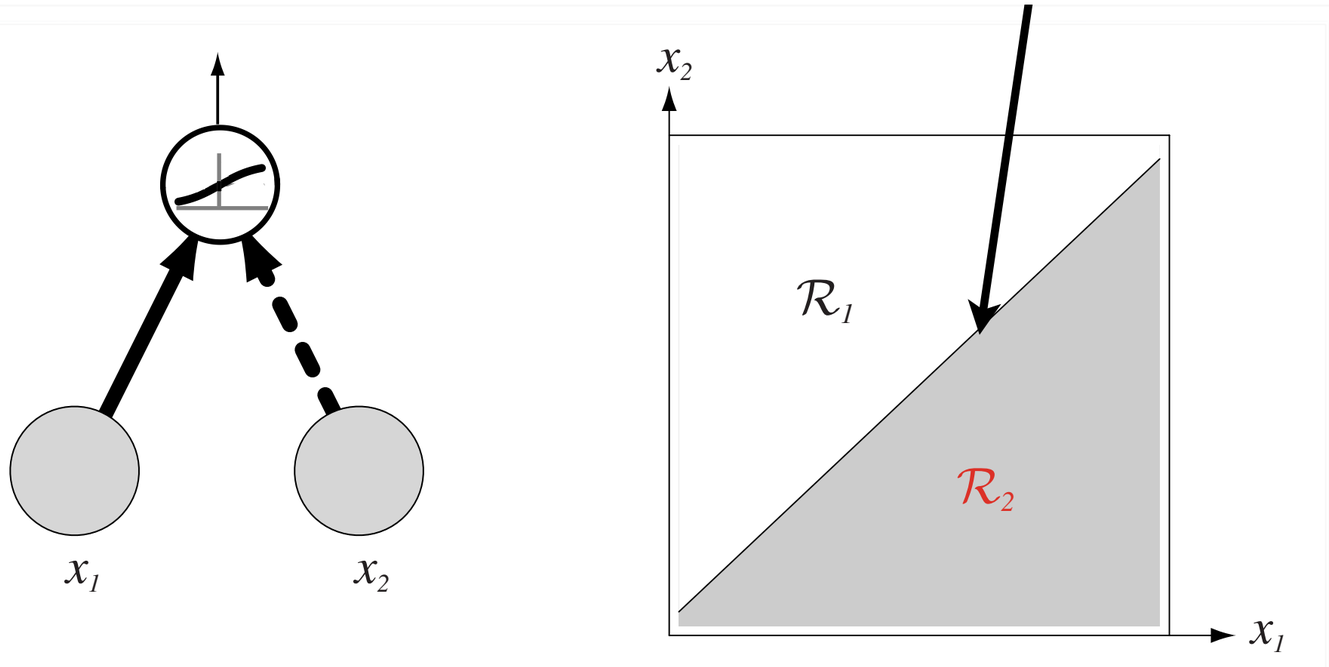
via jefkine.com

Binary Classification
Single Neuron

Could do binary classification:

- with sigmoid, can interpret neuron as estimating $p(y = 1|\boldsymbol{x})$
- also known as logistic regression classifier
  - if output > 0.5, predict class 1
  - otherwise, predict class 0

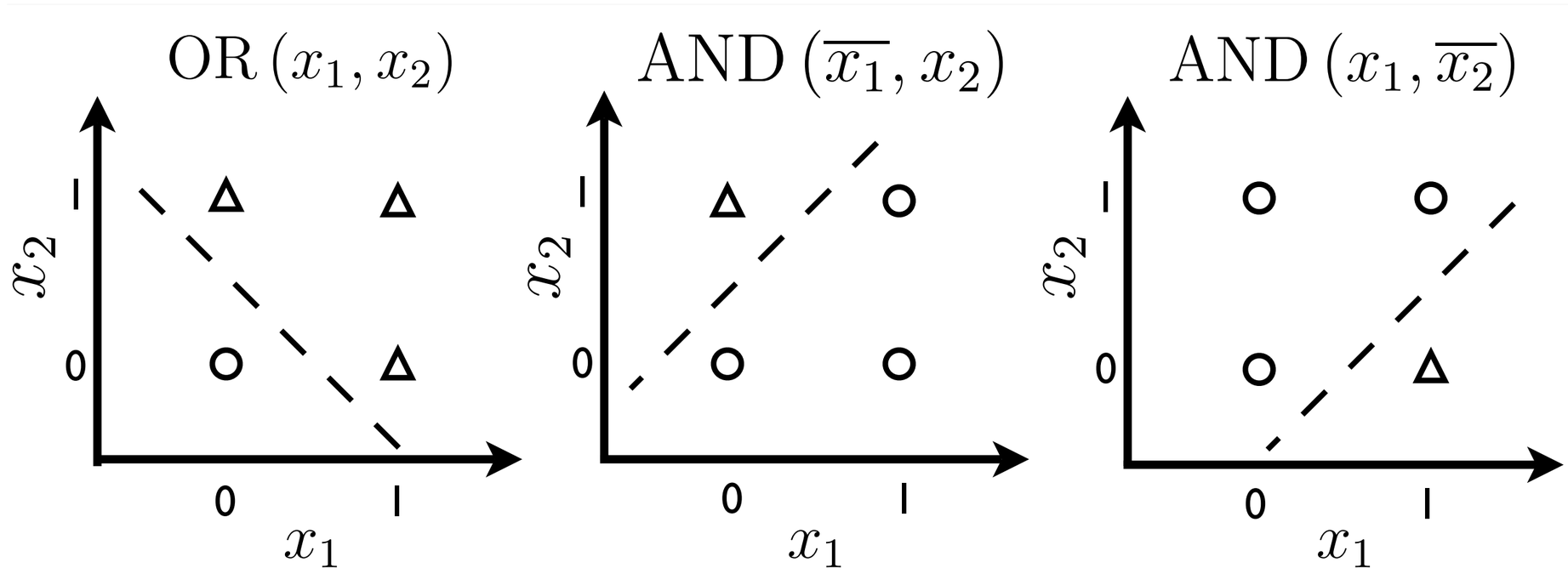
decision boundary is linear



via Hugo Larochelle (reproduced from Pascal Vincent's slides)

Capacity of a Single Neuron

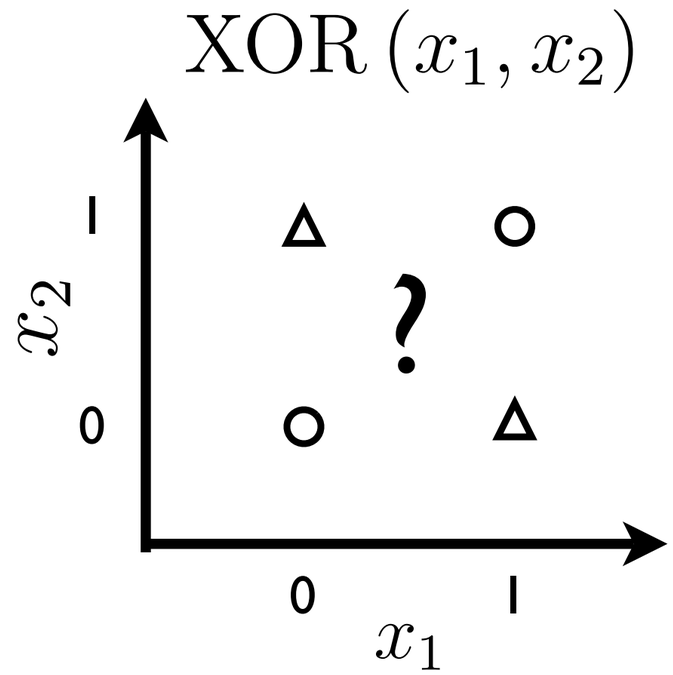
This single neuron is capable of solving linearly separable problems:

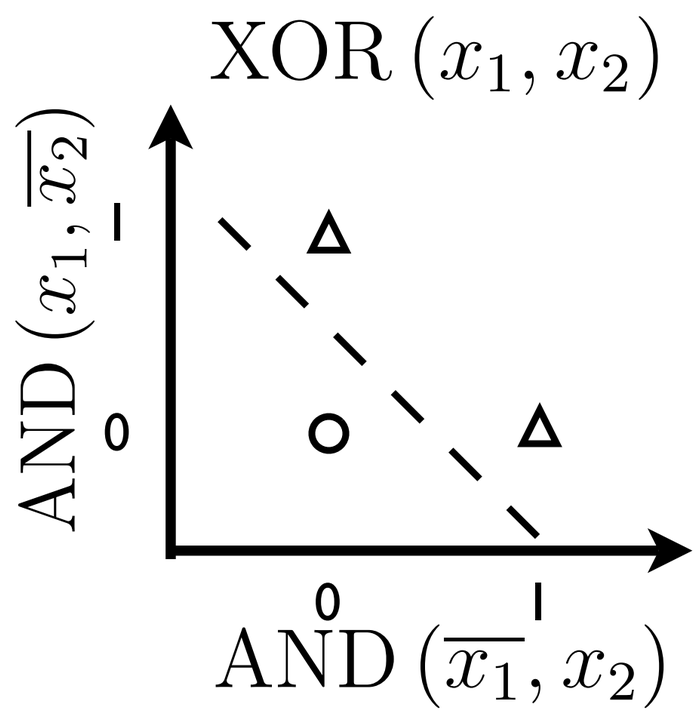


Reproduced from Hugo Larochelle's slides

Nonlinear Separability

However, it fails miserably on
nonlinearly separable problems:





…unless the input is transformed into a better representation

Reproduced from Hugo Larochelle's slides

FeedForward Network With One Hidden Layer

- Hidden layer pre-activation

$$\boldsymbol{a} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$

$$\left(a_i = \sum_j W^{(1)}_{i,j} x_j + b^{(1)}_i\right)$$

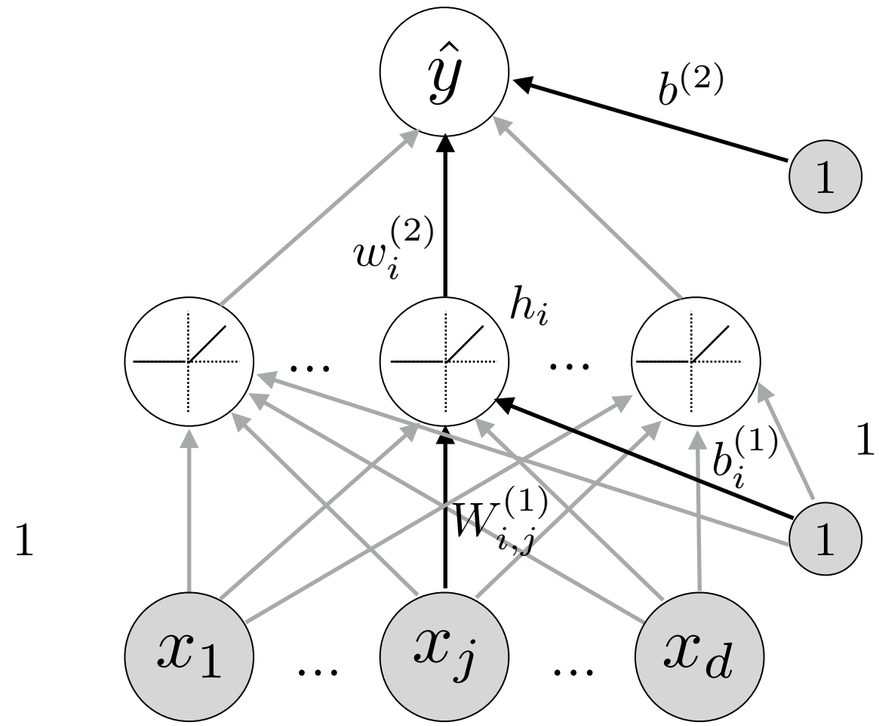
- Hidden layer activation

$$\boldsymbol{h} = g(\boldsymbol{a})$$

- Output layer activation

$$\hat{y} = f(\boldsymbol{x}) = o\left(\boldsymbol{w}^{(2)\top}\boldsymbol{h} + b^{(2)}\right)$$

output activation function



Output Activation Functions

The output activation function is typically matched to the nature of the output:

- To predict real-valued outputs, choose a linear activation
- To predict binary-valued outputs (e.g. binary classification), choose a sigmoid activation
- To predict positive unbounded outputs, choose a ReLU or exponential activation
- What about categorical outputs (e.g. multi-class classification)?

Softmax Activation Function

For multi-class classification:

- we need multiple outputs (1 per class)
- we would like to estimate $p(y = c|\boldsymbol{x}) \quad \forall c \in 1, \ldots, n$

Use the softmax activation at the output:

$$o(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)}, \ldots, \frac{\exp(a_n)}{\sum_c \exp(a_c)}\right]$$

- strictly positive
- sums to one
- Predicted class is the one with highest estimated probability

Reproduced from Hugo Larochelle's slides

Multilayer Neural Network

Could have l layers (hiddens + output)

- Layer pre-activation for $k > 0 \quad (\boldsymbol{h}^{(0)} = \boldsymbol{x})$

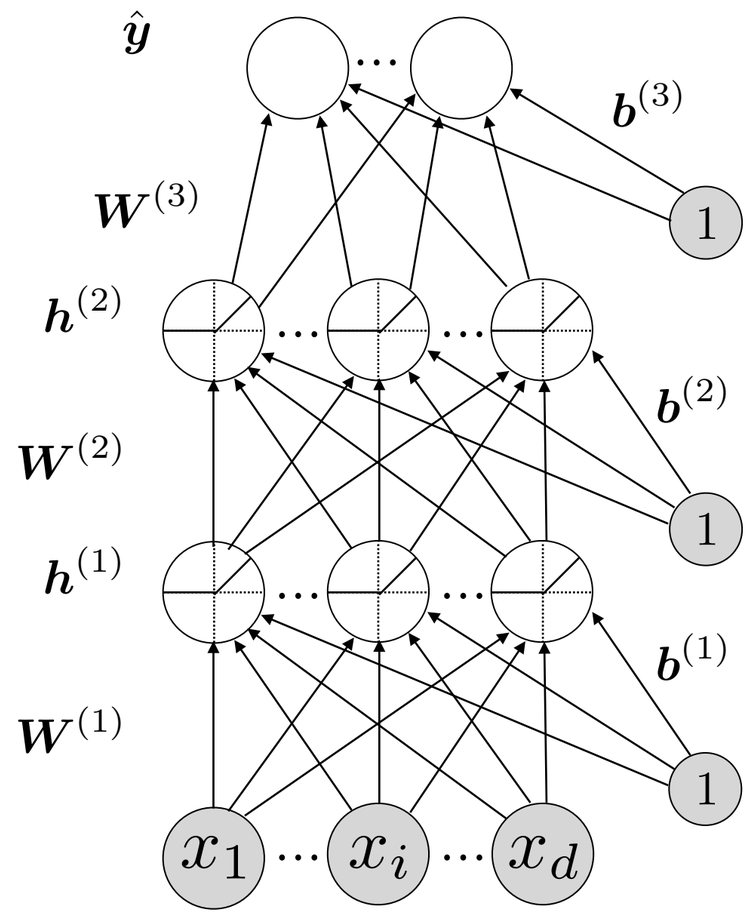
$$\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$$

- Hidden layer activation for $k = 1, \ldots, l-1$

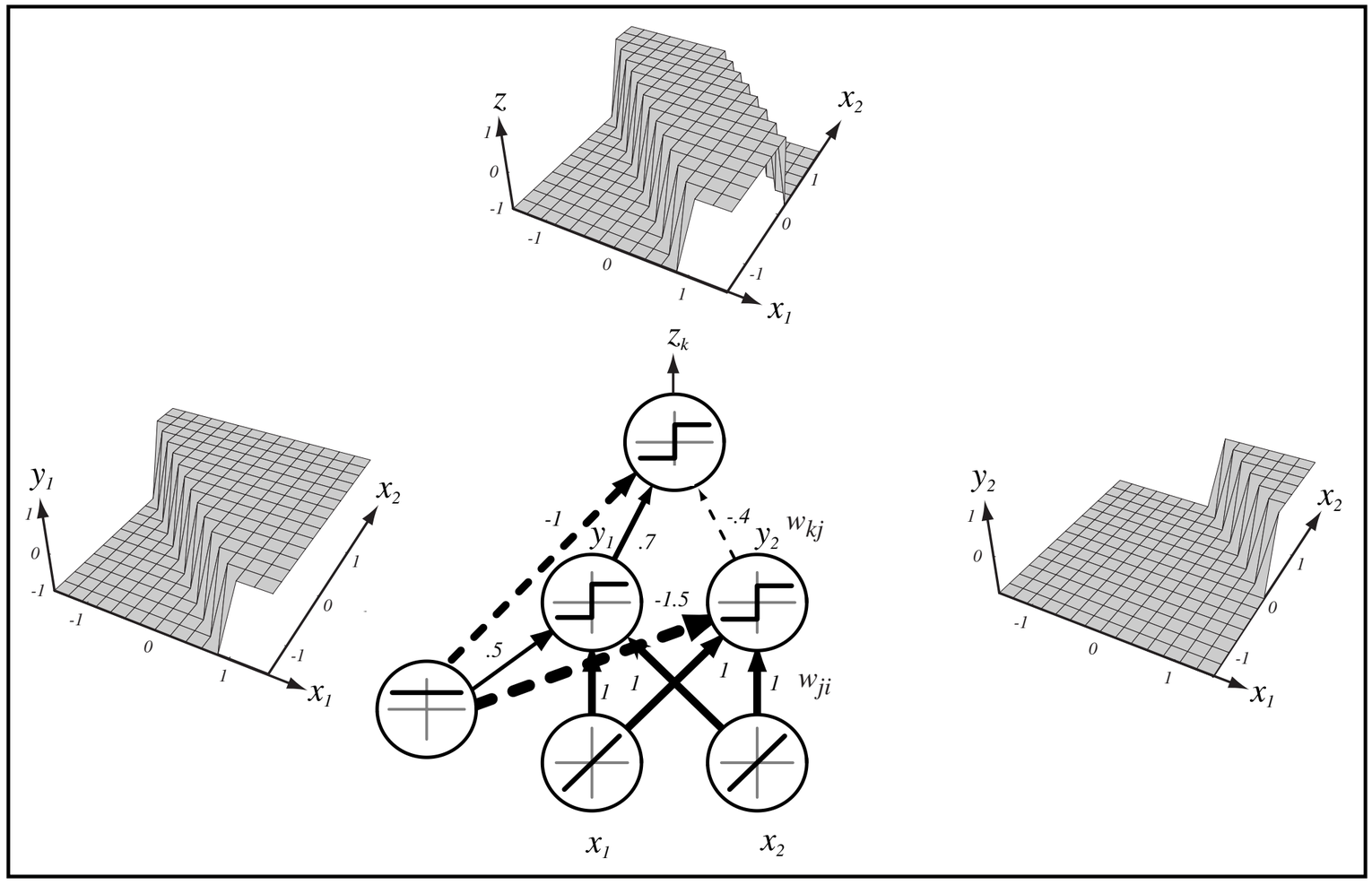
$$\boldsymbol{h}^{(k)} = g\left(\boldsymbol{a}^{(k)}\right)$$

- Output layer activation

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = o(\boldsymbol{a}^{(l)})$$

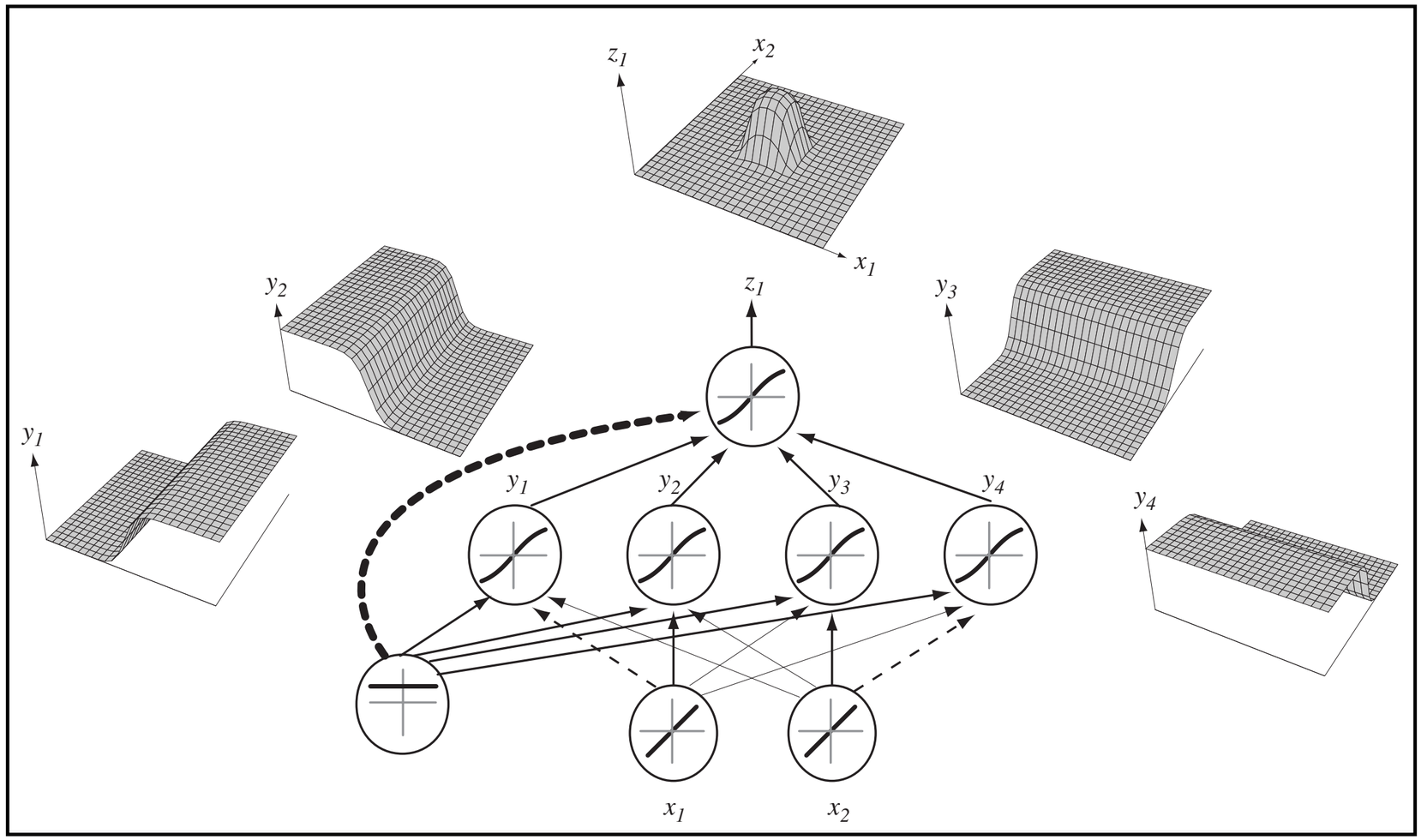


Capacity of a FeedForward Net With One Hidden Layer



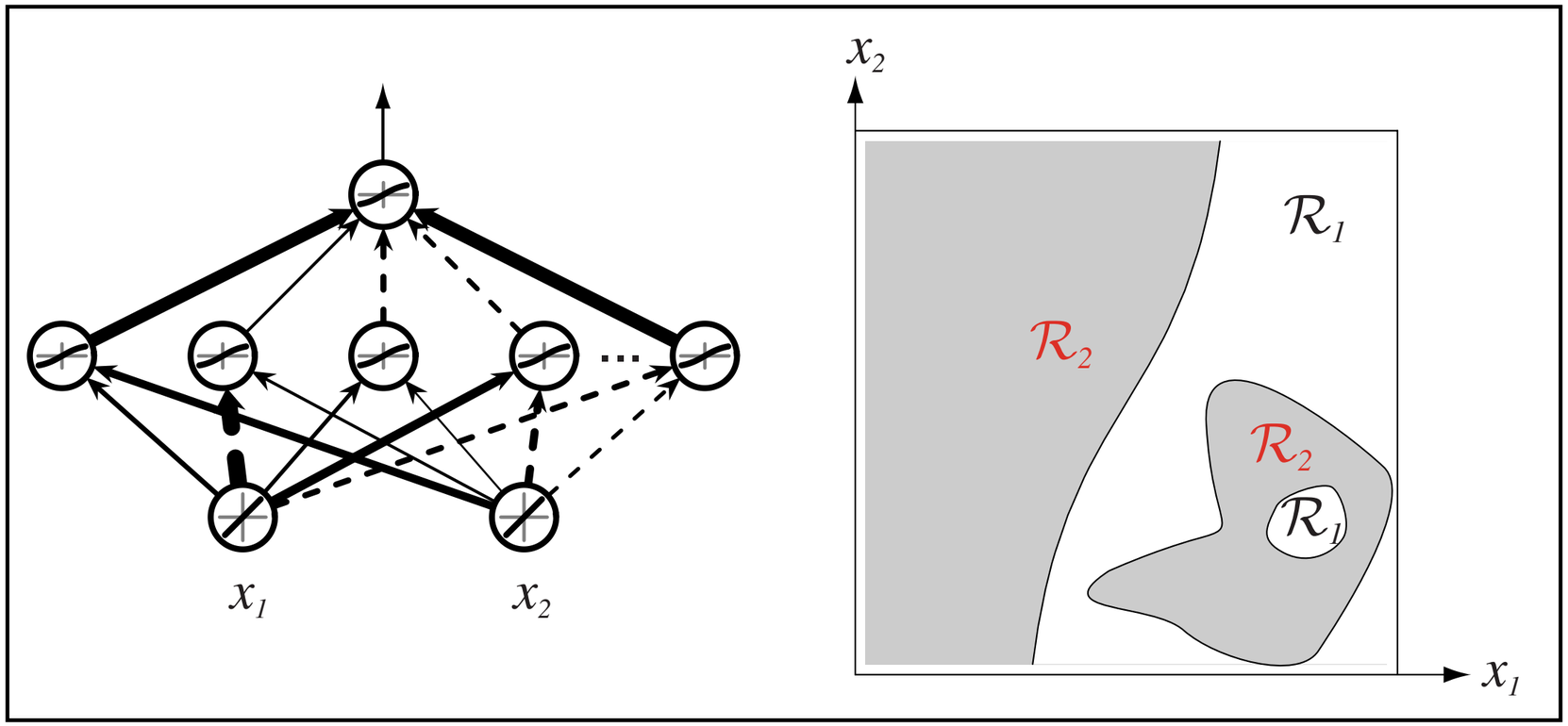
via Hugo Larochelle (reproduced from Pascal Vincent's slides)

Effect of Adding More Hidden Units



via Hugo Larochelle (reproduced from Pascal Vincent's slides)

Nonlinear Decision Boundary



via Hugo Larochelle (reproduced from Pascal Vincent's slides)

Universal Approximation Theorem

- Hornik, 1991: “A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

- The original theorems were first stated in terms of activation functions that saturate (sigmoid, tanh, etc.) but have been proved now for a wider class of activation functions, including ReLU

- Note that this doesn’t mean there is a learning algorithm that can find the right parameters!