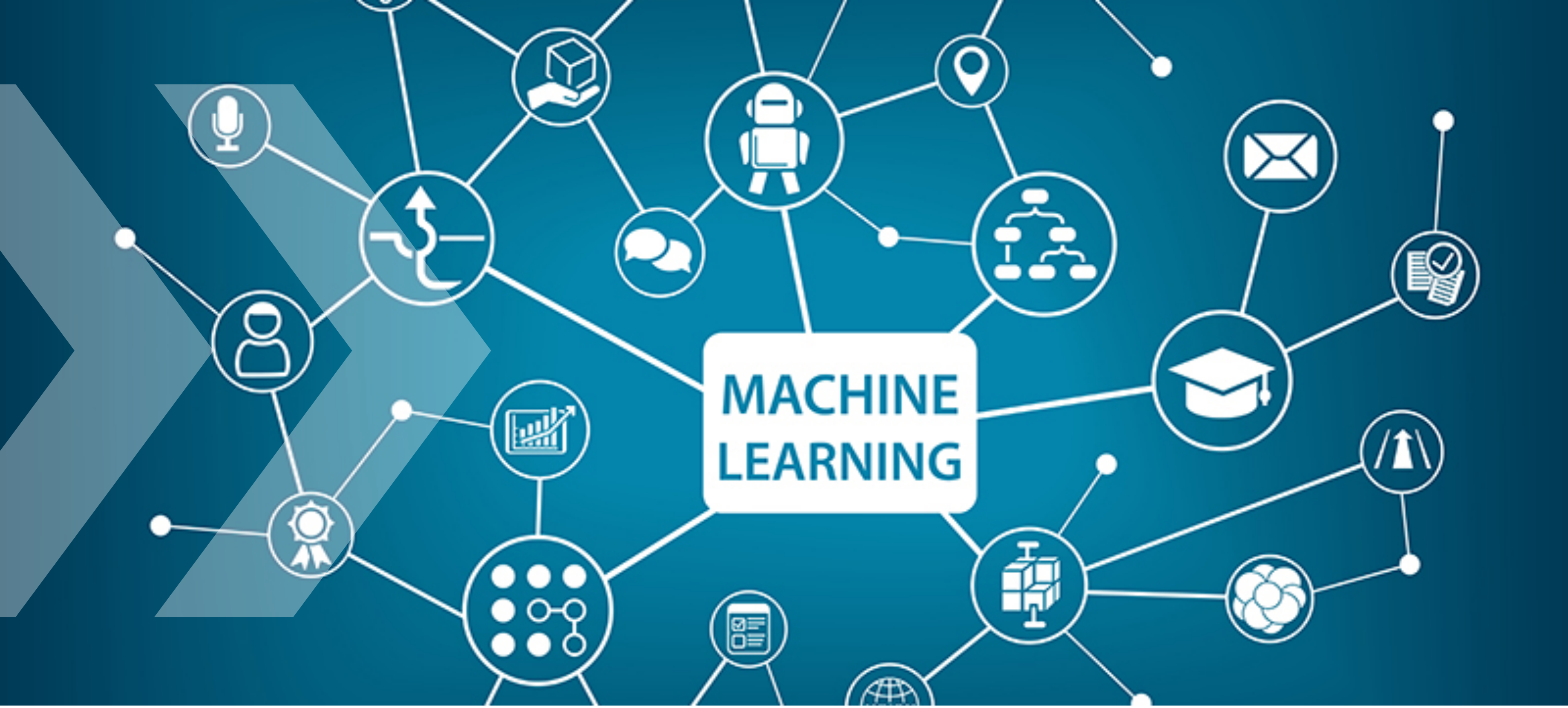


CS480/680: Intro to ML

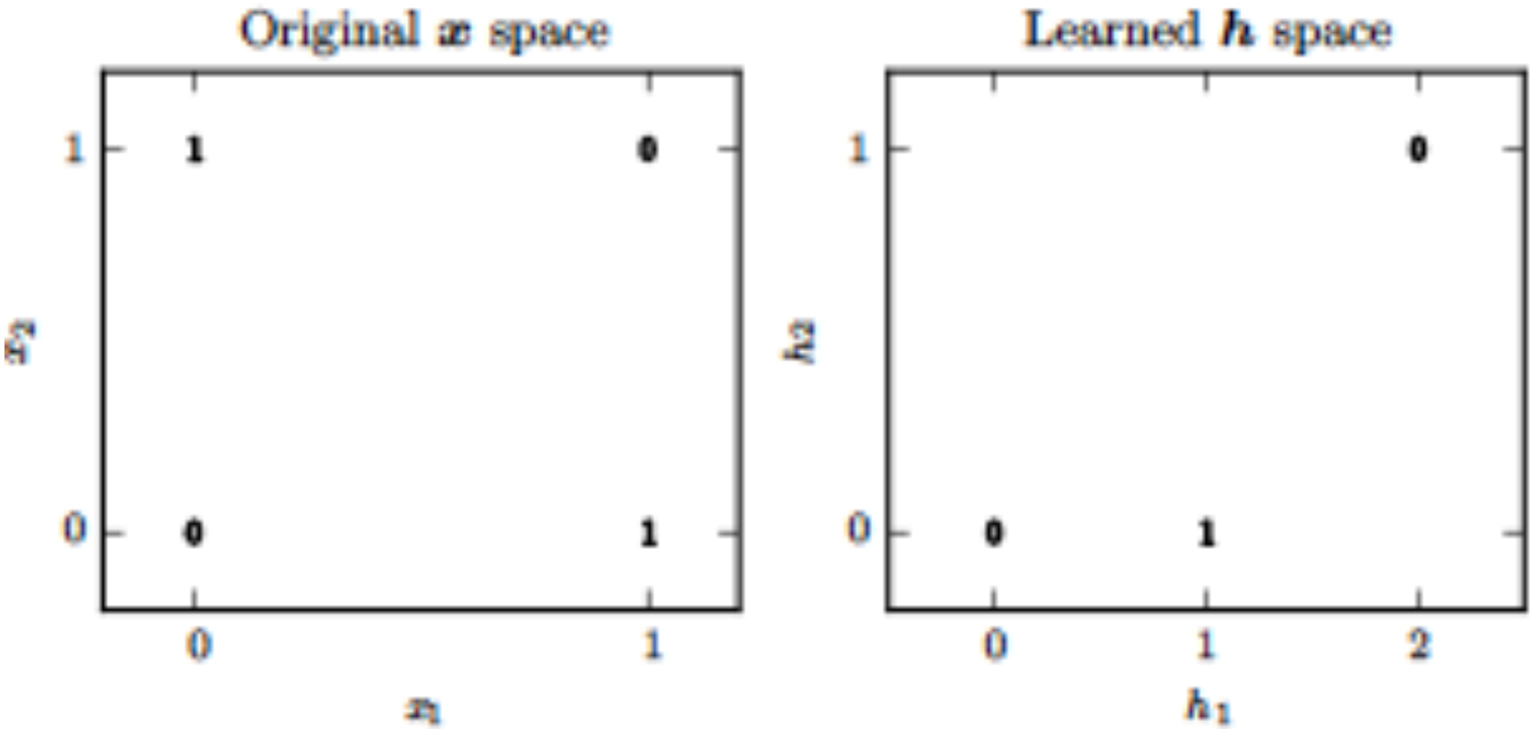
Lecture 12: Multi-layer Perceptron

UNIVERSITY OF WATERLOO

Outline

- Failure of Perceptron
- Neural Network
- Backpropagation
- Universal Approximator

The XOR problem



- $X = \{ (0,0), (0,1), (1,0), (1,1) \}$, $y = \{-1, 1, 1, -1\}$
- $f^*(\mathbf{x}) = \mathrm{sign}[f_{xor} - ½]$, $f_{xor}(\mathbf{x}) = (x_1 \& \sim x_2) \mid (\sim x_1 \& x_2)$

No separating hyperplane

$$y(\langle \mathbf{w}, \mathbf{x} \rangle + b) > 0$$

- $\mathbf{x}_1 = (0,0)$, $y_1 = -1$ → $b < 0$
- $\mathbf{x}_2 = (0,1)$, $y_2 = 1$ → $w_2 + b > 0$
- $\mathbf{x}_3 = (1,0)$, $y_3 = 1$ → $w_1 + b > 0$

$$\left.\begin{array}{l} w_2 + b > 0 \\ w_1 + b > 0 \end{array}\right\} (w_1 + w_2 + b) + b > 0$$

- $\mathbf{x}_4 = (1,1)$, $y_4 = -1$ → $w_1 + w_2 + b < 0$

Ex. what happens if we run perceptron on this example?

- Contradiction!
- [At least one of the blue or green inequalities has to be strict]

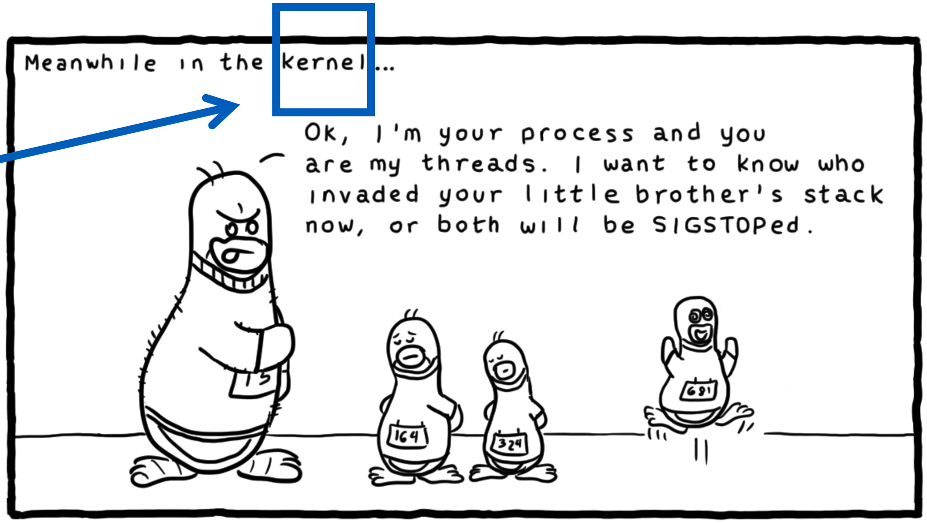
Fixing the problem

- Our model (hyperplanes) underfits the data (xor func)

- Fix representation, richer model

‖

- Fix model, richer representation


Meanwhile in the kernel...
Ok, I'm your process and you are my threads. I want to know who invaded your little brother's stack now, or both will be SIGSTOPed.
15
164
324
691

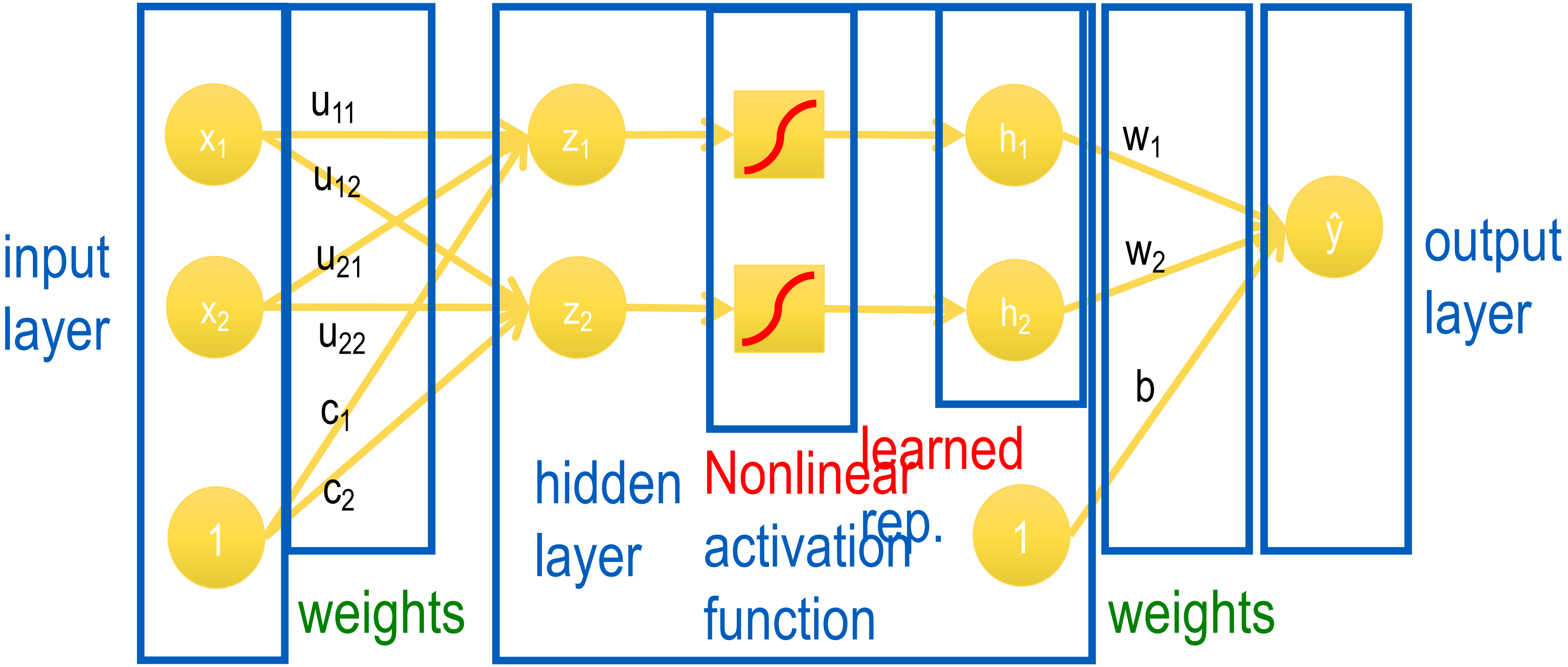
Daniel Stori {turnoff.us}

- NN: still use hyperplane, but learn representation simultaneously

Outline

- Failure of Perceptron
- Neural Network
- Backpropagation
- Universal Approximator

Two-layer perceptron



input layer

weights

hidden layer

Nonlinear activation function

learned rep.

weights

output layer

$$\mathbf{z} = U\mathbf{x} + \mathbf{c}$$

1st linear layer

makes all the difference!

$$\mathbf{h} = f(\mathbf{z})$$

nonlinear transform

2nd linear layer

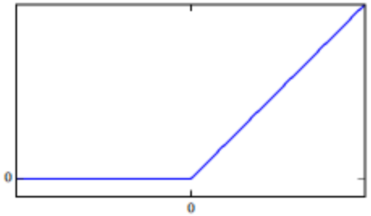
$$\hat{y} = \langle \mathbf{h}, \mathbf{w} \rangle + b$$

Does it work?

$$U = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} 2 \\ -4 \end{bmatrix} \quad b = -1$$

Rectified Linear Unit (ReLU)

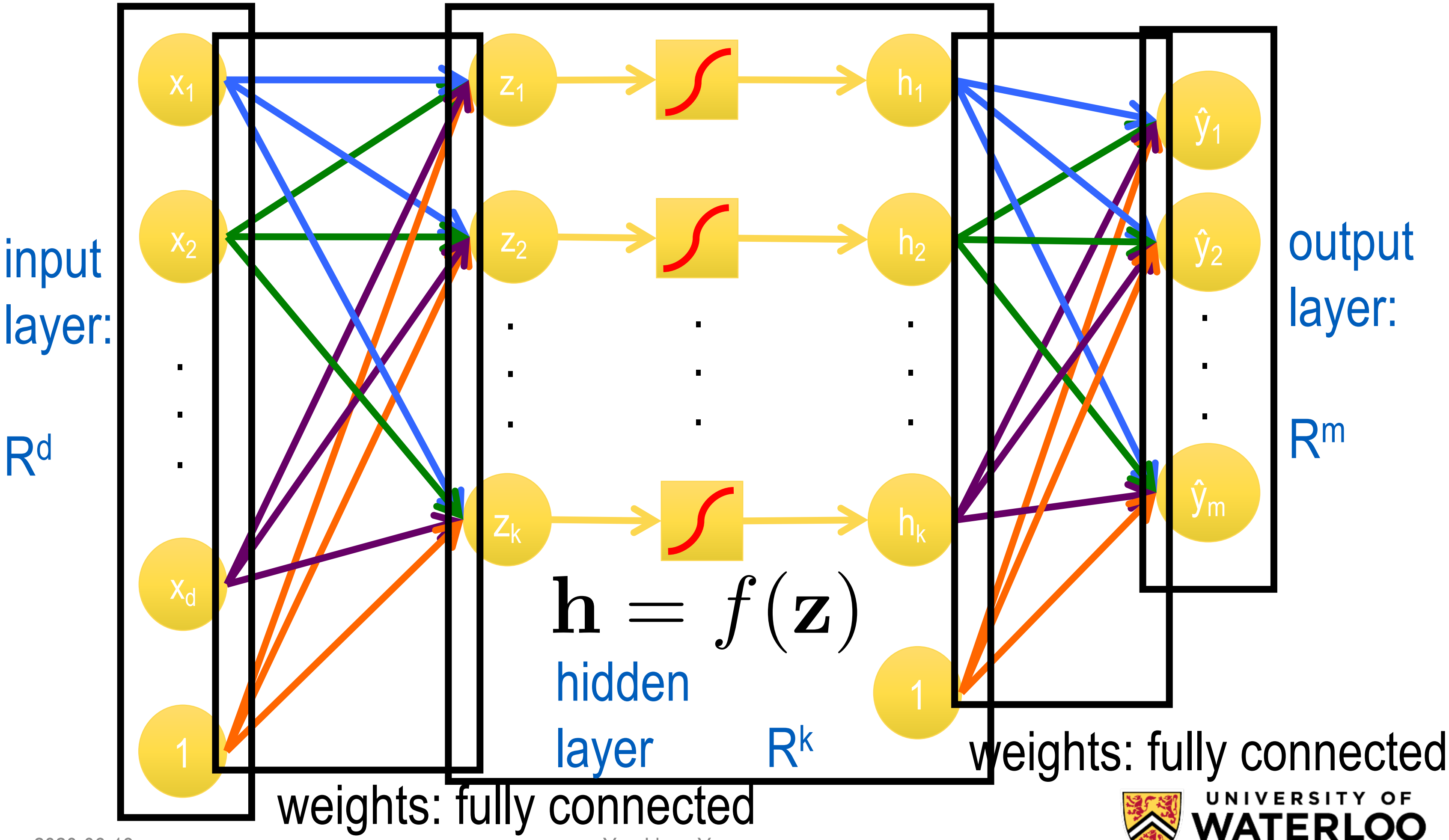
$$f(t) = t_+ := \max(t, 0)$$



- $\mathbf{x}_1 = (0,0)$, $y_1 = -1$ → $\mathbf{z}_1 = (0,-1)$, $\mathbf{h}_1 = (0,0)$ → $\hat{y}_1 = -1$
- $\mathbf{x}_2 = (0,1)$, $y_2 = 1$ → $\mathbf{z}_2 = (1,0)$, $\mathbf{h}_2 = (1,0)$ → $\hat{y}_2 = 1$
- $\mathbf{x}_3 = (1,0)$, $y_3 = 1$ → $\mathbf{z}_3 = (1,0)$, $\mathbf{h}_3 = (1,0)$ → $\hat{y}_3 = 1$
- $\mathbf{x}_4 = (1,1)$, $y_4 = -1$ → $\mathbf{z}_4 = (2,1)$, $\mathbf{h}_4 = (2,1)$ → $\hat{y}_4 = -1$

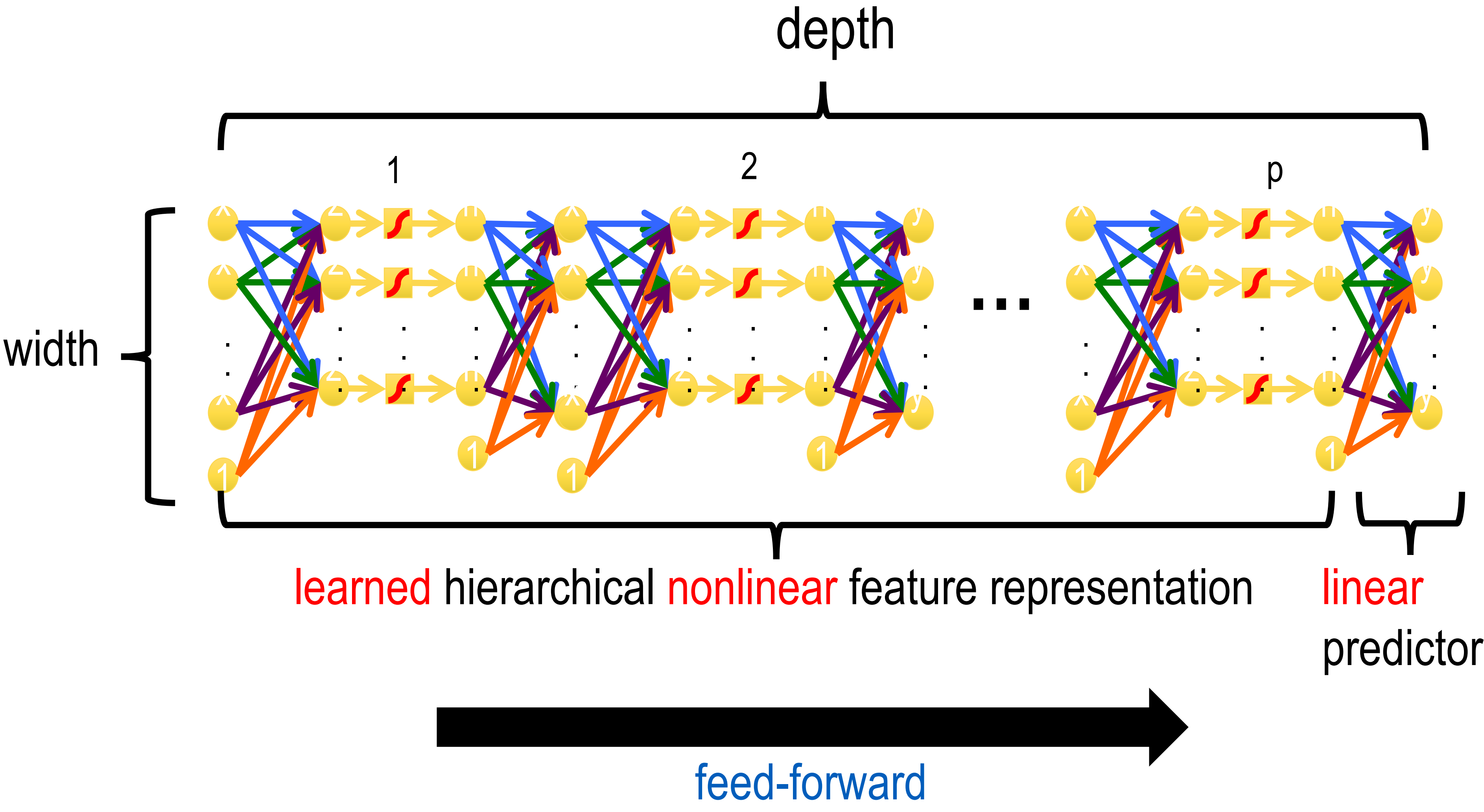
Multi-layer perceptron

$$\mathbf{z} = U\mathbf{x} + \mathbf{c} \qquad \hat{\mathbf{y}} = W\mathbf{h} + \mathbf{b}$$



$$\mathbf{h} = f(\mathbf{z})$$

Multi-layer perceptron (stacked)



depth

width

learned hierarchical nonlinear feature representation

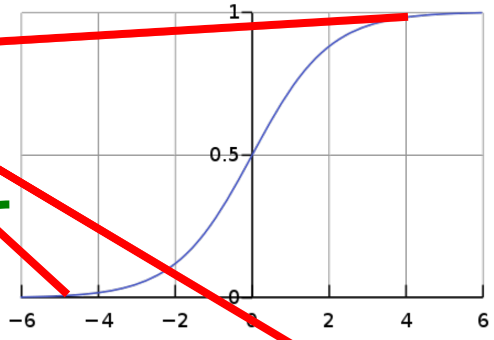
linear predictor

feed-forward

Activation function

- Sigmoid

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

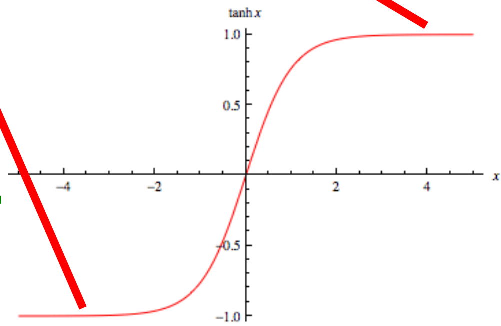


saturate

smooth

- Tanh

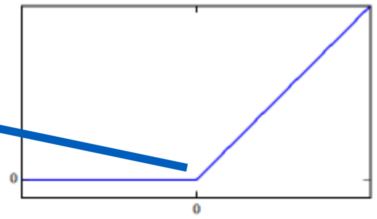
$$f(t) = \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$



- Rectified Linear

$$f(t) = t_+ := \max(t, 0)$$

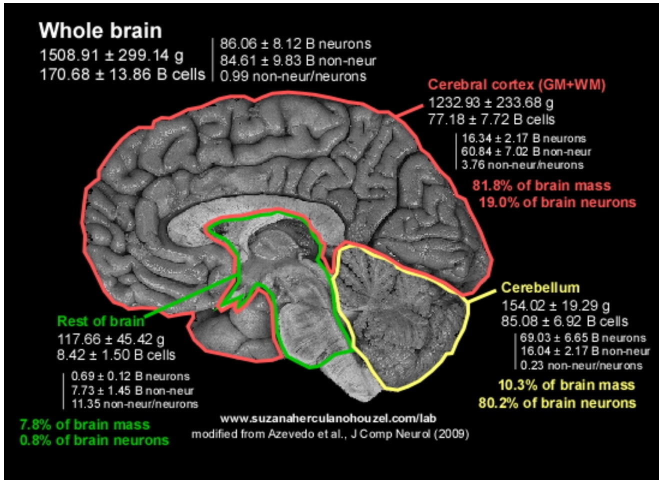
nonsmooth



Underfitting vs. Overfitting

- Linear predictor (perceptron / winnow / linear regression) underfits

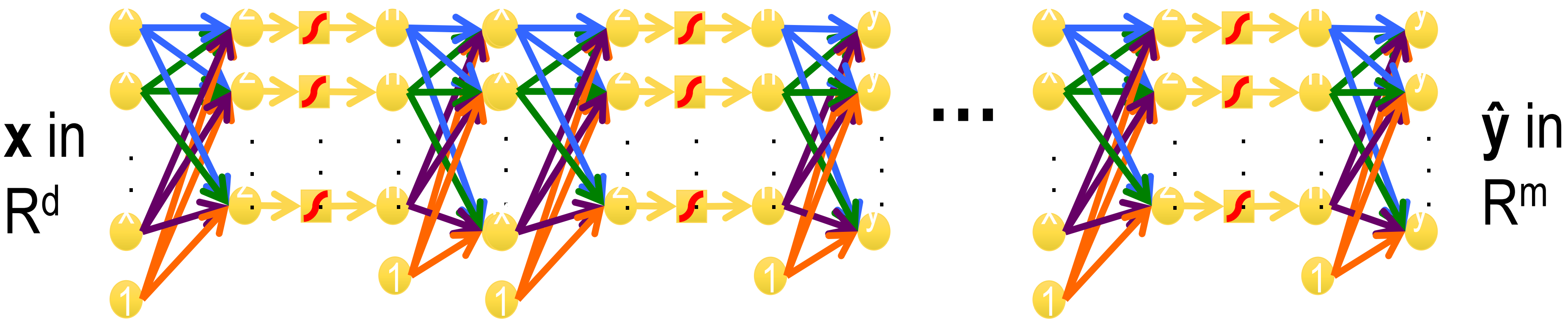
- NNs learn *hierarchical nonlinear* feature jointly with linear predictor
  - may overfit
  - tons of heuristics (some later)



www.suzanaherculanohouzel.com/lab
modified from Azevedo et al., J Comp Neurol (2009)

- Size of NN: # of weights/connections
  - ~ 1 billion; human brain 10^6 billions (estimated)

Weights training



$$\boxed{\hat{\mathbf{y}} = \mathbf{q}(\mathbf{x};\, \Theta)}$$

- Need a loss ℓ to measure diff. between pred **ŷ** and truth **y**
  - E.g., $(\hat{\mathbf{y}}-\mathbf{y})^2$; more later

- Need a training set $\{(\mathbf{x}_1,\mathbf{y}_1), \ldots, (\mathbf{x}_n,\mathbf{y}_n)\}$ to train weights Θ

Gradient Descent

$$\boxed{\min_{\Theta}} L(\Theta) := \frac{1}{n}\sum_{i=1}^{n} \ell\left[\mathbf{q}(\mathbf{x}_i;\Theta), \mathbf{y}_i\right]$$

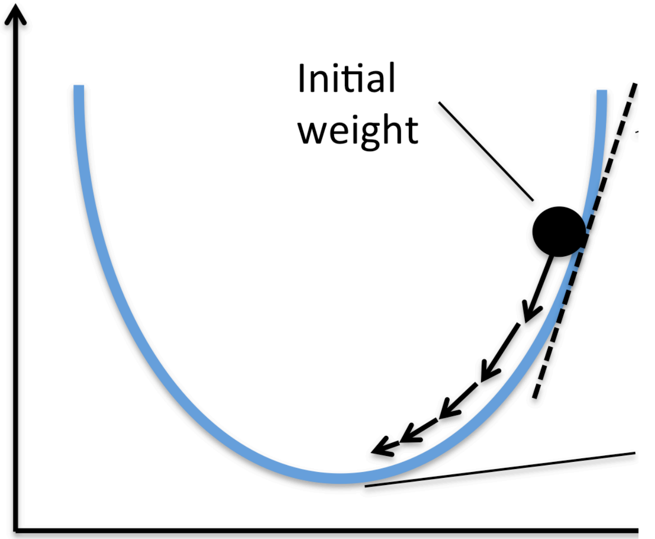
$$\boxed{\Theta_{t+1} \leftarrow \Theta_t \boxed{-} \eta_t \nabla L(\Theta_t)}$$

(Generalized) gradient

O(n) !

Step size (learning rate)

- const., if L is smooth
- diminishing, otherwise



Stochastic Gradient Descent (SGD)

$$\Theta_{t+1} = \Theta_t - \eta_t \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla \ell \Big[\mathbf{q}(\mathbf{x}_i; \Theta_t), \mathbf{y}_i \Big]$$

average over *n* samples

a random sample suffices

$$\boxed{\Theta_{t+1} = \Theta_t - \eta_t \nabla \ell \Big[\mathbf{q}(\mathbf{x}_{i_t}; \Theta_t), \mathbf{y}_{i_t} \Big]}$$

- diminishing step size, e.g., 1/sqrt{t} or 1/t
- averaging, momentum, variance-reduction, etc.
- sample w/o replacement; cycle; permute in each pass

Outline

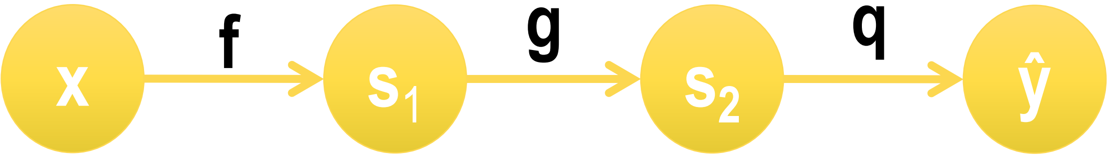
- Failure of Perceptron
- Neural Network
- Backpropagation
- Universal Approximator

Backpropogation

- Efficient way to compute the derivative in NN

- Two passes; complexity = O(size(NN))
    - forward pass: compute function value sequentially
    - backward pass: compute derivative sequentially

Forward differentiation



$$\hat{y} = q(\, g(\, f(x)\,)\,)$$

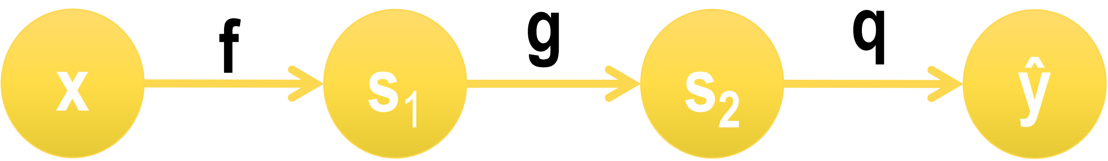
$$s_1 = f(x), \qquad s_2 = g(s_1), \qquad \hat{y} = q(s_2)$$

$$\frac{dL}{dx} = \frac{dL}{d\hat{y}}(\hat{y}) \times \frac{d\hat{y}}{ds_2}(s_2) \times \frac{ds_2}{ds_1}(s_1) \times \frac{ds_1}{dx}(x)$$

$$\xleftarrow{\qquad\qquad\qquad\qquad} \quad \frac{d}{dx}$$

$$= \boxed{L'(\hat{y}) \times} \boxed{q'(s_2) \times} \boxed{g'(s_1) \times} \boxed{f'(x)}$$

Backward differentiation



$$\hat{y} = q(\,g(\,f(x)\,)\,)$$

$s_1 = f(x)$, $s_2 = g(s_1)$, $\hat{y} = q(s_2)$

$$\frac{dL}{dx} = \frac{dL}{d\hat{y}}(\hat{y}) \times \frac{d\hat{y}}{ds_2}(s_2) \times \frac{ds_2}{ds_1}(s_1) \times \frac{ds_1}{dx}(x)$$

$$\frac{dL}{d} = \underrightarrow{\boxed{L'(\hat{y})} \times \boxed{q'(s_2)} \times \boxed{g'(s_1)} \times \boxed{f'(x)}}$$

Which way is cheaper?

- L: R^{c} → R^{e}, therefore L' in $R^{e \times c}$
- q: R^{b} → R^{c}, therefore q' in $R^{c \times b}$
- g: R^{a} → R^{b}, therefore g' in $R^{b \times a}$
- f: R^{d} → R^{a}, therefore f' in $R^{a \times d}$

- Forward: $O(bad + cbd + ecd) = O(d(ba+cb+ec))$
- Backward: $O(ecb + eba + ead) = O(e(ba+cb+ad))$

- Typically, output dim $e = 1$, input dim $d >> 1$

Tradeoff

$$s_1 = f(x), \qquad s_2 = g(s_1), \qquad \hat{y} = q(s_2)$$

- Forward: $\overleftarrow{L'(\hat{y}) \times q'(s_2) \times g'(s_1) \times f'(x)}$
  - computation is on-the-fly
- Backward: $\overrightarrow{L'(\hat{y}) \times q'(s_2) \times g'(s_1) \times f'(x)}$
  - need to store $\hat{y}$, s_2, s_1

Outline

- Failure of Perceptron
- Neural Network
- Backpropagation
- Universal Approximator

Rationals are dense in R

- Any real number can be approximated by some rational number arbitrarily well

- Or in fancy mathematical language

$$\forall r \in \mathbf{R}, \forall \epsilon > 0, \exists s \in \mathbf{Q}, \text{ such that } |r - s| < \epsilon$$

domain of interest

what we can rep.

metric for approx.

Universal Approximator

Theorem (Cybenko, Hornik et al., Leshno et al., …). Any continuous function *g*: $[0,1]^d$ → R can be uniformly approximated in arbitrary precision by a two-layer NN with an activation function *f* that is not a polynomial

- for deep networks, any f that is not affine would do
- conditions are necessary in some sense
- includes (almost) all activation functions in practice

Caveat and Remedy

- NNs were praised for being “universal”
  - but many kernels are universal as well
  - desirable but perhaps not THE explanation

- May need exponentially many hidden units…

- Increase depth may reduce network size, exponentially!

Questions?