

# Fundamental Belief: Universal Approximation Theorems

---

**Ju Sun**

Computer Science & Engineering  
University of Minnesota, Twin Cities

January 29, 2020

- HW 0 posted (due: midnight Feb 07)

- HW 0 posted (due: midnight Feb 07)
- Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2ed) now available at UMN library (limited e-access)

- HW 0 posted (due: midnight Feb 07)
- Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2ed) now available at UMN library (limited e-access)
- Guest lectures (Feb 04: Tutorial on Numpy, Scipy, Colab. Bring your laptops if possible!)

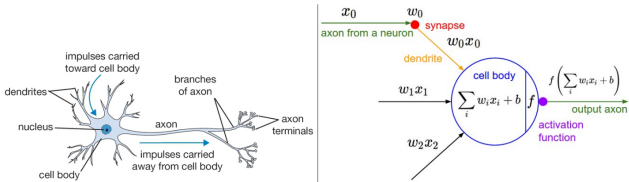
- HW 0 posted (due: midnight Feb 07)
- Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2ed) now available at UMN library (limited e-access)
- Guest lectures (Feb 04: Tutorial on Numpy, Scipy, Colab. Bring your laptops if possible!)
- Feb 06: discussion of the course project & ideas

Recap

Why should we trust NNs?

Suggested reading

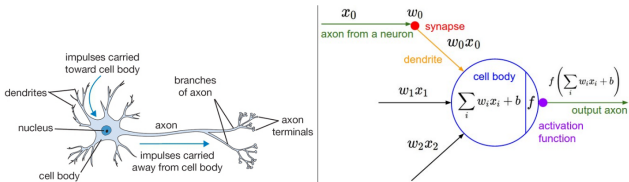
# Recap I



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

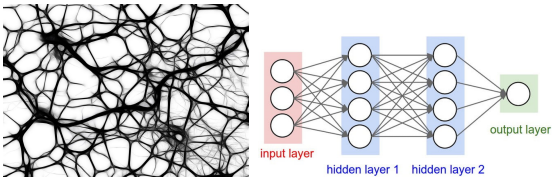
## biological neuron vs. artificial neuron

# Recap I



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

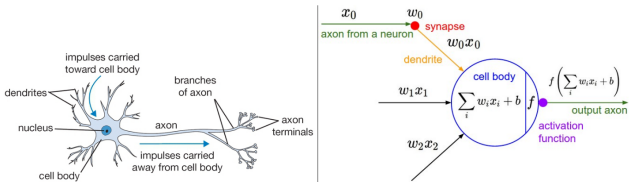
## biological neuron vs. artificial neuron



## biological NN vs. artificial NN

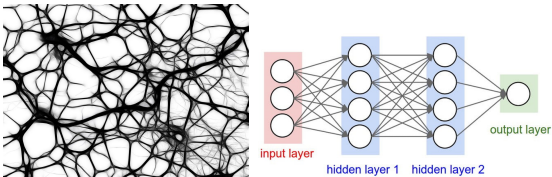


# Recap I



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

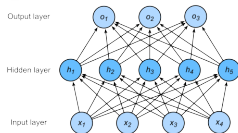
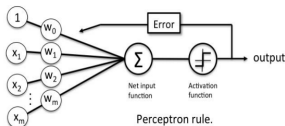
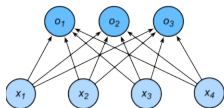
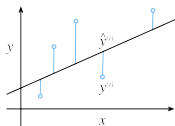
## biological neuron vs. artificial neuron



## biological NN vs. artificial NN

Artificial NN: (over)-simplification on **neuron** & **connection** levels

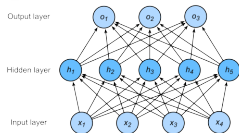
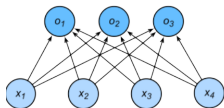
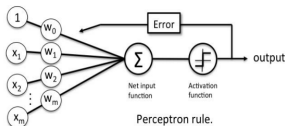
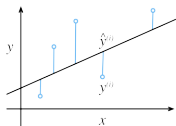
## Zoo of NN models in ML



- Linear regression
- Perception and Logistic regression
- Softmax regression
- Multilayer perceptron (feedforward NNs)

# Recap II

## Zoo of NN models in ML

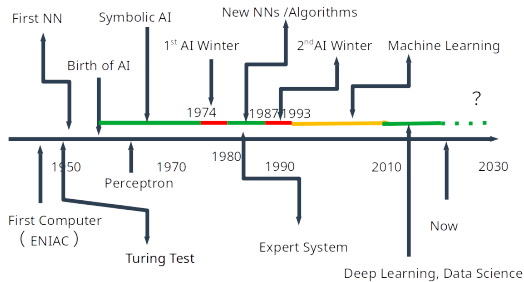


Also:

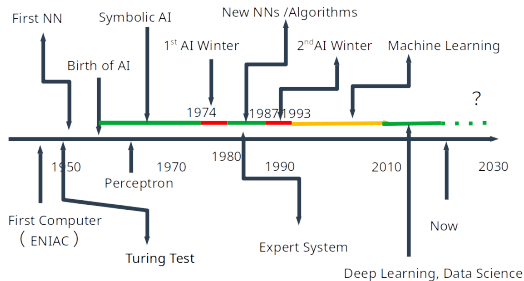
- Support vector machines (SVM)
- PCA (autoencoder)
- Matrix factorization

- Linear regression
- Perception and Logistic regression
- Softmax regression
- Multilayer perceptron (feedforward NNs)

# Recap III



# Recap III



## Brief history of NNs:

- 1943: first NNs invented (McCulloch and Pitts)
- 1958 –1969: perceptron (Rosenblatt)
- 1969: *Perceptrons* (Minsky and Papert)—end of perceptron
- 1980's–1990's: Neocognitron, CNN, back-prop, SGD—we use today
- 1990's–2010's: SVMs, Adaboosting, decision trees and random forests
- 2010's–now: DNNs and deep learning

Recap

Why should we trust NNs?

Suggested reading

# Supervised learning

## General view:

- Gather training data  $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)$
- Choose a family of functions, e.g.,  $\mathcal{H}$ , so that there is  $f \in \mathcal{H}$  to ensure  $\mathbf{y}_i \approx f(\mathbf{x}_i)$  for all  $i$
- Set up a loss function  $\ell$
- Find an  $f \in \mathcal{H}$  to minimize the average loss

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \ell(\mathbf{y}_i, f(\mathbf{x}_i))$$

## NN view:

- Gather training data  $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)$
- Choose a NN with  $k$  neurons, so that there is a group of weights, e.g.,  $(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)$ , to ensure  $\mathbf{y}_i \approx \{\text{NN}(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)\}(\mathbf{x}_i) \quad \forall i$
- Set up a loss function  $\ell$
- Find weights  $(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)$  to minimize the average loss

$$\min_{\mathbf{w}'s, b's} \frac{1}{n} \sum_{i=1}^n \ell[\mathbf{y}_i, \{\text{NN}(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)\}(\mathbf{x}_i)]$$

# Supervised learning

## General view:

- Gather training data  
 $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)$
- Choose a family of functions, e.g.,  $\mathcal{H}$ , so that there is  $f \in \mathcal{H}$  to ensure  $\mathbf{y}_i \approx f(\mathbf{x}_i)$  for all  $i$
- Set up a loss function  $\ell$
- Find an  $f \in \mathcal{H}$  to minimize the average loss

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \ell(\mathbf{y}_i, f(\mathbf{x}_i))$$

## NN view:

- Gather training data  
 $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)$
- Choose a NN with  $k$  neurons, so that there is a group of weights, e.g.,  $(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)$ , to ensure  $\mathbf{y}_i \approx \{\text{NN}(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)\}(\mathbf{x}_i) \quad \forall i$
- Set up a loss function  $\ell$
- Find weights  $(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)$  to minimize the average loss

$$\min_{\mathbf{w}'_s, b'_s} \frac{1}{n} \sum_{i=1}^n \ell[\mathbf{y}_i, \{\text{NN}(\mathbf{w}_1, \dots, \mathbf{w}_k, b_1, \dots, b_k)\}(\mathbf{x}_i)]$$

Why should we trust NNs?

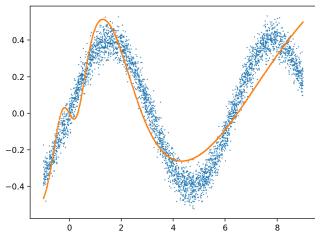


## Function approximation

More accurate description of supervised learning

# Function approximation

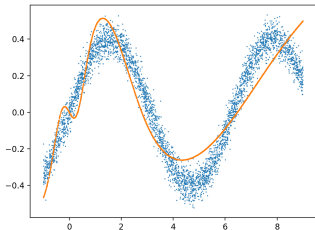
More accurate description of supervised learning



– Underlying true function:  $f_0$

# Function approximation

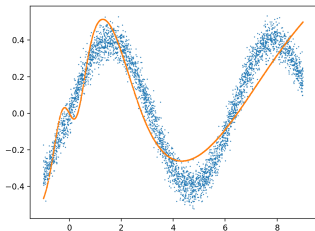
More accurate description of supervised learning



- Underlying true function:  $f_0$
- Training data:  $y_i \approx f_0(x_i)$

# Function approximation

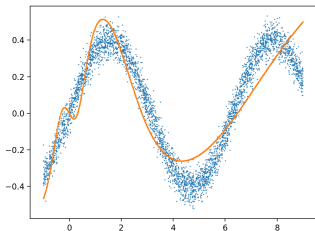
More accurate description of supervised learning



- Underlying true function:  $f_0$
- Training data:  $\mathbf{y}_i \approx f_0(\mathbf{x}_i)$
- Choose a family of functions  $\mathcal{H}$ , so that  $\exists f \in \mathcal{H}$  and  
 $f$  and  $f_0$  are close

# Function approximation

More accurate description of supervised learning

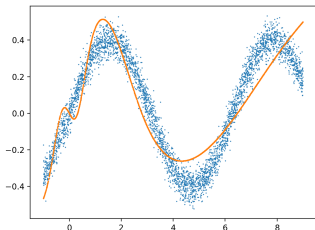


- Underlying true function:  $f_0$
- Training data:  $y_i \approx f_0(x_i)$
- Choose a family of functions  $\mathcal{H}$ , so that  $\exists f \in \mathcal{H}$  and  
 $f$  and  $f_0$  are close

- **Approximation capacity:**  $\mathcal{H}$  matters (e.g., linear? quadratic? sinusoids? etc)

# Function approximation

More accurate description of supervised learning

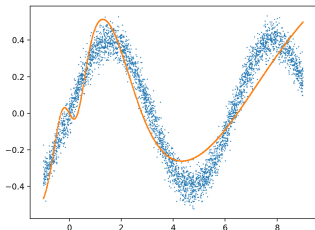


- Underlying true function:  $f_0$
- Training data:  $y_i \approx f_0(x_i)$
- Choose a family of functions  $\mathcal{H}$ , so that  $\exists f \in \mathcal{H}$  and  
 $f$  and  $f_0$  are close

- **Approximation capacity:**  $\mathcal{H}$  matters (e.g., linear? quadratic? sinusoids? etc)
- **Optimization & Generalization:** how to find the best  $f \in \mathcal{H}$  matters

# Function approximation

More accurate description of supervised learning

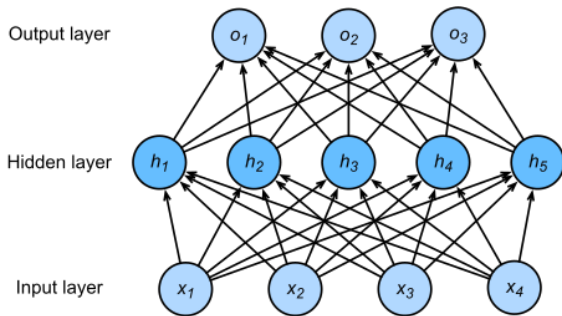


- Underlying true function:  $f_0$
- Training data:  $y_i \approx f_0(x_i)$
- Choose a family of functions  $\mathcal{H}$ , so that  $\exists f \in \mathcal{H}$  and  $f$  and  $f_0$  are close

- **Approximation capacity:**  $\mathcal{H}$  matters (e.g., linear? quadratic? sinusoids? etc)
- **Optimization & Generalization:** how to find the best  $f \in \mathcal{H}$  matters

We focus on approximation capacity now.

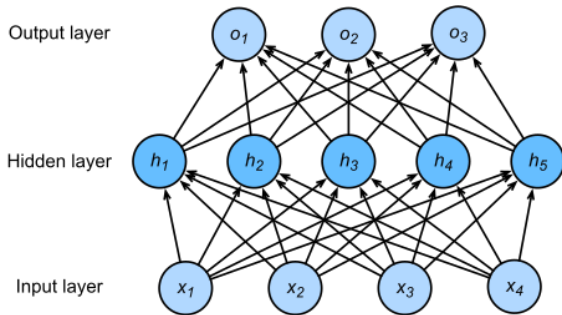
## A word on notation



- $k$ -layer NNs: with  $k$  layers of weights



## A word on notation



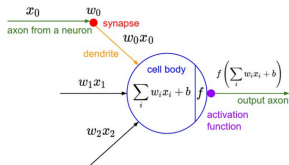
- $k$ -layer NNs: with  $k$  layers of weights
- $k$ -hidden-layer NNs: with  $k$  hidden layers of nodes (i.e.,  $(k + 1)$ -layer NNs)

# First trial

Think of single-output (i.e.,  $R$ ) problems first

## A single neuron

–  $\sigma$  identity or linear:



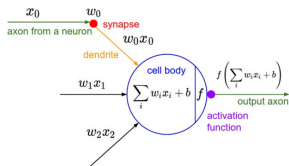
( $f \rightarrow \sigma$ : again, activation  
always as  $\sigma$ )

$$\mathcal{H} : \{x \mapsto \sigma(w^\top x + b)\}$$

# First trial

Think of single-output (i.e.,  $R$ ) problems first

## A single neuron



- $\sigma$  identity or linear: linear functions
- $\sigma$  sign function  $\text{sign}(\mathbf{w}^\top \mathbf{x} + b)$  (perceptron):

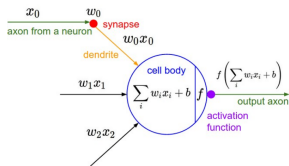
( $f \rightarrow \sigma$ : again, activation  
always as  $\sigma$ )

$$\mathcal{H} : \{\mathbf{x} \mapsto \sigma(\mathbf{w}^\top \mathbf{x} + b)\}$$

# First trial

Think of single-output (i.e.,  $R$ ) problems first

## A single neuron



- $\sigma$  identity or linear: linear functions
- $\sigma$  sign function  $\text{sign}(\mathbf{w}^\top \mathbf{x} + b)$   
(perceptron): 0/1 function with hyperplane threshold
- $\sigma = \frac{1}{1+e^{-z}}$ :

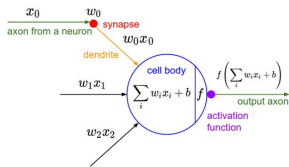
( $f \rightarrow \sigma$ : again, activation  
always as  $\sigma$ )

$$\mathcal{H} : \{\mathbf{x} \mapsto \sigma(\mathbf{w}^\top \mathbf{x} + b)\}$$

# First trial

Think of single-output (i.e.,  $R$ ) problems first

## A single neuron



( $f \rightarrow \sigma$ : again, activation  
always as  $\sigma$ )

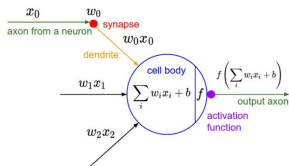
$$\mathcal{H} : \{x \mapsto \sigma(\mathbf{w}^\top x + b)\}$$

- $\sigma$  identity or linear: linear functions
- $\sigma$  sign function  $\text{sign}(\mathbf{w}^\top x + b)$   
(perceptron): 0/1 function with  
hyperplane threshold
- $\sigma = \frac{1}{1+e^{-z}} : \left\{ x \mapsto \frac{1}{1+e^{-(\mathbf{w}^\top x + b)}} \right\}$
- $\sigma = \max(0, z)$  (ReLU):

# First trial

Think of single-output (i.e.,  $R$ ) problems first

## A single neuron



( $f \rightarrow \sigma$ : again, activation  
always as  $\sigma$ )

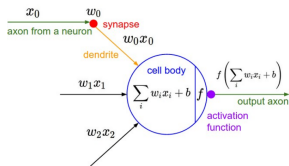
- $\sigma$  identity or linear: linear functions
- $\sigma$  sign function  $\text{sign}(\mathbf{w}^\top \mathbf{x} + b)$   
(perceptron): 0/1 function with  
hyperplane threshold
- $\sigma = \frac{1}{1+e^{-z}}$ :  $\left\{ \mathbf{x} \mapsto \frac{1}{1+e^{-(\mathbf{w}^\top \mathbf{x} + b)}} \right\}$
- $\sigma = \max(0, z)$  (ReLU):  
 $\left\{ \mathbf{x} \mapsto \max(0, \mathbf{w}^\top \mathbf{x} + b) \right\}$

$$\mathcal{H} : \{ \mathbf{x} \mapsto \sigma(\mathbf{w}^\top \mathbf{x} + b) \}$$

# First trial

Think of single-output (i.e.,  $R$ ) problems first

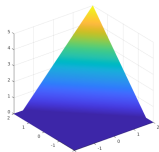
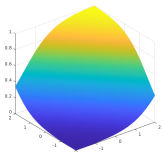
## A single neuron



( $f \rightarrow \sigma$ : again, activation  
always as  $\sigma$ )

$$\mathcal{H} : \{x \mapsto \sigma(w^\top x + b)\}$$

- $\sigma$  identity or linear: linear functions
- $\sigma$  sign function  $\text{sign}(w^\top x + b)$   
(perceptron): 0/1 function with  
hyperplane threshold
- $\sigma = \frac{1}{1+e^{-z}} : \left\{ x \mapsto \frac{1}{1+e^{-(w^\top x + b)}} \right\}$
- $\sigma = \max(0, z)$  (ReLU):  
 $\{x \mapsto \max(0, w^\top x + b)\}$



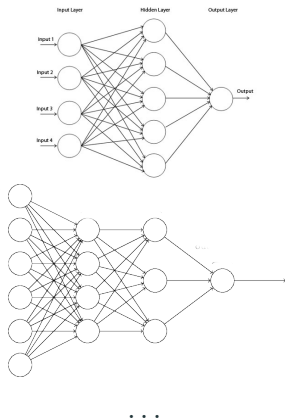
Think of single-output (i.e.,  $R$ ) problems first



## Second trial

Think of single-output (i.e.,  $R$ ) problems first

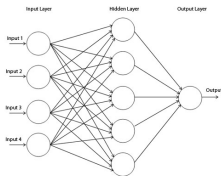
**Add depth!**



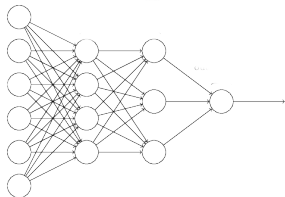
## Second trial

Think of single-output (i.e.,  $R$ ) problems first

**Add depth!**



But make all hidden-nodes activations  
identity or linear

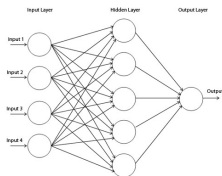


...

## Second trial

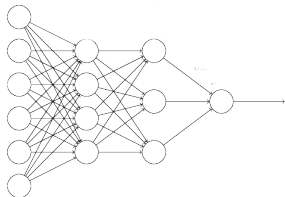
Think of single-output (i.e.,  $R$ ) problems first

**Add depth!**



But make all hidden-nodes activations  
identity or linear

$$\sigma(\mathbf{w}_L^T (\mathbf{W}_{L-1} (\dots (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \dots) \mathbf{b}_{L-1}) + b_L)$$

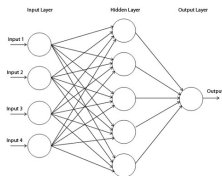


...

## Second trial

Think of single-output (i.e.,  $R$ ) problems first

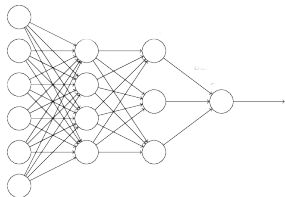
**Add depth!**



But make all hidden-nodes activations  
**identity or linear**

$$\sigma(\mathbf{w}_L^T (\mathbf{W}_{L-1} (\dots (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \dots) \mathbf{b}_{L-1}) + \mathbf{b}_L)$$

**No better than a single neuron!**  
**Why?**



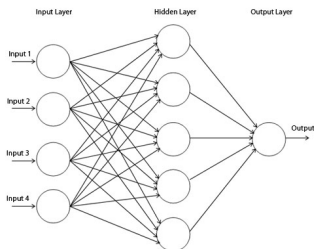
Think of single-output (i.e.,  $R$ ) problems first

**Add both depth & nonlinearity!**

## Third trial

Think of single-output (i.e.,  $R$ ) problems first

**Add both depth & nonlinearity!**

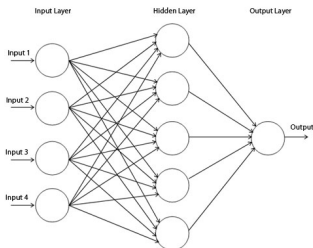


two-layer network, linear  
activation at output

# Third trial

Think of single-output (i.e.,  $R$ ) problems first

**Add both depth & nonlinearity!**



two-layer network, linear  
activation at output

Surprising news:

**universal approximation theorem**

The 2-layer network can approximate **arbitrary continuous** functions **arbitrarily** well, provided that the hidden layer is **sufficiently wide**.

— we don't worry about the capacity

## Universal approximation theorem

### Theorem (UAT, [Cybenko, 1989, Hornik, 1991])

Let  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  be a nonconstant, bounded, and continuous function. Let  $I_m$  denote the  $m$ -dimensional unit hypercube  $[0, 1]^m$ . The space of real-valued continuous functions on  $I_m$  is denoted by  $C(I_m)$ . Then, given any  $\varepsilon > 0$  and any function  $f \in C(I_m)$ , there exist an integer  $N$ , real constants  $v_i, b_i \in \mathbb{R}$  and real vectors  $w_i \in \mathbb{R}^m$  for  $i = 1, \dots, N$ , such that we may define:

$$F(x) = \sum_{i=1}^N v_i \sigma(w_i^T x + b_i)$$

as an approximate realization of the function  $f$ ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all  $x \in I_m$ .



- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)?

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**
- $I_m$  **denote the m-dimensional unit hypercube**  $[0, 1]^m$ :

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**
- $I_m$  **denote the m-dimensional unit hypercube**  $[0, 1]^m$ : this can be replaced by any compact subset of  $\mathbb{R}^m$

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**
- $I_m$  **denote the m-dimensional unit hypercube**  $[0, 1]^m$ : this can be replaced by any compact subset of  $\mathbb{R}^m$
- **there exist an integer**  $N$ :

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**
- $I_m$  **denote the m-dimensional unit hypercube**  $[0, 1]^m$ : this can be replaced by any compact subset of  $\mathbb{R}^m$
- **there exist an integer**  $N$ : but how large  $N$  needs to be?  
(later)

# Thoughts on UAT

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**
- $I_m$  **denote the m-dimensional unit hypercube**  $[0, 1]^m$ : this can be replaced by any compact subset of  $\mathbb{R}^m$
- **there exist an integer  $N$ :** but how large  $N$  needs to be?  
(later)
- **The space of real-valued continuous functions on  $I_m$ :**

# Thoughts on UAT

- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  **be a nonconstant, bounded, and continuous:**  
what about ReLU (leaky ReLU) or sign function (as in perceptron)? We have **theorem(s)**
- $I_m$  **denote the m-dimensional unit hypercube**  $[0, 1]^m$ : this can be replaced by any compact subset of  $\mathbb{R}^m$
- **there exist an integer  $N$ :** but how large  $N$  needs to be?  
(later)
- **The space of real-valued continuous functions on  $I_m$ :**  
two examples to ponder on
  - binary classification
  - learn to solve square root



# Why could UAT hold?

The proof is very technical ... functional analysis

- 1 Riesz Representation: Every linear functional on  $C^0([0, 1]^k)$  is given by

$$f \mapsto \int_{[0,1]^k} f(x) d\mu(x), \quad \mu \in \mathcal{M}$$

where  $\mathcal{M} = \{\text{finite signed regular Borel measures on } [0, 1]^k\}$ .

- 2 **Lemma.** Suppose for each  $\mu \in \mathcal{M}$ , we have

$$\int_{[0,1]^k} \phi(w \cdot x + b) d\mu(x) = 0 \quad \forall w, b \Rightarrow \mu = 0. \quad (0.1)$$

Then  $\text{Nets}_1(\phi)$  is dense in  $C^0([0, 1]^k)$ .

- 3 **Lemma.**  $\phi$  continuous, sigmoidal  $\Rightarrow$  satisfies (0.1).

# Why could UAT hold?

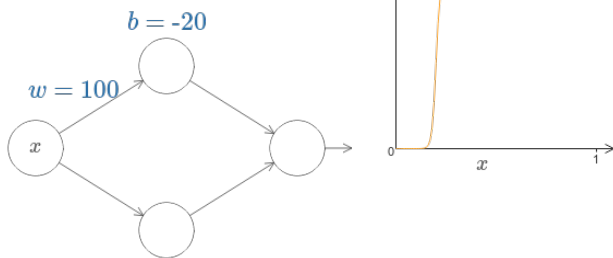
## Visual “proof”

(<http://neuralnetworksanddeeplearning.com/chap4.html>)

Think of  $\mathbb{R} \rightarrow \mathbb{R}$  functions first,  $\sigma = \frac{1}{1+e^{-z}}$

- Step 1: Build “step” functions
- Step 2: Build “bump” functions
- Step 3: Sum up bumps to approximate

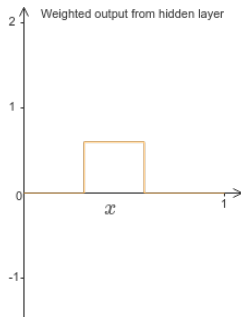
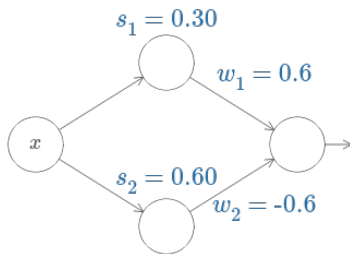
## Step 1: build step functions



$$y = \frac{1}{1 + e^{-(wx+b)}} = \frac{1}{1 + e^{-w(x-b/w)}}$$

- Larger  $w$ , sharper transition
- Transition around  $-b/w$ , written as  $s$

## Step 2: build bump functions

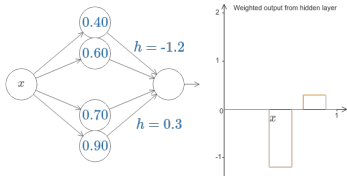


$$0.6 * \text{step}(0.3) - 0.6 * \text{step}(0.6)$$

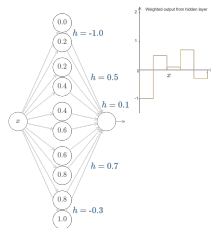
Write  $h$  as the bump height

# Step 3: sum up bumps to approximate

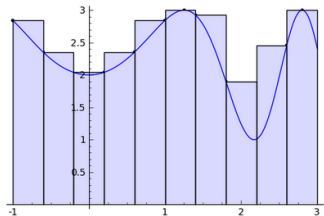
## two bumps



## five bumps



## ultimate idea



familiar?

Recap

Why should we trust NNs?

Suggested reading

## Suggested reading

- Chap 4, Neural Networks and Deep Learning (online book)  
<http://neuralnetworksanddeeplearning.com/chap4.html>

- [Cybenko, 1989] Cybenko, G. (1989). **Approximation by superpositions of a sigmoidal function.** *Mathematics of Control, Signals, and Systems*, 2(4):303–314.
- [Hornik, 1991] Hornik, K. (1991). **Approximation capabilities of multilayer feedforward networks.** *Neural Networks*, 4(2):251–257.