Fundamental Belief: Universal Approximation Theorems

Ju Sun
Computer Science & Engineering
University of Minnesota, Twin Cities

January 29, 2020

Logistics

- 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

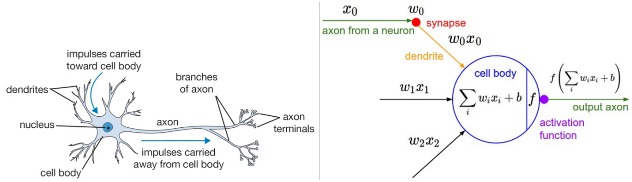
Outline

Recap

Why should we trust NNs?

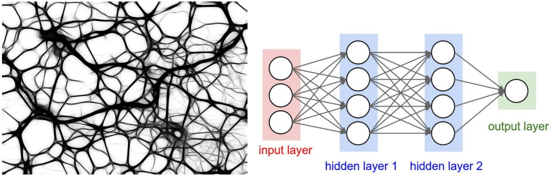
Suggested reading

Recap I



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

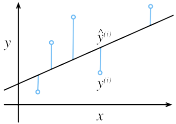
biological neuron vs. **artificial neuron**

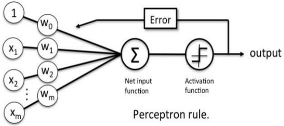


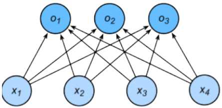
biological NN vs. **artificial NN**

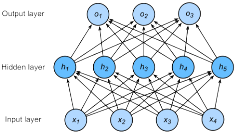
Artificial NN: (over)-simplification on **neuron** & **connection** levels

Zoo of NN models in ML







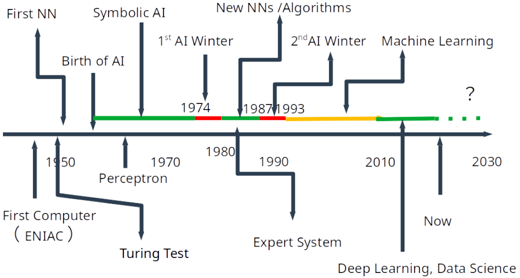


- Linear regression
- Perception and Logistic regression
- Softmax regression
- Multilayer perceptron (feedforward NNs)

Also:

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

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

Outline

Supervised learning

General view:

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

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

NN view:

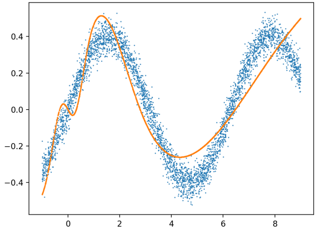
- Gather training data $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)$
- Choose a NN with k neurons, so that there is a group of weights, e.g., $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k)$, to ensure $\boldsymbol{y}_i \approx \{\mathrm{NN}(\boldsymbol{w}_1,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

$$\min_{\boldsymbol{w}'s, b's} \frac{1}{n} \sum_{i=1}^{n} \ell\left[\boldsymbol{y}_i, \{\mathrm{NN}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k)\}(\boldsymbol{x}_i)\right]$$

Why should we trust NNs?

Function approximation

More accurate description of supervised learning



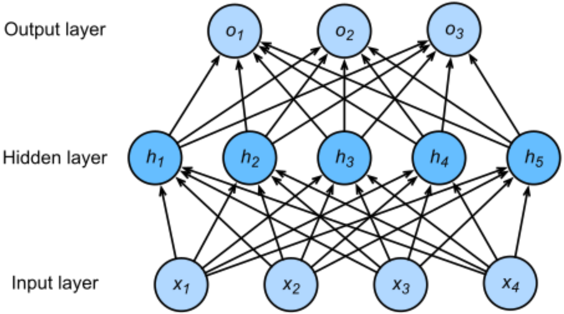
- Underlying true function: f_0
- Training data: $\boldsymbol{y}_i \approx f_0(\boldsymbol{x}_i)$
- Choose a family of functions $\mathcal{H}$, so that $\exists f \in \mathcal{H}$ and

  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.

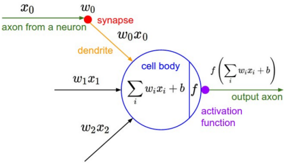
A word on notation



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

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

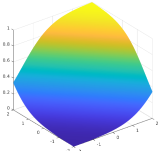
A single neuron

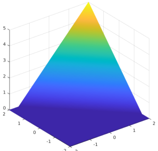


($f \to \sigma$: again, activation always as σ)

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

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

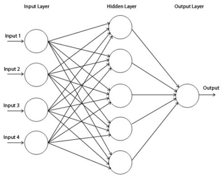


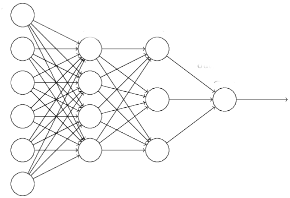


Second trial

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

Add depth!





...

But make all hidden-nodes activations identity or linear

$$\sigma\left(\boldsymbol{w}_L^{\intercal}\left(\boldsymbol{W}_{L-1}\left(\ldots\left(\boldsymbol{W}_1\boldsymbol{x}+\boldsymbol{b}_1\right)+\ldots\right)\boldsymbol{b}_{L-1}\right)+b_L\right)$$

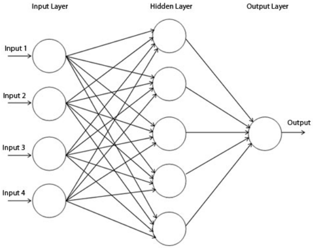
No better than a signle neuron!

Why?

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 $\sigma : \mathbb{R} \to \mathbb{R}$ 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 $i = 1, \ldots, N$, such that we may define:

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

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

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

for all $x \in I_m$.

Thoughts on UAT

- $\sigma : \mathbb{R} \to \mathbb{R}$ **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 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
$$f \mapsto \int_{[0,1]^k} f(x) d\mu(x), \qquad \mu \in \mathcal{M}$$
where $\mathcal{M} = \left\{\text{finite signed regular Borel measures on } [0,1]^k\right\}$.

2. **Lemma.** Suppose for each $\mu \in \mathcal{M}$, we have
$$\int_{[0,1]^k} \phi(w \cdot x + b) d\mu(x) = 0 \quad \forall w, b \quad \Rightarrow \quad \mu = 0. \quad (0.1)$$
Then $\mathrm{Nets}_1(\phi)$ is dense in $C^0([0,1]^k)$.

3. **Lemma.** ϕ continuous, sigmoidal $\quad \Rightarrow \quad$ satisfies (0.1).

Why could UAT hold?

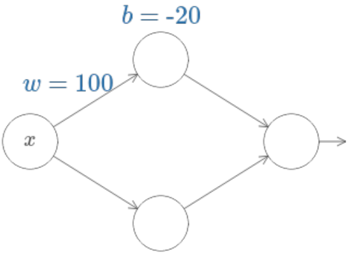
Visual "proof"

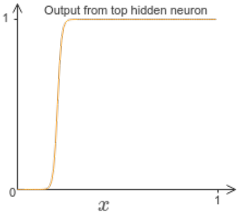
(`http://neuralnetworksanddeeplearning.com/chap4.html`)

Think of $\mathbb{R} \to \mathbb{R}$ 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

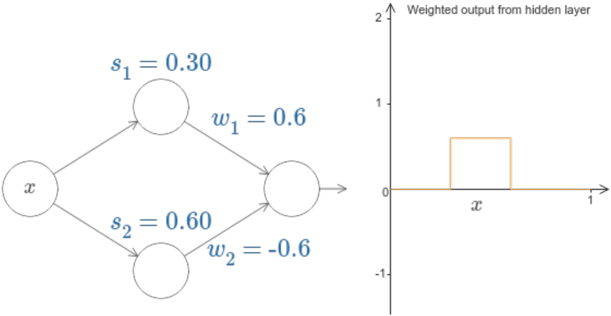




$$y = \frac{1}{1 + e^{-(wx+b)}} = \frac{1}{1 + e^{-w(x-b/w)}}$$

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

Step 2: build bump functions

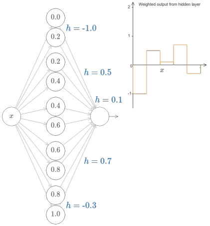


$$0.6 * \mathrm{step}(0.3) - 0.6 * \mathrm{step}\,(0.6)$$

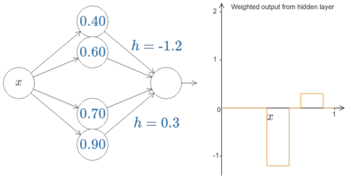
Write h as the bump height

Step 3: sum up bumps to approximate

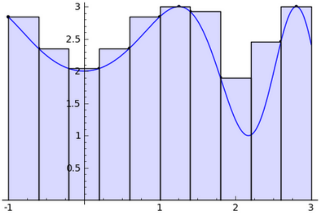
five bumps



two bumps



ultimate idea



familiar?

Outline

Recap

Why should we trust NNs?

Suggested reading

Suggested reading

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

References i

[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.