Introduction

Seminar Series Neural Networks for Finance

Lecture 2

Aim

Neural networks in depth: capacity and optimization

24-02-2022

Balint Negyesi (b.negyesi@tudelft.nl)
Deep versus Shallow slides written by Nikolaj Mücke (nikolaj.mucke@cwi.nl)

Agenda

(Fully-Connected) Feedforward Artificial Neural Networks

- A **fully-connected, feedforward artificial neural network** is simply a **hierarchical composition** of the transformations above, mapping an input $x \in \mathbb{R}^d$ to the output $\mathbf{\Phi} \in \mathbb{R}^q$ whose i'th coordinate reads as follows

$$\Phi_i(x|\theta) := \varphi_i^{(2)} \left(\sum_{j=1}^{p_1} W_{ij}^{(2)} \varphi_j^{(1)} \left(\sum_{k=1}^{d} W_{jk}^{(1)} x_k + b_j^{(1)} \right) + b_i^{(2)} \right) \tag{1}$$

where $\theta := (\theta^{(1)}, \theta^{(2)}) := (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}) \in \mathbb{R}^{p_1 \times (d+1) + q \times (p_1+1)}$

- The total number of parameters in the statistical model is

$$p = p_1 \times (d+1) + q \times (p_1 + 1) \tag{2}$$

Disclaimer

Disclaimer

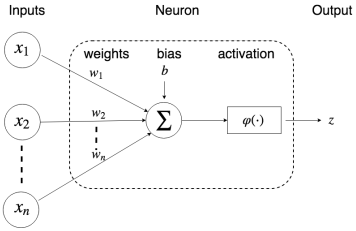
From this lecture on – as is usual in the field of statistics (and also machine learning) – we drop the bold face notation for vectors and matrices (and tensors). Tensor valued entries should be interpreted by context. For example: $x \leftarrow \mathbf{x} \in \mathbb{R}^d$, $\Theta \leftarrow \boldsymbol{\Theta} \in \mathbb{R}^p$, $W^{(l)} \leftarrow \mathbf{W}^{(l)} \in \mathbb{R}^{p_l \times p_{l-1}}$, ...

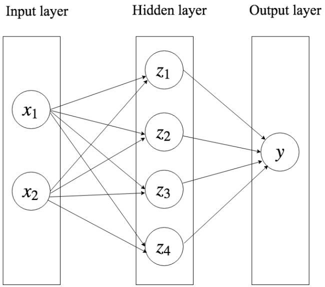
(Fully-Connected) Feedforward Artificial Neural Networks

- Denoting the *element-wise* non-linearities by $\varphi^{(n)}(z^{(n)}) := (\varphi_1^{(n)}(z_1^{(n)}), \dots, \varphi_{p_n}^{(n)}(z_{p_n}^{(n)})) \in \mathbb{R}^{p_n}$, in vector notation this can be written as follows

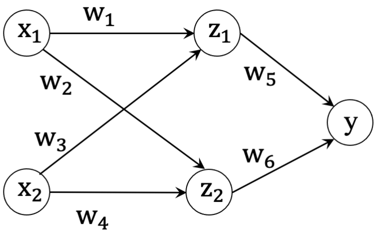
$$\Phi(x|\Theta) := \varphi^{(2)} \circ z^{(2)}(\cdot|\theta^{(2)}) \circ \varphi^{(1)} \circ z^{(1)}(x|\theta^{(1)}) \qquad (3)$$

- FCFF ANNs can be thought of as **directed, acyclic graphs**



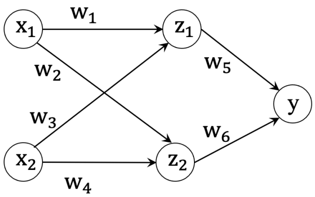


Example: XOR revisited



- Unlike with SLP, adding a *hidden layer*, and parametrizing with ANNs solves the problem
- Let $h_1(z_1)$ and $h_2(z_2)$ be perceptrons with $b_1 = b_2 = 0$
- Fix $w_1 = -1, w_2 = 1, w_3 = 1, w_4 = -1$
- The desired output is obtained by setting $w_5 = w_6 = 1$ and $b_3 = 1$

Example: XOR revisited



- $\widehat{\Theta} = (w_1 = -1, w_2 = 1, w_3 = 1, w_4 = -1, b_1 = 0, b_2 = 0, w_5 = 1, w_6 = 1, b_3 = 1)$ gives and the XOR problem is solved

x_1	x_2	z_1	z_2	y
-1	-1	1	1	1
-1	1	1	-1	-1
1	-1	-1	1	-1
1	1	-1	-1	1

Seminar Series Neural Networks for Finance

Activations: the key feature of non-linearity

Activation Functions

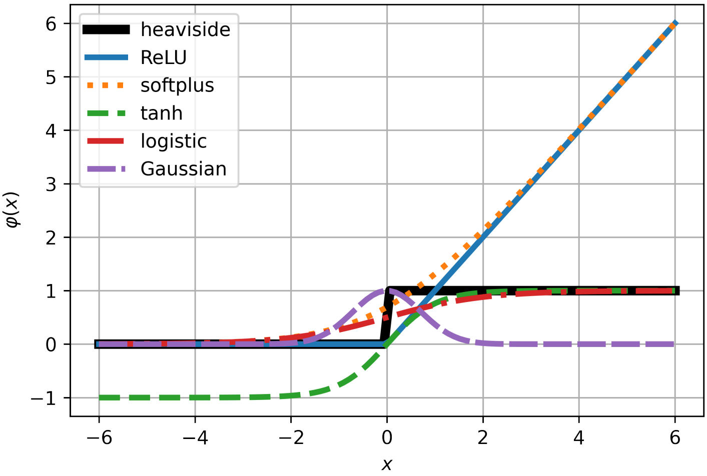
- The key component of *non-linearity* is given by the **activation functions** $\varphi_j^{(n)} : \mathbb{R} \to \mathbb{R}, j = 1, \dots, p_n$
- For SLP we had $\varphi_1^{(1)}(x) = \mathbb{1}_{x-b>0}(x)$
- Most often, application functions within the same layer are chosen the same $\varphi_1^{(n)}(x) = \cdots = \varphi_{p_n}^{(n)}(x)$
- In regression applications the output activations are usually chosen to be the identity function $\varphi^{(2)}(z^{(2)}) = z^{(2)}$
- The parametrization inherits **continuity** and **differentiability** properties from the chosen activations
- The non-linear activation functions are *hyperparameters* of the statistical model, appropriate choices vary depending on the problem

Activation Functions

name	$\varphi(x)$	$\varphi'(x)$	range	continuity
heaviside	$\begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$	$\begin{cases} 0, & x \neq 0 \\ \text{undefined}, & x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
ReLU	$\begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$	$\begin{cases} 0, & x < 0 \\ x, & x > 0 \\ \text{undefined}, & x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian	e^{-x^2}	$-2xe^{-x^2}$	$(0, 1]$	C^∞
tanh	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - \tanh^2(x)$	$(-1, 1)$	C^∞
logistic	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{1+e^{-x}}$	$(0, 1)$	C^∞
softplus	$\log(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
...	...	...	...	...

See wiki

Activation Functions



Layer dependent activations

- For classification problems it is chosen such that $\sum_{k=1}^{q} \varphi_k^{(2)}(z_k^{(2)}) = 1$
- In order to ensure that output *probabilities* indeed sum to one, activations are chosen such that they are not a collection of scalar-valued mappings of neurons but vector-valued functions depending on the whole layer $\varphi^{(n)} : \mathbb{R}^{p_n} \to \mathbb{R}^{p_n}$

name	$\varphi_i(x)$	$\partial_j \varphi_i(x)$	range	cont.
softmax	$\frac{e^{x_i}}{\sum_{j=1}^{p_n} e^{x_j}}$	$\varphi_i(x)(\delta_{ij} - \varphi_j(x))$	$(0,1)$	C^∞
maxout	$\max_{i=1,\dots,p_n} x_i$	$\begin{cases} 1, & j = \arg\max_i x_i \\ 0, & j \neq \arg\max_i x_i \end{cases}$	$(-\infty,\infty)$	C^0
...	...	...	...	...

Seminar Series Neural Networks for Finance

Deep Neural Networks

Fully-Connected Feedforward Deep Neural Networks

- So far: neural network = input + hidden + output layer
- These are called **shallow** feedforward ANNs

$$\Phi(x|\Theta) := \varphi^{(2)} \circ z^{(2)}(\cdot|\theta^{(2)}) \circ \varphi^{(1)} \circ z^{(1)}(x|\theta^{(1)}) \quad (4)$$

- One can extend the model to allow for L-many hidden layers in the composition, with corresponding widths $p_l, l = 1 \ldots, L$
- The resulting, **deep neural network** mapping reads as follows

$$\Psi(x|\Theta) := \varphi^{(L+1)} \circ z^{(L+1)}(\cdot|\theta^{(L+1)}) \circ \cdots \circ \varphi^{(1)} \circ z^{(1)}(x|\theta^{(1)}), \quad (5)$$

where $\Theta := (\theta^{(1)}, \ldots, \theta^{(L+1)}) \in \mathbb{R}^p$

- The total number of parameters in the statistical model rapidly increases

$$p = d \times (p_1 + 1) + \sum_{l=1}^{L-1} p_l \times (p_{l+1} + 1) + p_L \times (q + 1) \quad (6)$$

Fully-Connected Feedforward Deep Neural Networks

$$\Psi(x|\Theta) := \varphi^{(L+1)} \circ z^{(L+1)}(\cdot|\theta^{(L+1)}) \circ \cdots \circ \varphi^{(1)} \circ z^{(1)}(x|\theta^{(1)}), \quad (7)$$

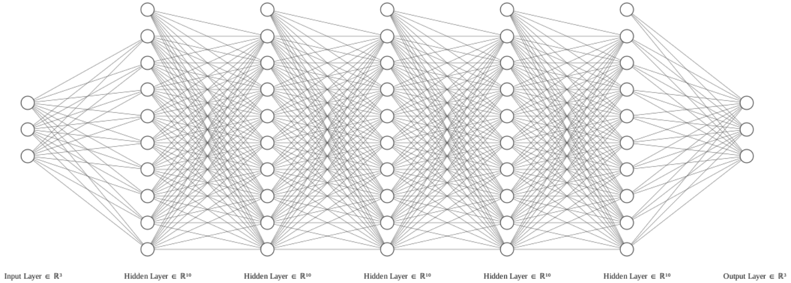


Figure: Illustration DNN, source: NN-SVG

$$p = d \times (p_1 + 1) + \sum_{l=1}^{L-1} p_l \times (p_{l+1} + 1) + p_L \times (q + 1)$$

Seminar Series Neural Networks for Finance

Universal Approximation Theorem

Most commonly used carpet

Topic

- We have seen: (deep) neural networks provide *one way* to parametrize non-linear mappings
- We have not seen: **is that a "good" way? What class of functions can actually be approximated by them?**

A classical analogy

Theorem (Weierstrass approximation theorem)

Let $f \in C([a,b];\mathbb{R})$. For any $\epsilon > 0$ there exists a polynomial $p : x \mapsto a_0 + a_1 x + a_2 x^2 + \dots$ such that $\sup_{x\in[a,b]} |f(x) - p(x)| < \epsilon$.

- **Weierstrass approximation theorem**: any continuous function can be uniformly approximated by a polynomial functions over closed intervals, with arbitrary accuracy
- Terminology the set of polynomials is **dense** in $C([a,b];\mathbb{R})$
- Conclusion: polynomials may be a good parametrization for continuous functions
- Notice: **not implementable**
  1. No bound on the order of the polynomial
  2. No way to find the *"right"* polynomial within the class

Universal Approximation Theorem

- In order to show that neural networks are *"good"* family of statistical models, we need a similar result $\rightarrow$ **Universal Approximation Theorem (UAT)**
- First: Weierstrass for shallow neural networks
- Definition: an activation $\varphi : \mathbb{R} \rightarrow \mathbb{R}$ is called **sigmoidal**, if it is continuous and $\lim_{x \rightarrow -\infty} \varphi(x) = 0, \lim_{x \rightarrow \infty} \varphi(x) = 1$

Theorem (UAT – continuous functions, Cybenko, 1989)

Let $\mathcal{G}_\varphi := \{\Phi(x|\Theta) : \Theta \in \mathbb{R}^p, p \in \mathbb{N}, \varphi \text{ fixed, sigmoidal}\}$ *be the family of shallow neural networks of the form* (3) *with* $d = q = 1$. *Then, for any* $f \in C([0,1];\mathbb{R})$ *and* $\epsilon > 0$ *there exists a shallow neural network* $\Phi(\cdot|\Theta^*) \in \mathcal{G}_\varphi$ *such that*

$$\sup_{x \in [0,1]} |\Phi(x|\Theta^*) - f(x)| < \epsilon.$$

Restrictions and extensions

- Similar restrictions as in Weierstrass: *arbitrary width*, no insight on Θ^*
- Natural extension from shallow to deep neural networks of the form (7)
- Several results exist, very actively researched area since the early 90s
  1. sigmoidal $\rightarrow$ non polynomial
  2. $[a, b] \rightarrow \mathbb{R} \longrightarrow$ compact subsets in general vector spaces to other vector spaces
  3. dense in $C([a, b]; \mathbb{R}) \rightarrow$ dense in the space of Lebesgue integrable functions (L^p spaces)
  4. approximating functions with arbitrary accuracy $\rightarrow$ approximating functions and **their derivatives** with arbitrary accuracy (Sobolev spaces) – see Hornik et al., 1990
  5. arbitrary width/depth $\rightarrow$ fixed width/depth with dominated error term (**curse of dimensionality**)
  6. ...

Seminar Series Neural Networks for Finance

Stochastic Gradient Descent

Optimization

Question

- UAT $\rightarrow$ neural networks may be an appropriate statistical model for a wide range of phenomena
- UAT: in the space of neural networks there lives a neural network $\Phi(x|\Theta^*)$ which approximates a given function $f(x)$ with arbitrary accuracy
- We do not know: **how do we find Θ^*, or at least a good approximation of it?**

Iterative schemes

- Recall: thanks to the **convexity** of $\beta \mapsto \|y - X\beta\|^2$, OLS has a closed form expression mapping measurements X, y to the **unique, global, minimizer** of MSE

$$\beta^* = (X^T X)^{-1} X^T y \tag{8}$$

- In neural network regression the loss is no longer convex $L(\Theta|D = (x, y)) = \|y - \Phi(x|\Theta)\|^2$ due to the non-linearity of $\Phi(x|\Theta)$ in Θ
- No closed-form expression for **a** minimizer $\implies$ **iterative schemes** needed
- **No unique solution** to $\nabla_\Theta L(\Theta|D) = 0$
- A solution to $\nabla_\Theta L(\Theta|D) = 0$ may not be an extremum at all (**saddle points**)
- **local minima** may not be global $\implies$ solving $\nabla_\Theta L(\Theta|D) = 0$ does not guarantee global minimum

Iterative schemes – general formulation

- An iterative scheme is a *sequence of approximations* Θ^n such that the resulting approximations $\Phi^n := \Phi(\cdot|\Theta^n)$ converge to a desired solution H in an appropriate sense (e.g. MSE)
- An iterative scheme needs an appropriate *update/iterative rule* which determines Θ^{n+1} given Θ^n
- How do we define a sensible update rule? $\longrightarrow$ **(Stochastic) Gradient Descent ((S)GD)**

Gradient Descent – intuition

- Assume we want to minimize some $f : \mathbb{R}^p \to \mathbb{R}_+$ non-linear real-valued function with some iterative scheme. Our current guess for the minimum is x^k. What should our next guess be? $\longrightarrow$ **steepest gradient**
- It is always nice to gather an intuition in one dimension

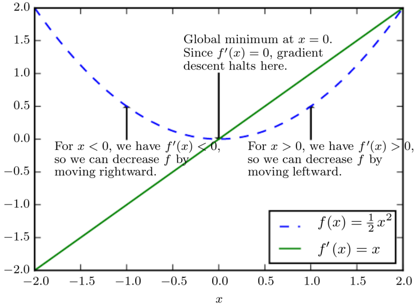


Figure: Goodfellow et al.: Deep Learning (2016), MIT Press [Figure 4.1, pg. 81]

Gradient Descent – general

- Back to the general setting: given a point $x^k \in \mathbb{R}^p$ we want to find the **direction** $e \in \mathbb{R}^p$, $\|e\| = 1$ in which the value of f decreases the fastest
- Formally we are concerned with the following minimization problem

$$\begin{aligned}\operatorname*{arg\,min}_{e\in\mathbb{R}^p,\|e\|=1} \left\langle e \middle| \nabla_x f(x^k) \right\rangle &= \operatorname*{arg\,min}_{e\in\mathbb{R}^p,\|e\|=1} \|e\| \left\| \nabla_x f(x^k) \right\| \cos\left(\vartheta_{e,x^k}\right) \\ &= \operatorname*{arg\,min}_{e\in\mathbb{R}^p,\|e\|=1} \left\| \nabla_x f(x^k) \right\| \cos\left(\vartheta_{e,x^k}\right) \\ &= \operatorname*{arg\,min}_{e\in\mathbb{R}^p,\|e\|=1} \cos\left(\vartheta_{e,x^k}\right),\end{aligned}$$

  where ϑ_{e,x^k} is the angle between e and $\nabla_x f(x^k)$
- This is minimized when $\vartheta_{e,x^k} = \pi$ leading to $e = -\nabla_x f(x^k)/\left\|\nabla_x f(x^k)\right\| \implies$ **steepest descent** or **gradient descent**

Gradient Descent

Gradient Descent (GD)

The iterative scheme of a gradient descent optimization then reads as

$$x^{k+1} = x_k - \eta^k \nabla_x f(x^k), \tag{9}$$

where the parameter η^k is called a **learning rate** and determines the size of each iteration step

Several strategies for the choice of $k \to \eta^k$: constant, adaptive (decay, validation loss linked, etc.), line search, . . .

Back to machine learning (statistics)

Our goal is to minimize a performance measure (loss)

$$L(\Theta) \coloneqq \mathbb{E}_{(x,y)\sim\mathbb{P}_{\text{data}}}\left[d(\Phi(x|\Theta), y)\right] \tag{10}$$

indirectly by minimizing an empirical loss defined by

$$L(\Theta|D) \coloneqq \frac{1}{M}\sum_{m=1}^{M} d(\Phi(x_m|\Theta), y_m) \tag{11}$$

given a finite number of M measurements drawn from the same data generating distribution

capacity, generalization, overfitting, ...

From Gradient Descent to Stochastic Gradient Descent

- The GD iteration in (9) on (11) would read as

$$\Theta^{k+1} = \Theta^k - \eta^k \frac{1}{M} \sum_{m=1}^{M} \nabla_\Theta d(\Phi(x_m|\Theta^k), y_m) \qquad (12)$$

- This iteration scheme is sometimes called batch gradient descent *(personal opinion: wrongly)*
- Drawbacks: computationally intensive (gradient for every sample), large memory requirements, cannot update model on the fly, ...
- Idea: random partition of the dataset and obtain gradient steps on these random subsets $\longrightarrow$ **Stochastic Gradient Descent (SGD)**

Stochastic Gradient Descent

The steepest descent is decomposed into steps. Choose $B < M$. Choose a random permutation of the dataset $\pi : \{1, \dots, M\} \to \{1, \dots, M\}$.

1. Random permutation of the dataset into π^k
2. Loop over each resulting subset $i = 1, \dots, \lceil M/B \rceil$ of size B: $\{x_{\pi_m}, y_{\pi_m}\}_{m=(i-1)B}^{\min(iB,M)}$
3. Update the parameters over each subset $i = 1, \dots, \lceil M/B \rceil$

$$\Theta^{k,i+1} = \Theta^{k,i} - \eta^k \frac{1}{B} \sum_{m=(i-1)B}^{\min(iB,M)} d(\Phi(x_m|\Theta^{k,i}), y_m) \qquad (13)$$

4. The $k+1$'th iteration step receives $\Theta^{k+1} = \Theta^{k+1,1} := \Theta^{k,\lceil M/B \rceil}$

Terminology: k: **epoch**, i: **batch**, B: **batch size**

Batch size

- $B = 1$ is often called **online learning**
- Larger Bs provide more accurate estimates of the gradient $\nabla_{\Theta} L(\Theta|D)$ but with diminishing returns
- Small batches often have a *regularization effect* and thus generalize better
- Small batches require more iterations per epoch and are thus slower
- Eventually: trial and error...

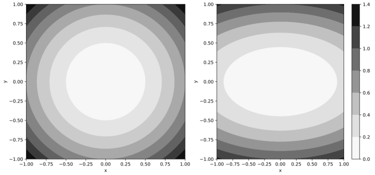
Initialization and convergence criteria

Any implementable iterative optimization scheme requires two more ingredients

- An **initialization** Θ^0 determining the sequence $\Theta^k \leftarrow \Theta^{k-1}, k \geq 1$
  - biases are usually initialized with an arbitrarily chosen constant value – most often 0
  - weights are initialized randomly, several options exist: $W_{ij}^{(l)} \sim U\left(-\sqrt{\frac{6}{p_{l-1}+p_l}}, \sqrt{\frac{6}{p_{l-1}+p_l}}\right)$, $W_{ij}^{(l)} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{p_{l-1}+p_l}}\right)$
  - usually designed in a way to fix the variances of each layer's output activations, but also the variances of the gradients
  - crucial step for any successful iteration (no local optima, ...), actively researched topic
- A **convergence criterion** which stops the iteration at a given step K concluding in approximations $\widehat{\Theta} := \Theta^K$ – based on validation loss, maximum iteration number, ...

Learning rate

- The size of an iteration step in (9)–(13) in the *steepest direction* depends on the **learning rate**
- Closer to the minimum, gradients tend to decrease; smaller steps can thus lead to a better approximation of the minimum
- Several strategies $k \to \eta^k$: constant, adaptive (decay, validation loss linked, etc.), line search, . . .
- Eventually: trial and error
- **Momentum** based estimates: momentum SGD, Nesterov, . . .
- Alternative **adaptive** optimization methods: Adam, AdaGrad, RMSProp, . . .



Seminar Series Neural Networks for Finance

Automatic differentiation: descending into a computer

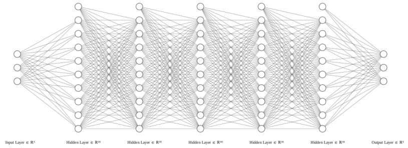
Topic

- We have seen: gradient based iterative optimization methods to minimize empirical loss
- However, they by definition rely on being able to calculate the **gradient** of the loss with respect to the parameters in the model $\nabla_\Theta L(\Theta|D)$
- If $d(f,g)$ in 11 is the mean-squared error we, e.g., have $\nabla_\Theta L(\Theta|D) = 2(\nabla_\Theta \Phi(x|\Theta))^T(\Phi(x|\Theta) - y)$
- $\Theta \in \mathbb{R}^p$ is a very high-dimensional vector (huge parameter spaces) – (6)

Question

How can such a large dimensional gradient can be computed in a **precise**, **efficient**, and **robust** way?
Finite differences: high errors, slow, bad scaling; symbolic expressions: memory consumption, very long and redundant expressions $\longrightarrow$ **automatic differentiation**

Recall: neural networks' hierarchical structure



- Hierarchical sequence of compositions of the form (7)
- Complete parameter space $\Theta := (W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}) \in \mathbb{R}^p$ with $W^{(l)} \in \mathbb{R}^{p_l \times p_{l-1}}$, $b^{(l)} \in \mathbb{R}^{p_l}$ and $p_0 = d$
- Activations by non-linear mappings $\varphi^{(l)}$ and their affine combinations

$$z^{(l+1)} = W^{(l+1)} z^{(l)} + b^{(l+1)}, \quad a^{(l+1)} = \varphi^{(l+1)}(z^{(l+1)}),$$

$l = 0, \ldots, L$

Computational graphs

- This structure

$$x \overset{W^{(1)},b^{(1)}}{\longmapsto} z^{(1)} \overset{\varphi^{(1)}}{\longmapsto} a^{(1)} \mapsto \dots \overset{\varphi^{(L+1)}}{\longmapsto} a^{(L+1)}$$

  can be represented as **directed, acyclic graph** $\longrightarrow$ **computational graph**
- Each node in the graph corresponds to an operation and *data flows through the vertices*
- An operation is characterized by three characteristics: a compute function which determines the node's output; set of input nodes; and set of output nodes
- Computational graphs: complex calculations decomposed into a sequence of elementary operations; far more general than machine learning

Forward propagation

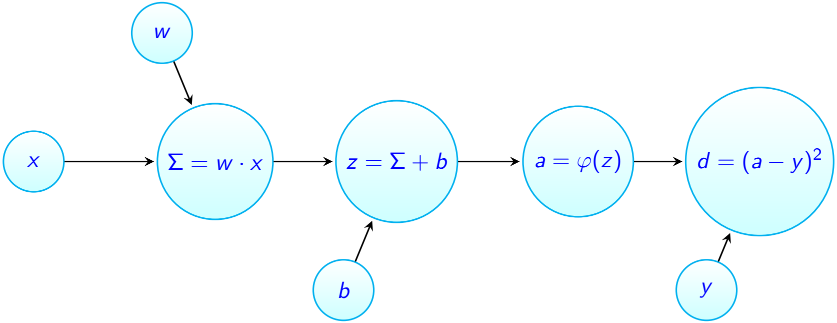


Figure: Forward Propagation

Gradients

- The input nodes of $z^{(l)}$ are $a^{(l-1)}, W^{(l)}, b^{(l)}$

$$z_i^{(l)} = \sum_{j=1}^{p_{l-1}} W_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)} \tag{14}$$

- With respect to these inputs are easily obtained

$$\frac{\partial z_i^{(l)}}{\partial a_j^{(l-1)}} = W_{ij}^{(l)}, \quad \frac{\partial z_i^{(l)}}{\partial b_j^{(l)}} = \delta_{ij}, \quad \frac{\partial z_i^{(l)}}{\partial W_{kj}^{(l)}} = \delta_{ki} a_j^{(l-1)} \tag{15}$$

- The input nodes of $a^{(l)}$ are in $z^{(l)}$: $a_i^{(l)} = \varphi_i^{(l)}(z^{(l)})$

$$\frac{\partial a_i^{(l)}}{\partial z_j^{(l)}} = (\nabla_{z^{(l)}} \varphi^{(l)}(z^{(l)}))_{ij} \tag{16}$$

Derivation

With Jacobian matrices

$$\text{Forward}\begin{cases} a^{(0)} & = x, \\ z^{(1)} & = W^{(1)} \cdot a^{(0)} + \mathbf{b}^{(1)}, \\ a^{(1)} & = \varphi^{(1)}(z^{(1)}), \\ \quad \dots & \\ z(L+1) & = W^{(L+1)} \cdot a^{(L)} + \mathbf{b}^{(L+1)}, \\ a^{(L+1)} & = \varphi^{(L+1)}(z^{(L+1)}) \end{cases} \tag{17}$$

$$\text{Backward}\begin{cases} \nabla_x a^{(0)} & = I_d, \\ \nabla_{a^{(0)}} z^{(1)} & = W^{(1)}, \\ \nabla_{z^{(1)}} a^{(1)} & = \nabla_{z^{(1)}} \varphi^{(1)}(z^{(1)}), \\ \quad \dots & \\ \nabla_{a^{(L)}} z^{(L+1)} & = W^{(L+1)}, \\ \nabla_{z^{(L+1)}} a^{(L+1)} & = \nabla_{z^{(L+1)}} \varphi^{(L+1)}(z^{(L+1)}) \end{cases}$$

Back-propagation

- Suppose there is one node in the last layer, matrix $W^{(L+1)} = [w_{1j}^{(L+1)}, j = 1, \dots, p_L]$ is a vector then
- We compute the sensitivity of the mean-squared distance function d with respect to weight $w_{1j}^{(L+1)}$ by the **chain rule**

$$\begin{aligned}
\frac{\partial d}{\partial W_{1j}^{(L+1)}} &= \frac{\partial d}{\partial a_1^{(L+1)}} \frac{\partial a_1^{(L+1)}}{\partial W_{1j}^{(L+1)}} \\
&= \frac{\partial d}{\partial a_1^{(L+1)}} \frac{\partial a_1^{(L+1)}}{\partial z_1^{(L+1)}} \frac{\partial z_1^{(L+1)}}{\partial W_{1j}^{(L+1)}} \\
&= 2(a_1^{(L+1)} - y)\varphi^{(L+1)'}(z_1^{(L+1)}) a_j^{(L)},
\end{aligned}$$

from Eq. (15)

Gradient

- Similarly, the **chain rule** gives an expression for the derivative of L wrt the inputs x

$$
\begin{aligned}
(\nabla_x d)^T &= (\nabla_{a^{(L+1)}} d)^T \nabla_{z^{(L+1)}} a^{(L+1)} \nabla_{a^{(L)}} z^{(L+1)} \ldots \\
&\quad \ldots \nabla_{z^{(1)}} a^{(1)} \nabla_{a^{(0)}} z^{(1)} \nabla_x a^{(0)} \\
&= (\nabla_{a^{(L+1)}} d)^T \nabla_{z^{(L+1)}} \varphi^{(L+1)}(z^{(L+1)}) W^{(L+1)} \ldots \\
&\quad \ldots \nabla_{z^{(1)}} \varphi^{(1)}(z^{(1)}) W^{(1)} I
\end{aligned}
\tag{18}
$$

  where each element is a Jacobian matrix
- **Three types of terms:**, the gradient of the distance with respect to the outputs $\nabla_{a^{(L+1)}} d$; Jacobian matrix of each layer's output activations $\nabla_{z^{(l)}} \varphi^{(l)}(z^{(l)})$; and the derivative of the affine transformation function with respect to input activations $\nabla_{a^{(l-1)}} z^{(l)}$

Backpropagation

To derive the closed-form expressions of backward propagation, we define the auxiliary error vector

$$[\delta^{(l)}]^T := [\nabla_{a^{(L+1)}} d]^T \nabla_{z^{(L+1)}} \varphi^{(L+1)}(z^{(L+1)}) W^{(L+1)} \dots \\ \dots W^{(l+1)} \nabla_{z^{(l)}} \varphi^{(l)}(z^{(l)}), \tag{19}$$

which is a vector of length $\mathbb{R}^{p_l}$

Backpropagation

The backpropagation scheme, implementing the derivative of the loss function is then defined by the following recursive expressions

$$\begin{aligned} \nabla_{b^{(l)}} L &= \delta^{(l)}, \\ \nabla_{W^{(l)}} L &= \delta^{(l)} (z^{(l-1)})^T, \\ [\delta^{(l)}]^T &= [\delta^{(l+1)}]^T (W^{(l+1)}) \nabla_{z^{(l)}} \varphi^{(l)}(z^{(l)}). \end{aligned} \tag{20}$$

Backward propagation

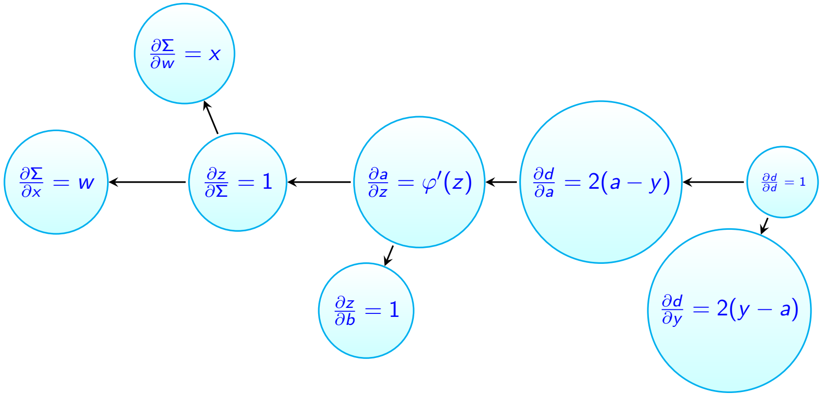

$\frac{\partial \Sigma}{\partial w} = x$
$\frac{\partial \Sigma}{\partial x} = w$
$\frac{\partial z}{\partial \Sigma} = 1$
$\frac{\partial a}{\partial z} = \varphi'(z)$
$\frac{\partial d}{\partial a} = 2(a - y)$
$\frac{\partial d}{\partial d} = 1$
$\frac{\partial z}{\partial b} = 1$
$\frac{\partial d}{\partial y} = 2(y - a)$

Figure: Backpropagation

Seminar Series Neural Networks for Finance

Gradient based optimization problems

Challenges: local minima, saddle points, etc.

Together with backpropagation/automatic differentiation, SGD is a fully-implementable iterative optimization algorithm – their combination is called **training**
However, not without challenges

- local minima
- saddle points
- cliffs and plateaus: exploding and vanishing gradients
- ill-conditioned problems
- inaccurate gradient estimates – over batches
- regularization: **overfitting**
- . . .

Local minima

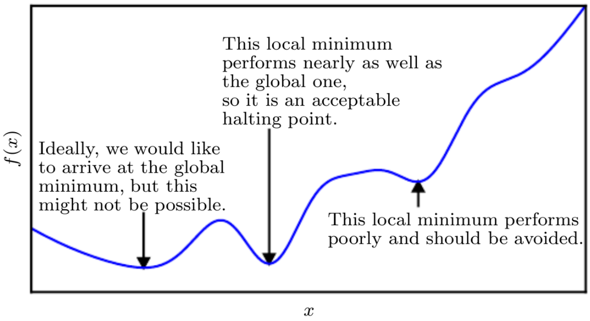


Figure: Goodfellow et al.: Deep Learning (2016), MIT Press [Figure 4.3, pg. 83]

Saddle points

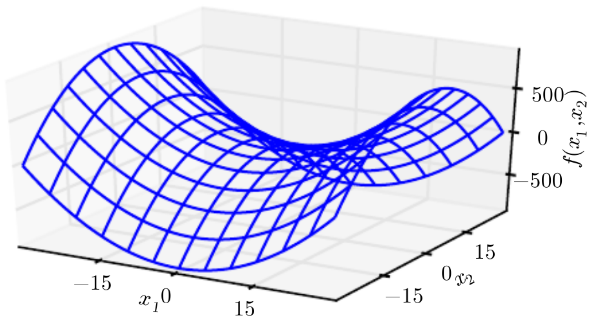


Figure: Goodfellow et al.: Deep Learning (2016), MIT Press [Figure 4.5, pg. 88]

Vanishing gradients, plateaus, cliffs, etc.

- The gradient in the current layer is equal to a product of terms of all the later layers
- Vanishing gradient problem when one term is close to zero; gradient explosion when the product approaches infinity (numerical instabilities, rounding errors)

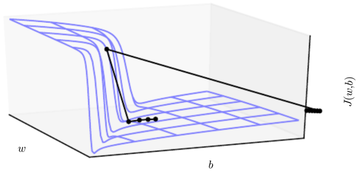


Figure: Goodfellow et al.: Deep Learning (2016), MIT Press [Figure 8.3, pg. 285]

Remedies: **ReLU** – unbounded from above; renormalization; gradient clipping, etc.

Advanced training techniques

Without the sake of completeness

- Input normalization
- **Batch normalization**: reparametrization, shifts and renormalizes activations within a layer across batches, *parametrizes* appropriate the appropriate means and variances
- **Dropout** is technique, which randomly turns off certain neurons (i.e., the gradient becomes zero for those neurons) to force the ANN to learn more features rather than saturate
- Constrained optimization: regularization with $L^1 - L^2$ penalties, etc.
- ...

Curvature, adaptive and second-order optimization

- Momentum: descent modeled as a particle's momentum with unit mass; previous updates' gradients decay with some hyperparameter α

$$\begin{aligned} v^{k+1} &= \alpha v^k - \eta^k \nabla_\Theta L(\Theta^k | D) \\ \Theta^{k+1} &= \Theta^k + v^{k+1} \end{aligned} \tag{21}$$

- Second-order optimization schemes: **Newton's method** to *approximate* the root of a multivariate function $g : \mathbb{R}^n \to \mathbb{R}^n$ by

$$z^0 \in \mathbb{R}^n, \quad z^{k+1} = z^k - [\nabla_z g(z^k)]^{-1} g(z^k), \tag{22}$$

  where $\nabla_z g(z)$ is the Jacobian matrix of the vector-valued mapping g defined by $[\nabla_z g(z)]_{ij} := \partial_{z_j} g_i(z)$

Curvature, adaptive and second-order optimization

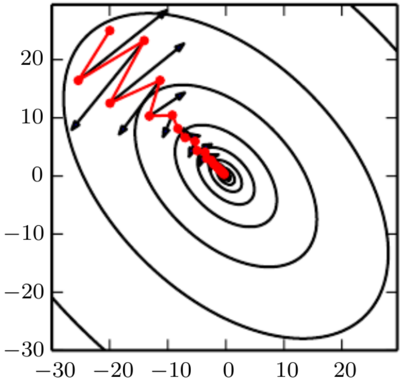


Figure: Goodfellow et al.: Deep Learning (2016), MIT Press [Figure 8.5, pg. 293]

Seminar Series Neural Networks for Finance

Deep versus Shallow

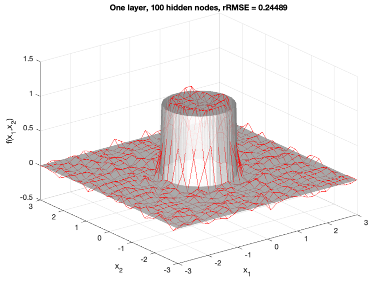
Deep versus Shallow

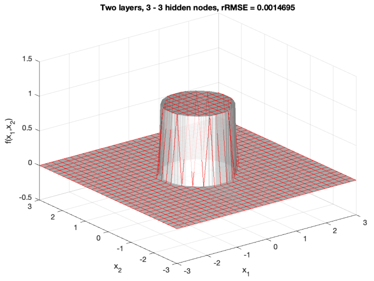
Questions

- When does it make sense to go deep?
- Why does it make sense to go deep?

- Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., & Liao, Q. (2017). Why and when can deep – but not shallow – networks avoid the curse of dimensionality: A review. *arXiv:1611.00740 [cs]*
- Mhaskar, H., Liao, Q., & Poggio, T. (2017). When and why are deep networks better than shallow ones? [Number: 1]. *Proceedings of the AAAI Conference on Artificial Intelligence, 31*(1)
- Mhaskar, H., & Poggio, T. (2016). Deep vs. shallow networks : An approximation theory perspective. *arXiv:1608.03287 [cs, math]*

Empirical difference





RMSE $= 0.2449$
1 hidden layer with 302 neurons
Total number of neurons: 302

RMSE $= 0.0014$
3 hidden layers with 3 neurons
Total number of neurons: 25

Degree of Approximation

- **Complexity**: number of parameters $\sim$ of neurons/units in a network
- Let $\mathcal{V}_N := \{\Phi(\cdot|\Theta) : \Theta \in \mathbb{R}^N\} \subseteq \mathcal{G}$ be the set of networks with complexity N, $\mathcal{V}_N \subseteq \mathcal{V}_{N+1}$
- **Degree of Approximation** is defined by

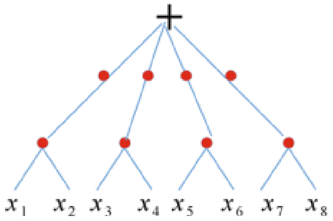
$$\mathrm{dist}(f, \mathcal{V}_N) := \inf_{\Phi \in \mathcal{V}_N} ||f - \Phi||_X,$$

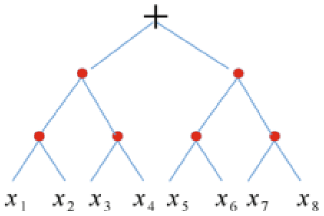
  for $f \in \mathcal{X}$
- $\mathrm{dist}(f, \mathcal{V}_N) = \mathcal{O}(N^{-\gamma}) \implies$ a network with complexity $N = \mathcal{O}(\epsilon^{-\frac{1}{\gamma}})$ is sufficient to guarantee accuracy at least ϵ

Compositional Functions

We will consider **hierarchical compositions** of functions, such as

$$\begin{aligned} f(x_1,\ldots,x_8) &= h_3\Big(\quad h_{21}(h_{11}(x_1,x_2),h_{12}(x_3,x_4)), \\ & \qquad\quad h_{22}(h_{13}(x_5,x_6),h_{14}(x_7,x_8))\Big) \end{aligned}$$





Compositional Functions' Class

- Let $I^n = [-1,1]^n$ and $C(I^n)$ be the space of continuous functions on I^n with norm

$$||f|| = \max_{x \in I^n} |f(x)|$$

- Then consider the subspace $\mathcal{W}_r^n \subset C(I^n)$ consisting of r times continuously differentiable functions on I^n, where

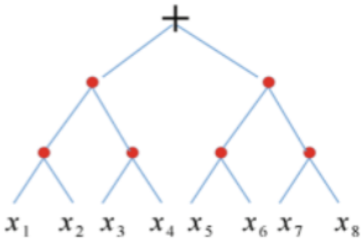
$$||f|| + \sum_{1 \leq |k|_1 \leq r} ||D^k f|| \leq 1,$$

where k is a multiindex and D^k denotes the partial derivative indicated by k, and $|k|_1$ is the sum of the multiindex

Compositional Functions - Function Class

- Let $\mathcal{W}_r^{n,2}$ be the class of functions with a hierarchical structure with each constituent function, $h_{ij} \in \mathcal{W}_r^2$
  - Binary tree functions
- Note $\mathcal{W}_r^{n,2} \subset \mathcal{W}_r^n$
- Example of a $\mathcal{W}_r^{4,2}$ function:

$$f(x_1, x_2, x_3, x_4) = \underbrace{h_2}_{\in \mathcal{W}_r^2}(\underbrace{h_{11}}_{\in \mathcal{W}_r^2}(x_1, x_2), \underbrace{h_{12}}_{\in \mathcal{W}_r^2}(x_3, x_4))$$



Shallow Networks

1. Shallow network: One hidden layer
   - Let $\mathcal{S}_{N,n}$ be the class of shallow networks with N neurons and n-dimensional input of the form

$$x \mapsto \sum_{k=1}^{N} a_k \sigma(w_k \cdot x + b_k), \quad w_k \in \mathbb{R}^n, \quad a_k, b_k \in \mathbb{R},$$

   - where $\sigma : \mathbb{R} \to \mathbb{R}$ is the (nonlinear) activation function.
   - Number of trainable parameters: $p = (n+2)N \sim N$
2. Deep neural networks: at least two hidden layers
   - Let $\mathcal{D}_{N,2}$ be the class of Deep networks that take n-dimensional input with N neurons and binary tree structure
   - Each constituent function in the deep network is a $\mathcal{S}_{Q,2}$ network
   - Complexity: $(n-1)Q = N$

Comparison theorems

Theorem – Shallow neural networks

Under some assumptions on the non-linear activations, for any $f \in \mathcal{W}_r^n$ the complexity of shallow networks that provide accuracy at least ϵ is $N = \mathcal{O}(\epsilon^{-n/r})$ and is the best possible $\implies$ complexity increases **exponentially** with the input dimension - **curse of dimensionality**

Theorem – Deep neural networks

Under some assumptions on the non-linear activations, for any $f \in \mathcal{W}_r^{n,2}$ the complexity of a deep network (with the same compositional achitecture) that provide accuracy at least ϵ is $N = \mathcal{O}((n-1)\epsilon^{-2/r})$ and is the best possible $\implies$ The complexity **does NOT** increase exponentially with the input dimension!

Why is this an interesting result?

- Many functions have an underlying hierarchical composition. Example:

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= ac^2x_4^4x_1^3 + 2acx_1^3x_4^2x_3^3 + ax_1^3x_3^6 \\ &+ bc^2x_1x_2x_4^4 + 2bcx_1x_2x_4^4x_3^3 + bx_1x_2x_3^6 \\ &= (ax_1^3 + bx_2x_1)(x_3^3 + cx_4^2)^2 \\ &= h(h_{11}(x_1,x_2), h_{12}(x_3,x_4)) \end{aligned}$$

where

$$h(x,y) = xy^2, \quad h_{11}(x,y) = ax^3 + byx, \quad h_{12}(x,y) = x^3 + cy^2$$

$\Longrightarrow$ instead of approximating a 9th degree polynomial, a deep network approximates 3 polynomials of 3rd degree

References I

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function [Publisher: Springer]. *Mathematics of control, signals and systems*, *2*(4), 303–314. https://doi.org/10.1007/BF02551274

Hornik, K., Stinchcombe, M., & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, *3*(5), 551–560. https://doi.org/10.1016/0893-6080(90)90005-6

Mhaskar, H., Liao, Q., & Poggio, T. (2017). When and why are deep networks better than shallow ones? [Number: 1]. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31*(1).

References II

Mhaskar, H., & Poggio, T. (2016). Deep vs. shallow networks : An approximation theory perspective. *arXiv:1608.03287 [cs, math]*.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., & Liao, Q. (2017). Why and when can deep – but not shallow – networks avoid the curse of dimensionality: A review. *arXiv:1611.00740 [cs]*.