Third dimension: Data x Space Complexity

dr. pr. D. Papadimitriou[1,2], dr. B.C. Vu[2]

dpapadimitriou@3nlab.org

(1) Math. & Comp. Optimization Inst, Leuven, BE
(2) 3nLab & Huawei BeRC, Leuven, BE

DSO Summer School 2023
Universitat Bielefeld
Sep. 4-8, 2023

Outline and Structure

1. Handling of NP-Hardness
2. Data space properties and Space complexity
3. Function Approximation and Approximate Representation
4. External penalty method

Motivations: Why do we need to understand Complexity ?

- The performance of an algorithm influences the execution of the program. Hence, understanding the complexity of an algorithm is significant

- Thus, the designer/developer should be aware of how to find both **time** and **space** complexity of an algorithm

Motivations: Why do we need to understand Complexity ?

- The performance of an algorithm influences the execution of the program. Hence, understanding the complexity of an algorithm is significant

- Thus, the designer/developer should be aware of how to find both **time** and **space** complexity of an algorithm

- The complexity of an algorithm enables to
  - Estimate space complexity or storage : memory required for the algorithm
    $\Rightarrow$ Execute the algorithm with the optimized data resources
  - Estimate time complexity : running time taken by the algorithm to produce the output (results)
    $\Rightarrow$ Ensure results produced by the execution of the algorithm are valid and not outdated

Motivations: Why do we need to understand Complexity ?

- The performance of an algorithm influences the execution of the program. Hence, understanding the complexity of an algorithm is significant

- Thus, the designer/developer should be aware of how to find both **time** and **space** complexity of an algorithm

- The complexity of an algorithm enables to
  - Estimate space complexity or storage : memory required for the algorithm
    $\Rightarrow$ Execute the algorithm with the optimized data resources
  - Estimate time complexity : running time taken by the algorithm to produce the output (results)
    $\Rightarrow$ Ensure results produced by the execution of the algorithm are valid and not outdated

Two criteria to evaluate performance of algorithms (wrt approximation ratio): complexity in space vs. time

- **Space Complexity** of an algorithm : amount of memory it needs to run to completion.
- **Time Complexity** of an algorithm : amount of CPU time it needs to run to completion

Space complexity

Memory space S(P) needed by a program P, consists of two components

1. A fixed part: fixed elements/components

   → fixed space requirements (C) : Independent of the characteristics of the inputs and outputs

   - needed for instruction space (byte code)
   - space for simple variables, fixed-size structured variable, constants
     ⇒ comprises variables, and constants used in the program

Space complexity

Memory space S(P) needed by a program P, consists of two components

1. A fixed part: fixed elements/components

   → fixed space requirements (C) : Independent of the characteristics of the inputs and outputs

   - needed for instruction space (byte code)
   - space for simple variables, fixed-size structured variable, constants
     ⇒ comprises variables, and constants used in the program

2. A variable part : changeable elements/components

   → variable space (Sp): dependent on a particular instance (I)

   - Number, size, values of inputs and outputs associated with instance I
   - Refers to the program size: recursive stack space, formal parameters, local variables, return address

$$\boxed{S(P) = c + Sp(instance)}$$

Time complexity

Time required T(P) to run a program P also consists of two components:

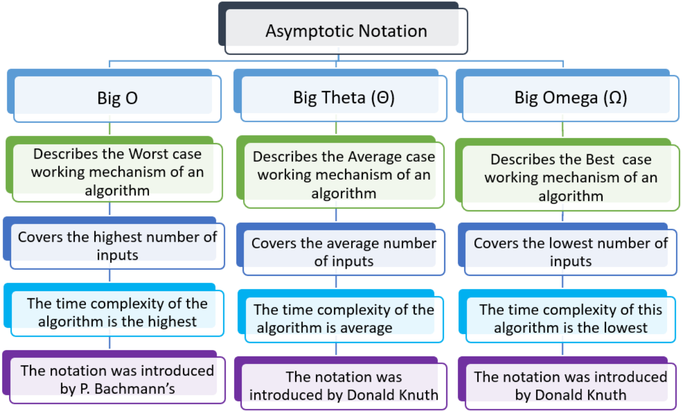
1. A fixed part (c): compile time which is independent of the problem instance
2. A variable part (Tp): run time which depends on the problem instance

$$\boxed{T(P) = c + Tp(instance)}$$

Asymptotic notation

Asymptotic notation

- analytical tool used to describe the execution of an algorithm considering its input values
- facilitate the comparative analysis of algorithms without considering their constants and input variables



Asymptotic Notation

Big O	Big Theta (Θ)	Big Omega (Ω)
Describes the Worst case working mechanism of an algorithm	Describes the Average case working mechanism of an algorithm	Describes the Best case working mechanism of an algorithm
Covers the highest number of inputs	Covers the average number of inputs	Covers the lowest number of inputs
The time complexity of the algorithm is the highest	The time complexity of the algorithm is average	The time complexity of this algorithm is the lowest
The notation was introduced by P. Bachmann's	The notation was introduced by Donald Knuth	The notation was introduced by Donald Knuth

Source: https://www.scholarhat.com/tutorial/datastructures/complexity-analysis-of-data-structures-and-algorithms

Relation to TIME Complexity Theory (1)

Time Complexity classes

- DTIME : amount of computation time (or number of computation steps) that a computer would take to solve a certain computational problem using a certain algorithm

  If a problem of input size n can be solved in $O(f(n))$

  Then, complexity class DTIME$(f(n))$

Relation to TIME Complexity Theory (1)

Time Complexity classes

- DTIME : amount of computation time (or number of computation steps) that a computer would take to solve a certain computational problem using a certain algorithm

  If a problem of input size n can be solved in $O(f(n))$

  Then, complexity class DTIME$(f(n))$

- PTIME (or P): set of all problems that can be solved by a **deterministic Turing machine** using a polynomial (amount of computation) time $O(n^k)$

  In terms of DTIME : PTIME $= \bigcup_{k\in\mathbb{N}}$ DTIME(n^k)

  - Defines class of problems which are tractable - polynomial degree can be very large

Relation to TIME Complexity Theory (1)

Time Complexity classes

- DTIME : amount of computation time (or number of computation steps) that a computer would take to solve a certain computational problem using a certain algorithm

  If a problem of input size n can be solved in $O(f(n))$

  Then, complexity class DTIME$(f(n))$

- PTIME (or P): set of all problems that can be solved by a **deterministic Turing machine** using a polynomial (amount of computation) time $O(n^k)$

  In terms of DTIME : $\text{PTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$

  - Defines class of problems which are tractable - polynomial degree can be very large

- NTIME : set of all problems that can be solved by a **nondeterministic Turing machine** in time $O(f(n))$

  Complexity class NP defined in terms of NTIME : $\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$

  where NTIME(n^k) : set of problems that can be solved by a NTM in polynomial time $O(n^k)$

Relation to TIME Complexity Theory (2)

- EXPTIME : set of all problems that are solvable by a **deterministic TM** in exponential time, i.e., in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of n

  In terms of DTIME : $\text{EXPTIME} = \bigcup_{k\in\mathbb{N}} \text{DTIME}\left(2^{n^k}\right)$

- NEXPTIME : set of all problems that can be solved by a **non-deterministic TM** in $2^{n^{O(1)}}$ time

  In terms of NTIME : $\text{NEXPTIME} = \bigcup_{k\in\mathbb{N}} \text{NTIME}\left(2^{n^k}\right)$

Relation to TIME Complexity Theory (2)

- EXPTIME : set of all problems that are solvable by a **deterministic TM** in exponential time, i.e., in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of n

  In terms of DTIME : $\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}\left(2^{n^k}\right)$

- NEXPTIME : set of all problems that can be solved by a **non-deterministic TM** in $2^{n^{O(1)}}$ time

  In terms of NTIME : $\text{NEXPTIME} = \bigcup_{k \in \mathbb{N}} \text{NTIME}\left(2^{n^k}\right)$

Relationship to NP-HARDness ?

- Warning
  - There are problems in EXPTIME that are not NP-hard
  - There are NP-hard problems that are not in EXPTIME
- Relates to **NP (class) problems**: computational problems that can be solved in polynomial time by NTM (and can be verified in polynomial time by DTM)
- A problem C is NP-complete if (1) C is in NP, and (2) Every problem in NP is reducible to C in polynomial time
- A problem X is NP-hard if only (2)
  - If C is NP-complete; then C is NP-hard (not vice-versa)
  - A problem is **NP-complete** if it is both NP and NP-hard

NP-Hard CO

NP-Hard Combinatorial Optimization (CO) problems

- MCF : Min-cost (unsplittable) flow problem
- MCP : Multi-constrained (single) path problem
- QAP : Quadratic assignment problem
- . . .

In general: math.program with integer/binary variables ($\Rightarrow$ nonconvex) $\Rightarrow$ NP-Hardness

NP-Hard CO: Classes

Approximation Scheme (AS): ALG solves OP with input instance I of size *n*

Polynomial Time AS (PTAS): $(1 \pm \varepsilon)$ approximation ratio

- For min.OP: ALG/OPT $\leq (1 + \varepsilon)$, for $\varepsilon > 0$
- For max.OP: OPT/ALG $\leq (1 - \varepsilon)$, for $\varepsilon > 0$
- Time complexity: polynomial in the size *n* of input instance I (PTIME) for any fixed > 0
- Example: min. partitioning, multiple knapsack
- Sub-class **Fully-Polynomial Time AS (FPTAS)**
    - Time complexity: polynomial in both input size *n* **and** $1/\varepsilon$
    - Example: 0-1 knapsack problem

NP-Hard CO: Classes

Approximation Scheme (AS): ALG solves OP with input instance I of size n

Polynomial Time AS (PTAS): $(1 \pm \varepsilon)$ approximation ratio

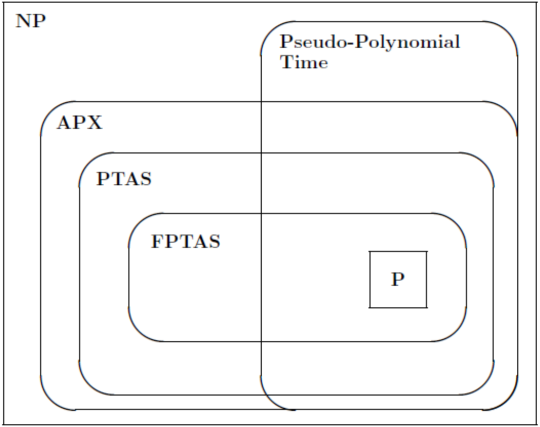
- For min.OP: ALG/OPT $\leq (1 + \varepsilon)$, for $\varepsilon > 0$
- For max.OP: OPT/ALG $\leq (1 - \varepsilon)$, for $\varepsilon > 0$
- Time complexity: polynomial in the size n of input instance I (PTIME) for any fixed > 0
- Example: min. partitioning, multiple knapsack
- Sub-class **Fully-Polynomial Time AS (FPTAS)**
  - Time complexity: polynomial in both input size n **and** $1/\varepsilon$
  - Example: 0-1 knapsack problem

APX

- all min.OP that admit PTIME approximation algorithm with some finite worst case approx. ratio (constant-factor c approximation algorithms): ALG/OPT $\leq c$
- all max.OP that admit PTIME approximation algorithm with some positive worst case approx. ratio: OPT/ALG $\leq c$
- Example: metric-TSP, MIS, min.vertex cover
- Note: $f(n)$-APX: approximation by a factor of $f(n)$

NP-Hard CO: Approximation

Strict inclusion: $\mathrm{P} \subsetneq \mathrm{FPTAS} \subsetneq \mathrm{PTAS} \subsetneq \mathrm{APX} \subsetneq \mathrm{NP}$



A **pseudo-polynomial time** (numeric) algorithm : worst-case time complexity is polynomial in the numeric value of input (the largest integer present in the input) but not necessarily in the length of the input (the number of bits required to represent it),

Source: Lectures on Scheduling, edited by R.H. Moehring, C.N. Potts, A.S. Schulz, G.J. Woeginger, L.A. Wolsey, 2009.

NP-Hard CO: Involvement of ML ?

Involvement of (machine/statistical) learning-based methods

- Combining: NP-Hard (often) × NP-Hard CO
- Q: Does it help ?
- Yes

  ...it may help in reducing search space for NP-Hard CO problem solving schemes

Examples from CO

How can it help ?

Example: **Path-enumeration problem**

- k-shortest simple (no cycle) path problem (kSSP)
  - Directed graphs: $O(kn(m + nlog(n)))$ [Yen1971]
    Note: for undirected graphs: $O(k(m + nlog(n)))$ [Katoh1982]
  - **How to set k ?**

Examples from CO

How can it help ?

Example: **Path-enumeration problem**

- k-shortest simple (no cycle) path problem (kSSP)
  - Directed graphs: $O(kn(m + nlog(n)))$ [Yen1971]
    Note: for undirected graphs: $O(k(m + nlog(n)))$ [Katoh1982]
  - **How to set k ?**
- Exhaustive search (simple paths)

$$O((n-1) \times (n-2) \times \ldots \times n - (n-2) \times n - (n-1)) \quad (1)$$

$$= O((n-1) \times (n-2) \times \ldots \times 2 \times 1) \quad (2)$$

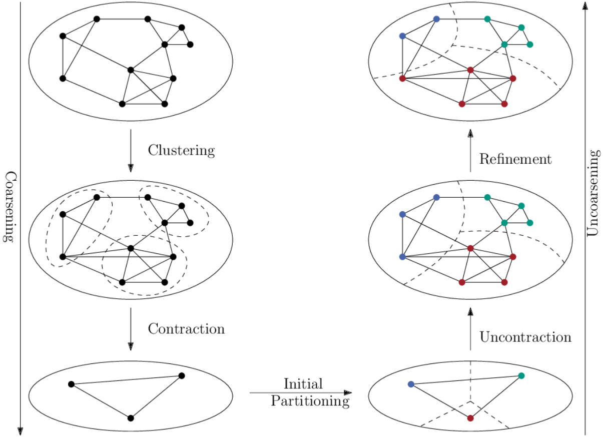
$$= O((n-1)!) \quad (3)$$

per vertex !

- Fundamental to identify "size" of search space itself

Examples from CO

- Many techniques to reduce (or characterize) search space
  - Graph sparsification: clustering - coarsening (by edge contraction) - partitioning
  - Dimensionality reduction: (input) $\mathbb{R}^n \rightarrow \mathbb{R}^d$ (data), $n \gg d$
  - ...



Source: Parallel and External High Quality Graph Partitioning Yaroslav Akhremtsev, 2019, Computer Science

Examples from CO

But then what about **strongly** NP-Hard CO ? (no FPTAS)

- strongly NP-Hard : still NP-hard even when all numbers in the input are bounded by some polynomial in the length of the input) ?
- Ex: subset-sum partition, bin-packing, etc. are strongly NP-Hard problems
- In layman terms: still intractable on small instances

Dealing with NP-Hardness

Approximation: Compromise on exactness (bound on quality of solution)

- Find an efficient algorithm running in $O(n^c)$ time that may not return the exact solution but something “close” (approximate solution) with guarantees of performance
- Example: instead of finding a k-clique, maybe will find a k/2-clique or k vertices that are “almost” a clique

Dealing with NP-Hardness

Approximation: Compromise on exactness (bound on quality of solution)

- Find an efficient algorithm running in $O(n^c)$ time that may not return the exact solution but something "close" (approximate solution) with guarantees of performance
- Example: instead of finding a k-clique, maybe will find a k/2-clique or k vertices that are "almost" a clique

Restriction/Reduction: Compromise on generality

- Restriction to Class P (PTIME transform)
- Sometimes possible to work with restricted classes of inputs.
- Example: 3-SAT to 2-SAT $\in$ P

Dealing with NP-Hardness

Approximation: Compromise on exactness (bound on quality of solution)

- Find an efficient algorithm running in $O(n^c)$ time that may not return the exact solution but something "close" (approximate solution) with guarantees of performance
- Example: instead of finding a k-clique, maybe will find a k/2-clique or k vertices that are "almost" a clique

Restriction/Reduction: Compromise on generality

- Restriction to Class P (PTIME transform)
- Sometimes possible to work with restricted classes of inputs.
- Example: 3-SAT to 2-SAT $\in$ P

Heuristics: Compromise on efficiency

- Some problems have algorithms that run in exponential time in worst-case but worst-case does not seem to happen often in practice
- Producing feasible solution in reasonable time, improving expected runtime on large subset of instances
- Example: matheuristics, greedy heuristic, neural networks

Data ?

It is all about automatically reducing the search space and/or guiding (informed) search in this (reduced) space $><$ brute force search

Data ?

It is all about automatically reducing the search space and/or guiding (informed) search in this (reduced) space >< brute force search

Rule of thumb (usual): Know your data (space)

Data ?

It is all about automatically reducing the search space and/or guiding (informed) search in this (reduced) space >< brute force search

Rule of thumb (usual): Know your data (space)

In practice: You never know enough about your data space (in particular, when using statistical learning methods)

Data ?

It is all about automatically reducing the search space and/or guiding (informed) search in this (reduced) space >< brute force search

Rule of thumb (usual): Know your data (space)

In practice: You never know enough about your data space (in particular, when using statistical learning methods)

- **Formally**: training set made up of N samples (drawn from unknown probability distribution) $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$

Data ?

It is all about automatically reducing the search space and/or guiding (informed) search in this (reduced) space >< brute force search

Rule of thumb (usual): Know your data (space)

In practice: You never know enough about your data space (in particular, when using statistical learning methods)

- **Formally**: training set made up of N samples (drawn from unknown probability distribution) $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$
- For example: $N \ll (n-1)!$ with $n = 17 : (n-1)! = 16! = 2.0e13$

Data ?

It is all about automatically reducing the search space and/or guiding (informed) search in this (reduced) space >< brute force search

Rule of thumb (usual): Know your data (space)

In practice: You never know enough about your data space (in particular, when using statistical learning methods)

- **Formally**: training set made up of N samples (drawn from unknown probability distribution) $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$
- For example: $N \ll (n-1)!$ with $n = 17 : (n-1)! = 16! = 2.0e13$

$\Rightarrow$ Learning from examples implies either Sampling or Selection

Statistical Properties of Data

- Statistical properties of data
    - Machine learning methods (supervised or not) ⇒ Learning from experience (examples)
    - May be agnostic to data generation process NOT (statistical) data properties

Statistical Properties of Data

- Statistical properties of data
  - Machine learning methods (supervised or not) ⇒ Learning from experience (examples)
  - May be agnostic to data generation process NOT (statistical) data properties
- Statistical learning methods and learned models: more explanatory (informative) than predictive
  - “(Still) Better at explaining the past than predicting the future”

Statistical Properties of Data

- Statistical properties of data
  - Machine learning methods (supervised or not) ⇒ Learning from experience (examples)
  - May be agnostic to data generation process NOT (statistical) data properties

- Statistical learning methods and learned models: more explanatory (informative) than predictive
  - “(Still) Better at explaining the past than predicting the future”

- Complexity of learning tasks (not atomic)
  - Tradeoff between learning error (bias of model) and generalization error (variance of model)
  - Lack of automated/online training: end-to-end model (feature extraction + classification/regression) still involves domain knowledge/human intervention
  - Cost/gain ratio vs. (added functionality and) performance

Applicability of Statistical (Learning) methods

1. **Descriptive**: what has happened ? why has it happened (explanatory: root cause analysis, diagnostics)

   Techniques

   - Statistical methods: quantitative/qualitative data analysis
   - Pattern/event detection and recognition (feature extraction and classification)
   - Physics: Inverse modeling

Applicability of Statistical (Learning) methods

1. **Descriptive**: what has happened ? why has it happened (explanatory: root cause analysis, diagnostics)

   Techniques

   - Statistical methods: quantitative/qualitative data analysis
   - Pattern/event detection and recognition (feature extraction and classification)
   - Physics: Inverse modeling

2. **Predictive**: what might happen ? what's likely to happen ?

   Techniques

   - Statistical modeling methods: regression analysis – time series analysis (forecasting/trend)
   - Pattern/event prediction
   - Physics: Predictive/forward modeling

Applicability of Statistical (Learning) methods

1. **Descriptive**: what has happened ? why has it happened (explanatory: root cause analysis, diagnostics)

   Techniques

   - Statistical methods: quantitative/qualitative data analysis
   - Pattern/event detection and recognition (feature extraction and classification)
   - Physics: Inverse modeling

2. **Predictive**: what might happen ? what's likely to happen ?

   Techniques

   - Statistical modeling methods: regression analysis – time series analysis (forecasting/trend)
   - Pattern/event prediction
   - Physics: Predictive/forward modeling

3. **Prescriptive**: what would happen, what could happen if (which decision/action should be taken ? What should be done) ?

   Techniques

   - Structured learning
   - Math.programming models to optimize a set of decisions for directing a given "objective", for achieving desired outcome
   - Physics: none

Importance of IID Assumption

- IID Assumption: **critical assumption** in statistics, machine learning, etc.
  - Formally: tuples of set $\{(x_i, y_i)\}_{i=!}^{N}$ are drawn independently from the same joint distribution $P(x, y) = \mathbf{P(y|x)}P(x)$

    Note: $P(y|x)$ relationship to capture by means of learning algorithm either directly (discriminative): SVM, NN, etc. or indirectly (generative): HMM
  - Non-IIDness may induce bias, incompleteness, poor generalisation, performance degradation (divergence), etc.

Importance of IID Assumption

- IID Assumption: **critical assumption** in statistics, machine learning, etc.
  - Formally: tuples of set $\{(x_i, y_i)\}_{i=!}^{N}$ are drawn independently from the same joint distribution $P(x, y) = \mathbf{P}(\mathbf{y}|\mathbf{x})P(x)$

    Note: $P(y|x)$ relationship to capture by means of learning algorithm either directly (discriminative): SVM, NN, etc. or indirectly (generative): HMM
  - Non-IIDness may induce bias, incompleteness, poor generalisation, performance degradation (divergence), etc.

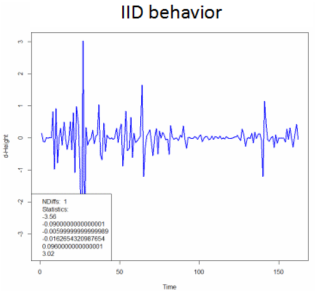
- **Definition**: collection of RV is independent and identically distributed if
  - Each tuple (x_i, y_i) has **same** probability distribution as every other sample: $(x_i, y_i) \sim P(x, y), \forall i \in \{1, \ldots, N\}$
  - All tuples are **mutually independent**

Importance of IID Assumption

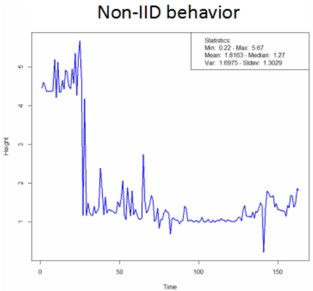
- IID Assumption: **critical assumption** in statistics, machine learning, etc.
    - Formally: tuples of set $\{(x_i, y_i)\}_{i=!}^{N}$ are drawn independently from the same joint distribution $P(x, y) = \mathbf{P}(\mathbf{y}|\mathbf{x})P(x)$

      Note: $P(y|x)$ relationship to capture by means of learning algorithm either directly (discriminative): SVM, NN, etc. or indirectly (generative): HMM
    - Non-IIDness may induce bias, incompleteness, poor generalisation, performance degradation (divergence), etc.

- **Definition**: collection of RV is independent and identically distributed if
    - Each tuple (x_i, y_i) has **same** probability distribution as every other sample: $(x_i, y_i) \sim P(x, y), \forall i \in \{1, \ldots, N\}$
    - All tuples are **mutually independent**



IID behavior



Non-IID behavior

Importance of IID Assumption

Stationary process $\not\Rightarrow$ IID process

- Stationary process: joint probability distribution of the RVs invariant to time shifts but RVs may be dependent over time (thus, not IID)

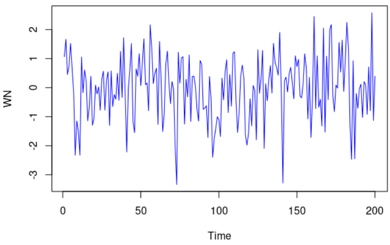
$$P(X_t|X_{1:t-1};\theta,t) = P(X_t|X_{1:t-1};\theta)$$

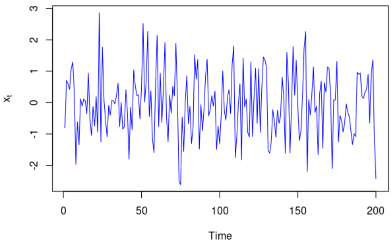
  - Stationary process of IID RVs: $P(X_t|X_{1:t-1};\theta,t) = P(X_t;\theta)$
- Note: IID $\not\Rightarrow$ Stationary process
  - Ex: process where each RV has a different probability distribution depending on the time step is IID but not stationary
- Consequence: can't use common stationarity tests ADF, PP or KPSS (for testing IIDness, in particular, checking mutual independence is far from trivial)

Importance of IID Assumption: facts

White noise process $\not\Rightarrow$ IIDness

- Conditions to be verified
  - Zero expectation: $\mathbb{E}[X_t] = 0$
  - Finite variance: $\text{Var}[X_t] < \infty$
  - Elements are uncorrelated: $\text{Cor}(X_t, X_s) = 0$
- Consequence: can't use (classical) white noise tests
- Notable exception: Gaussian white noise $\Rightarrow$ IIDness


White Noise


Gaussian WN

Implications

IID Assumption Holds

Common memoryless generative process

- Independent draws from same JPD
- Data distribution at training time $\equiv$ at testing and generalization time

Linear regression $y = f(X, \beta) + \varepsilon$, with f linear in parameters vector β

Implications

IID Assumption Holds

Common memoryless generative process

- Independent draws from same JPD
- Data distribution at training time $\equiv$ at testing and generalization time

Linear regression $y = f(X, \beta) + \varepsilon$, with f linear in parameters vector β

Error (unobserved) RV ε: adds "noise" to linear relationship between dependent variable X and regressor y

- Uncorrelated $E[\varepsilon_i \varepsilon_j | X] = 0$
- Zero mean $E[\varepsilon | X] = 0$ (exogeneity)
- Equal var. $E[\varepsilon^2 | X] = \sigma^2$ (homoskedastic)

Then Gauss-Markov theorem holds: BLUE (min.var) = least-square estimator

- (data) Normally distributed: error decreases by aggregating measures (over long time periods)
- (data) Not normally distributed: large datasets (invoke CLT)

Implications

IID Assumption Holds

Common memoryless generative process

- Independent draws from same JPD
- Data distribution at training time $\equiv$ at testing and generalization time

Linear regression $y = f(X, \beta) + \varepsilon$, with f linear in parameters vector β

Error (unobserved) RV ε: adds "noise" to linear relationship between dependent variable X and regressor y

- Uncorrelated $E[\varepsilon_i \varepsilon_j | X] = 0$
- Zero mean $E[\varepsilon | X] = 0$ (exogeneity)
- Equal var. $E[\varepsilon^2 | X] = \sigma^2$ (homoskedastic)

Then Gauss-Markov theorem holds: BLUE (min.var) = least-square estimator

- (data) Normally distributed: error decreases by aggregating measures (over long time periods)
- (data) Not normally distributed: large datasets (invoke CLT)

In practice

Non-IIDness

- Coupling/relationship (not independent) and/or heterogeneity (not identical)
- Data distribution at training $\neq$ at testing and generalization time $\Rightarrow$ Poor generalization (re-training)

Error uncorrelated, unequal variance (heteroskedastic)

- Different values of response var. y have different variance in their errors
- Gauss-Markov theorem does not apply: OLS estimators still unbiased but not efficient (not least variance)
- Weighted LS –possibly- at rescue (exact weights)

Error correlated, unequal variance

- Error distribution known (independently of data): Generalized LS
- Variance not completely known: Iterative Reweigthed LS

Comparison against classical regression/inversion methods

Stat. method: regression problems

- Given (x, y_{obs}), find parameters β such that $y_{obs} \approx f(x;\beta)$ i.e. such that f estimates relationship between dependent (y) and independent var (x)
- Then, predict $y_{pre} = f(x;\beta)$

Comparison against classical regression/inversion methods

Stat. method: regression problems

- Given (x, y_{obs}), find parameters β such that $y_{obs} \approx f(x; \beta)$ i.e. such that f estimates relationship between dependent (y) and independent var (x)
- Then, predict $y_{pre} = f(x; \beta)$

Indirect method: inverse problems

- Given (y_{obs}, m), model identification step to find g s.t. $y_{obs} \approx g(m)$
- Then, inversion method: $m \approx g^{-1}(y_{obs})$
- For nonlinear inverse problems: $y = g(m)$
  - If $g(m)$ well-behaved: first-order approx: $\Delta y = y - y_{est} = J_g \Delta m$

    Thus, $\Delta m = (J_g^\intercal J_g + \lambda \mathcal{I})^{-1} J_g^\intercal \Delta y$, where $(J_g)_{ij} = \frac{\partial g(m)_i}{\partial m_j}$
  - Otherwise, transformation to well-posed problem (second-order variational methods)

Comparison against classical regression/inversion methods

Stat. method: regression problems

- Given (x, y_{obs}), find parameters β such that $y_{obs} \approx f(x;\beta)$ i.e. such that f estimates relationship between dependent (y) and independent var (x)
- Then, predict $y_{pre} = f(x;\beta)$

Indirect method: inverse problems

- Given (y_{obs}, m), model identification step to find g s.t. $y_{obs} \approx g(m)$
- Then, inversion method: $m \approx g^{-1}(y_{obs})$
- For nonlinear inverse problems: $y = g(m)$
  - If $g(m)$ well-behaved: first-order approx: $\Delta y = y - y_{est} = J_g \Delta m$

    Thus, $\Delta m = (J_g^{\intercal} J_g + \lambda \mathcal{I})^{-1} J_g^{\intercal} \Delta y$, where $(J_g)_{ij} = \frac{\partial g(m)_i}{\partial m_j}$
  - Otherwise, transformation to well-posed problem (second-order variational methods)

Direct/algorithmic method: NN approximates g^{-1}

- Given (y_{obs}, m), find g^{-1} such that $m \approx g^{-1}(y_{obs})$
- Then, predict $y_{pre} = g(m)$
- How (forward process): Invertible Neural Networks (INNs), Bayesian NN

Outline and Structure

1. Handling of NP-Hardness
2. Data space properties and Space complexity
3. Function Approximation and Approximate Representation
4. External penalty method

Space Complexity Theory: DSPACE

In computational complexity theory

- DSPACE: defines the set of all problems that can be solved by Turing machines using $O(f(n))$ space for some function f of the input size n
  - Represents the total amount of computational resource (memory space) for a deterministic Turing machine to solve a given computational problem with a given algorithm
  - Corresponds closely to an important physical resource: the amount of physical computer memory needed to run a given program

Space Complexity Theory: DSPACE

In computational complexity theory

- DSPACE: defines the set of all problems that can be solved by Turing machines using $O(f(n))$ space for some function f of the input size n
  - Represents the total amount of computational resource (memory space) for a deterministic Turing machine to solve a given computational problem with a given algorithm
  - Corresponds closely to an important physical resource: the amount of physical computer memory needed to run a given program
- Measure DSPACE is used to define complexity classes, sets of all of the decision problems that can be solved using a certain amount of memory space
- For each function $f(n)$, there is a complexity class SPACE(f(n)), the set of decision problems that can be solved by a deterministic Turing machine using space $O(f(n))$
- There is no restriction on the amount of computation time that can be used, though there may be restrictions on some other complexity measures (like alternation)

Space Complexity Theory: PSPACE

In computational complexity theory, PSPACE is the set of all decision problems which can be solved by a Turing machine using a polynomial amount of space.

Space Complexity Theory: PSPACE

In computational complexity theory, PSPACE is the set of all decision problems which can be solved by a Turing machine using a polynomial amount of space.

Formal definition: PSPACE

- **If** DSPACE($t(n)$) denotes the set of all problems that can be solved by Turing machines using at most $t(n)$ space for some function t of the input size n
- **Then** PSPACE can be formally defined as

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k)$$

Space Complexity Theory: PSPACE

In computational complexity theory, PSPACE is the set of all decision problems which can be solved by a Turing machine using a polynomial amount of space.

Formal definition: PSPACE

- **If** DSPACE($t(n)$) denotes the set of all problems that can be solved by Turing machines using at most $t(n)$ space for some function t of the input size n
- **Then** PSPACE can be formally defined as

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k)$$

Examples

- Finite horizon POMDPs (Partially Observable Markov Decision Processes)
- Canadian traveller problem (CTP): generalization of SP problem to graphs that are partially observable
- Plan existence problem in automated planning and scheduling

Space Complexity Theory: EXPSPACE

EXPSPACE is the set of all decision problems solvable by a deterministic Turing machine in exponential space, i.e., in $O(2^{p(n)})$ space, where $p(n)$ is a polynomial function of n

Formal definition: EXPSPACE

- **If** DSPACE($t(n)$) denotes the set of all problems that can be solved by Turing machines using at most $t(n)$ space for some function t of the input size n
- Then PSPACE can be formally defined as

$$\text{EXSPACE} = \bigcup_{k\in\mathbb{N}} \text{DSPACE}(2^{n^k})$$

Space Complexity Theory: EXPSPACE

EXPSPACE is the set of all decision problems solvable by a deterministic Turing machine in exponential space, i.e., in $O(2^{p(n)})$ space, where $p(n)$ is a polynomial function of n

Formal definition: EXPSPACE

- **If** DSPACE($t(n)$) denotes the set of all problems that can be solved by Turing machines using at most $t(n)$ space for some function t of the input size n
- Then PSPACE can be formally defined as

$$\text{EXSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(2^{n^k})$$

Examples

- Temporal Planning with concurrent actions
- Verifying finite concurrent systems through Linearizability [Alur, 2000]

Space Complexity Theory: EXPSPACE

EXPSPACE is the set of all decision problems solvable by a deterministic Turing machine in exponential space, i.e., in $O(2^{p(n)})$ space, where $p(n)$ is a polynomial function of n

Formal definition: EXPSPACE

- **If** DSPACE($t(n)$) denotes the set of all problems that can be solved by Turing machines using at most $t(n)$ space for some function t of the input size n
- Then PSPACE can be formally defined as

$$\text{EXSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(2^{n^k})$$

Examples

- Temporal Planning with concurrent actions
- Verifying finite concurrent systems through Linearizability [Alur, 2000]

Note: PSPACE $\subseteq$ EXPTIME $\subseteq$ NEXPTIME $\subseteq$ EXPSPACE

Space Hierarchy Theorems

In computational complexity theory

- **Space hierarchy theorems**: separation results that show that both deterministic and nondeterministic Turing machines can solve more problems in (asymptotically) more space, subject to certain conditions.
    - For example, a DTM can solve more decision problems in space $n^2 \log(n)$ than in space $n \log(n)$ than in space n.
    - The somewhat weaker analogous theorems for time are the time hierarchy theorems

Space Hierarchy Theorems

In computational complexity theory

- **Space hierarchy theorems**: separation results that show that both deterministic and nondeterministic Turing machines can solve more problems in (asymptotically) more space, subject to certain conditions.
    - For example, a DTM can solve more decision problems in space $n^2 \log(n)$ than in space $n \log(n)$ than in space n.
    - The somewhat weaker analogous theorems for time are the time hierarchy theorems
- Foundation for hierarchy theorems lies in the intuition that with either more time or more space comes the ability to compute more functions (or decide more languages)

Space Hierarchy Theorems

In computational complexity theory

- **Space hierarchy theorems**: separation results that show that both deterministic and nondeterministic Turing machines can solve more problems in (asymptotically) more space, subject to certain conditions.
  - For example, a DTM can solve more decision problems in space $n^2 \log(n)$ than in space $n \log(n)$ than in space n.
  - The somewhat weaker analogous theorems for time are the time hierarchy theorems
- Foundation for hierarchy theorems lies in the intuition that with either more time or more space comes the ability to compute more functions (or decide more languages)
- Hierarchy theorems are used to demonstrate that time and space complexity classes form a hierarchy where classes with tighter bounds contain fewer languages than those with more relaxed bounds.

Space Hierarchy Theorems

Space hierarchy theorems rely on the concept of space-constructible functions

Formally: function $f : \mathbb{N} \rightarrow \mathbb{N}$ is space constructible if

- $f(n) \geq log(n)$
- and there exists a Turing machine which computes the function $f(n)$ in space $O(f(n))$ when starting with an input 1^n, where 1^n represents a string of n 1's

Space Hierarchy Theorems

Space hierarchy theorems rely on the concept of space-constructible functions

Formally: function $f : \mathbb{N} \rightarrow \mathbb{N}$ is space constructible if

- $f(n) \geq log(n)$
- and there exists a Turing machine which computes the function $f(n)$ in space $O(f(n))$ when starting with an input 1^n, where 1^n represents a string of n 1's

Note: common functions are space-constructible, including polynomials, exponents, and logarithms.

Space Hierarchy Theorems

Space hierarchy theorems rely on the concept of space-constructible functions

Formally: function $f : \mathbb{N} \rightarrow \mathbb{N}$ is space constructible if

- $f(n) \geq log(n)$
- and there exists a Turing machine which computes the function $f(n)$ in space $O(f(n))$ when starting with an input 1^n, where 1^n represents a string of n 1's

Note: common functions are space-constructible, including polynomials, exponents, and logarithms.

Deterministic and nondeterministic space hierarchy theorems state that

for all space-constructible functions f(n), $\text{SPACE}(o(f(n))) \subsetneq \text{SPACE}(f(n))$, where SPACE stands for either DSPACE or NSPACE

Space Complexity Theory: Classes

Class of regular languages REG $=$ DSPACE$(o(\log\log n))$ i.e. $\Omega(\log\log n)$ space is required to recognize any non-regular language

Space Complexity Theory: Classes

Class of regular languages REG $=$ DSPACE$(o(\log\log n))$ i.e. $\Omega(\log\log n)$ space is required to recognize any non-regular language

Statement: For every space constructible function $f : \mathbb{N} \to \mathbb{N}$, there exists a language L that is decidable in space $O(f(n))$ but not in space $o(f(n))$

Space Complexity Theory: Classes

Class of regular languages REG $=$ DSPACE$(o(\log\log n))$ i.e. $\Omega(\log\log n)$ space is required to recognize any non-regular language

Statement: For every space constructible function $f : \mathbb{N} \to \mathbb{N}$, there exists a language L that is decidable in space $O(f(n))$ but not in space $o(f(n))$

The above theorem implies the necessity of the space-constructible function assumption in the space hierarchy theorem

- L(ogSpace) $=$ DSPACE$(log(n))$
- PSPACE $= \bigcup_{k\in\mathbb{N}}$ DSPACE(n^k)
- EXPSPACE $= \bigcup_{k\in\mathbb{N}}$ DSPACE(2^{n^k})

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem (1970) for any function $f : \mathbb{N} \rightarrow \mathbb{R}^{+}$ with $f(n) \geq logn$: NSPACE$(f(n)) \subseteq$ DSPACE$(f^2(n))$

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem (1970) for any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq logn$: NSPACE$(f(n)) \subseteq$ DSPACE$(f^2(n))$

Consequence: nondeterminism adds almost no power to space-bounded TM i.e. allowing Turing machine to be non-deterministic does not add any extra power

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem (1970) for any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq logn$: NSPACE($f(n)$) $\subseteq$ DSPACE($f^2(n)$)

Consequence: nondeterminism adds almost no power to space-bounded TM i.e. allowing Turing machine to be non-deterministic does not add any extra power

Reason: because of Savitch's theorem, NPSPACE is equivalent to PSPACE, essentially because a deterministic Turing machine can simulate a nondeterministic Turing machine without needing much more space (even though it may use much more time)

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem (1970) for any function $f : \mathbb{N} \to \mathbb{R}^+$ with $f(n) \geq logn$: NSPACE($f(n)$) $\subseteq$ DSPACE($f^2(n)$)

Consequence: nondeterminism adds almost no power to space-bounded TM i.e. allowing Turing machine to be non-deterministic does not add any extra power

Reason: because of Savitch's theorem, NPSPACE is equivalent to PSPACE, essentially because a deterministic Turing machine can simulate a nondeterministic Turing machine without needing much more space (even though it may use much more time)

Note: the complements of all problems in PSPACE are also in PSPACE, meaning that Co-PSPACE = PSPACE

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem: NPSPACE is equivalent to PSPACE

$$\text{NPSPACE} = \text{PSPACE}$$

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem: NPSPACE is equivalent to PSPACE

$$\text{NPSPACE} = \text{PSPACE}$$

Consequences

- Adding non-determinism to the TM does not take up any more space
  Even though it may take up more time

Savitch's theorem: NPSPACE is equivalent to PSPACE

Savitch's theorem: NPSPACE is equivalent to PSPACE

$$\text{NPSPACE} = \text{PSPACE}$$

Consequences

- Adding non-determinism to the TM does not take up any more space

  Even though it may take up more time
- We can simulate a NTM on a DTM without needing more than a polynomial increase in space

  Even though there is a (potentially) exponential increase in the number of states

Relation to TIME Complexity Theory (1)

Time Complexity classes

- DTIME : amount of computation time (or number of computation steps) that a computer would take to solve a certain computational problem using a certain algorithm

  If a problem of input size n can be solved in $O(f(n))$

  Then, complexity class DTIME($f(n)$)

Relation to TIME Complexity Theory (1)

Time Complexity classes

- DTIME : amount of computation time (or number of computation steps) that a computer would take to solve a certain computational problem using a certain algorithm

  If a problem of input size n can be solved in $O(f(n))$

  Then, complexity class DTIME$(f(n))$

- PTIME (or P): set of all problems that can be solved by a **deterministic Turing machine** using a polynomial (amount of computation) time $O(n^k)$

  In terms of DTIME : $\text{PTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$

  - Defines class of problems which are tractable - polynomial degree can be very large

Relation to TIME Complexity Theory (1)

Time Complexity classes

- DTIME : amount of computation time (or number of computation steps) that a computer would take to solve a certain computational problem using a certain algorithm

  If a problem of input size n can be solved in $O(f(n))$

  Then, complexity class DTIME$(f(n))$

- PTIME (or P): set of all problems that can be solved by a **deterministic Turing machine** using a polynomial (amount of computation) time $O(n^k)$

  In terms of DTIME : PTIME $= \bigcup_{k \in \mathbb{N}}$ DTIME(n^k)

  - Defines class of problems which are tractable - polynomial degree can be very large

- NTIME : set of all problems that can be solved by a **nondeterministic Turing machine** in time $O(f(n))$

  Complexity class NP defined in terms of NTIME : NP $= \bigcup_{k \in \mathbb{N}}$ NTIME(n^k)

  where NTIME(n^k) : set of problems that can be solved by a NTM in polynomial time $O(n^k)$

Relation to TIME Complexity Theory (2)

- EXPTIME : set of all problems that are solvable by a **deterministic TM** in exponential time, i.e., in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of n

  In terms of DTIME : $\mathrm{EXPTIME} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}\left(2^{n^k}\right)$
- NEXPTIME : set of all problems that can be solved by a **non-deterministic TM** in $2^{n^{O(1)}}$ time

  In terms of NTIME : $\mathrm{NEXPTIME} = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}\left(2^{n^k}\right)$

Relation with other complexity classes

DSPACE is the deterministic counterpart of NSPACE: the class of memory space on a non-deterministic Turing machine

Relation with other complexity classes

DSPACE is the deterministic counterpart of NSPACE: the class of memory space on a non-deterministic Turing machine

By Savitch's theorem, we have that

$$\mathrm{DSPACE}[s(n)] \subseteq \mathrm{NSPACE}[s(n)] \subseteq \mathrm{DSPACE}[(s(n))^2]$$

Relation with other complexity classes

DSPACE is the deterministic counterpart of NSPACE: the class of memory space on a non-deterministic Turing machine

By Savitch's theorem, we have that

$$\mathrm{DSPACE}[s(n)] \subseteq \mathrm{NSPACE}[s(n)] \subseteq \mathrm{DSPACE}[(s(n))^2]$$

NTIME relation to DSPACE: for any time constructible function $t(n)$, we have

$$\mathrm{NTIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$$

Relation with other complexity classes

DSPACE is the deterministic counterpart of NSPACE: the class of memory space on a non-deterministic Turing machine

By Savitch's theorem, we have that

$$\mathrm{DSPACE}[s(n)] \subseteq \mathrm{NSPACE}[s(n)] \subseteq \mathrm{DSPACE}[(s(n))^2]$$

NTIME relation to DSPACE: for any time constructible function $t(n)$, we have

$$\mathrm{NTIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$$

Relationships to other classes:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME} \subseteq \mathrm{EXPSPACE}$$

Relation with other complexity classes

DSPACE is the deterministic counterpart of NSPACE: the class of memory space on a non-deterministic Turing machine

By Savitch's theorem, we have that

$$\mathrm{DSPACE}[s(n)] \subseteq \mathrm{NSPACE}[s(n)] \subseteq \mathrm{DSPACE}[(s(n))^2]$$

NTIME relation to DSPACE: for any time constructible function $t(n)$, we have

$$\mathrm{NTIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$$

Relationships to other classes:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME} \subseteq \mathrm{EXPSPACE}$$

By time hierarchy theorem and space hierarchy theorem:

$$\mathrm{P} \subsetneq \mathrm{EXPTIME}, \mathrm{NP} \subsetneq \mathrm{NEXPTIME} \text{ and } \mathrm{PSPACE} \subsetneq \mathrm{EXPSPACE}$$

PSPACE-complete

Definition: PSPACE

All problems which can be solved by programs which only need a polynomial (in the length of the problem instance specification) amount of memory to run

PSPACE-complete

Definition: PSPACE

All problems which can be solved by programs which only need a polynomial (in the length of the problem instance specification) amount of memory to run

It turns out that some of these problems are essentially as hard as any such problem can be: such problems are called **PSPACE complete**.

Definition: PSPACE-complete

In complexity theory, a decision problem is PSPACE-complete IF

- it is in PSPACE (thus, can be solved using a polynomial amount of memory)
- and every problem in PSPACE can be reduced to it in polynomial time.

PSPACE-complete

Definition: PSPACE

All problems which can be solved by programs which only need a polynomial (in the length of the problem instance specification) amount of memory to run

It turns out that some of these problems are essentially as hard as any such problem can be: such problems are called **PSPACE complete**.

Definition: PSPACE-complete

In complexity theory, a decision problem is PSPACE-complete IF

- it is in PSPACE (thus, can be solved using a polynomial amount of memory)
- and every problem in PSPACE can be reduced to it in polynomial time.

- The problems that are PSPACE-complete can be thought of as the hardest problems in PSPACE
- These problems are widely suspected to be outside of P and NP (but that is not known)
- It is known that they lie outside of NC (problems decidable in polylog. time on a parallel computer with a polynomial number of processors)

PSPACE-complete vs. NP-complete

PSPACE-complete problems take **exponential time** to deterministically compute the result

- Just like NP-complete problems
- Either the decision or functional problem versions

PSPACE-complete vs. NP-complete

PSPACE-complete problems take **exponential time** to deterministically compute the result

- Just like NP-complete problems
- Either the decision or functional problem versions

But PSPACE-complete problems take **exponential time** to deterministically verify the result $><$ NP-complete can verify a solution in polynomial time

Outline and Structure

1. Handling of NP-Hardness
2. Data space properties and Space complexity
3. Function Approximation and Approximate Representation
4. External penalty method

Characterizing (Artificial) Neural Networks

- **Expressivity**: what class(es) of functions $\mathcal{F}$ can be represented/approximated by neural network Φ cf. Universal approximation Theorem [Cybenko1989], [Hornik1991], [Leshno1993]

Characterizing (Artificial) Neural Networks

- **Expressivity**: what class(es) of functions $\mathcal{F}$ can be represented/approximated by neural network Φ cf. Universal approximation Theorem [Cybenko1989], [Hornik1991], [Leshno1993]

- **Effectiveness** vs. **Efficiency**: how many layers (depth) and units per layer (width) are needed to compute functions $f(\in \mathcal{F}) : \mathbb{R}^d \to \mathbb{R}$

  Bounds depend on i) approx. error, ii) input dim. d and iii) structure (depth L vs. width W)

Characterizing (Artificial) Neural Networks

- **Expressivity**: what class(es) of functions $\mathcal{F}$ can be represented/approximated by neural network Φ cf. Universal approximation Theorem [Cybenko1989], [Hornik1991], [Leshno1993]

- **Effectiveness** vs. **Efficiency**: how many layers (depth) and units per layer (width) are needed to compute functions $f(\in \mathcal{F}) : \mathbb{R}^d \to \mathbb{R}$

  Bounds depend on i) approx. error, ii) input dim. d and iii) structure (depth L vs. width W)

- **Learnability/Trainability**: given structure (nbr of layers, neurons per layer), activation function σ, and labeled data points (x, y), find parameters w for the best fitting function $f \in F \to$ curse of non-convexity and dimensionality

  Note: low complexity $f \not\Rightarrow$ highly trainable model (low training resources/time)

Characterizing (Artificial) Neural Networks

- **Expressivity**: what class(es) of functions $\mathcal{F}$ can be represented/approximated by neural network Φ cf. Universal approximation Theorem [Cybenko1989], [Hornik1991], [Leshno1993]

- **Effectiveness** vs. **Efficiency**: how many layers (depth) and units per layer (width) are needed to compute functions $f(\in \mathcal{F}) : \mathbb{R}^d \to \mathbb{R}$

  Bounds depend on i) approx. error, ii) input dim. d and iii) structure (depth L vs. width W)

- **Learnability/Trainability**: given structure (nbr of layers, neurons per layer), activation function σ, and labeled data points (x, y), find parameters w for the best fitting function $f \in F \to$ curse of non-convexity and dimensionality

  Note: low complexity $f \not\Rightarrow$ highly trainable model (low training resources/time)

- **Generalization**: learn a neural model that minimizes the difference between the expected and the empirical risk (generalization error on previously unseen (x, y))

  Note: deep relationship with bias vs. variance tradeoff and stability property

Expressive power of Neural Networks

Universal approximation property: neural network with finite number of neurons can uniformly approximate arbitrarily well any continuous function on compact subsets of $\mathbb{R}^d$ when given appropriate parameters

Expressive power of Neural Networks

Universal approximation property: neural network with finite number of neurons can uniformly approximate arbitrarily well any continuous function on compact subsets of $\mathbb{R}^d$ when given appropriate parameters

UA Theorem [Leshno,1993]

- Generalizes Hornik's Theorems (1989,1991) by dropping assumptions of continuity, monotonicity, and boundedness on activation function
- Establishes NSC on activation functions to ensure universal approximation property
- **Th**: standard multilayer feedforward network with a locally bounded, piecewise continuous activation function can approximate any continuous function to any degree of accuracy **iff** activation function σ is not polynomial almost everywhere

Expressive power of Neural Networks

Universal approximation property: neural network with finite number of neurons can uniformly approximate arbitrarily well any continuous function on compact subsets of $\mathbb{R}^d$ when given appropriate parameters

UA Theorem [Leshno,1993]

- Generalizes Hornik's Theorems (1989,1991) by dropping assumptions of continuity, monotonicity, and boundedness on activation function
- Establishes NSC on activation functions to ensure universal approximation property
- **Th**: standard multilayer feedforward network with a locally bounded, piecewise continuous activation function can approximate any continuous function to any degree of accuracy **iff** activation function σ is not polynomial almost everywhere

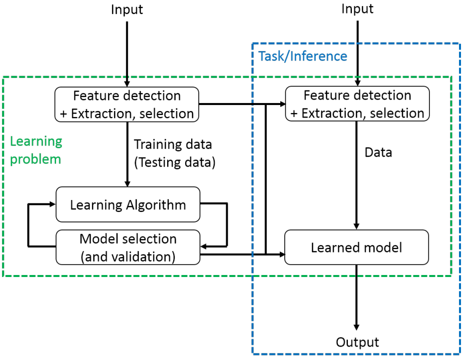
Intuition

- Neural network universality **iff** if σ does not coincide with polynomial (almost everywhere)
- Necessary condition: assume
  - σ: polynomial of degree N
  - Φ: neural network of depth L (hidden layers) and activation σ_L

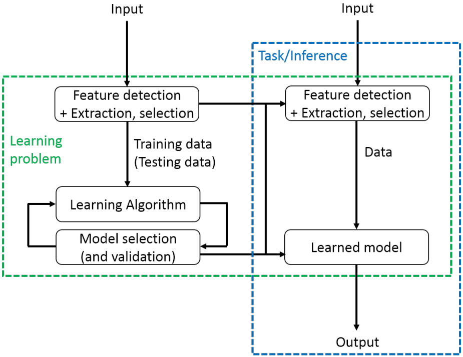
$\Rightarrow$ Each function approximated by Φ: polynomial of degree at most N^L irrespective of width W

Neural networks as ML problems solver

- Pro's: Compelling arguments for using NNs as general template (matching input to outputs) for solving machine learning (ML) problems including nonlinear regression
- Con's: BUT **designing** and **training** the right NN for given learning task coins many theoretical gaps and practical concerns


Input
Input
Task/Inference
Feature detection + Extraction, selection
Feature detection + Extraction, selection
Learning problem
Training data (Testing data)
Data
Learning Algorithm
Model selection (and validation)
Learned model
Output

Neural networks as ML problems solver


Input
Input
Task/Inference
Feature detection + Extraction, selection
Feature detection + Extraction, selection
Learning problem
Training data (Testing data)
Data
Learning Algorithm
Model selection (and validation)
Learned model
Output

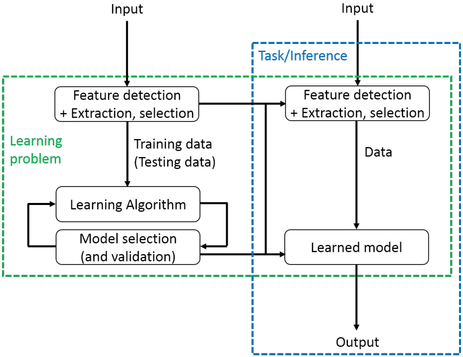
Optimization perspective: **Efficiency** ↔ **Training**

Quality of **design** and its properties: efficiency, robustness, etc.

→ Structure of learned model (model selection)

⇒ Optimization of machine learning models

Neural networks as ML problems solver



Optimization perspective: **Efficiency** ↔ **Training**

Quality of **design** and its properties: efficiency, robustness, etc.

$\rightarrow$ Structure of learned model (model selection)

$\Rightarrow$ Optimization of machine learning models

Quality of **training** and its properties: performance bounds/guarantees, robustness, etc.

$\rightarrow$ Parameters of learned model (parameter learning)

$\Rightarrow$ Computational/Optimization methods for machine learning

Learning Problem

Assumptions

- Training set $\{X, Y\}$ of n random independent and identically distributed (i.i.d.) observations/labeled data points $(x_i, y_i) \in \{\mathcal{X}, \mathcal{Y}\}, i \in \{1, \dots, n\}$
- Learning machine capable of implementing set of prediction functions $\{p(w, x) : \mathbb{R}^d \times \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}\}$ parameterized by parameter vector $w \in W \subseteq \mathbb{R}^d$

Learning Problem

Assumptions

- Training set $\{X, Y\}$ of n random independent and identically distributed (i.i.d.) observations/labeled data points $(x_i, y_i) \in \{\mathcal{X}, \mathcal{Y}\}, i \in \{1, \ldots, n\}$
- Learning machine capable of implementing set of prediction functions $\{p(w, x) : \mathbb{R}^d \times \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}\}$ parameterized by parameter vector $w \in W \subseteq \mathbb{R}^d$

Supervised Learning Problem

- Find vector w^* such that function $p(w^*, x)$ obtained from any given input $x \in \mathcal{X}$ is best at predicting the appropriate label $y \in \mathcal{Y}$ corresponding to x

Learning Problem

Assumptions

- Training set $\{X, Y\}$ of n random independent and identically distributed (i.i.d.) observations/labeled data points $(x_i, y_i) \in \{\mathcal{X}, \mathcal{Y}\}, i \in \{1, \ldots, n\}$
- Learning machine capable of implementing set of prediction functions $\{p(w, x) : \mathbb{R}^d \times \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}\}$ parameterized by parameter vector $w \in W \subseteq \mathbb{R}^d$

Supervised Learning Problem

- Find vector w^* such that function $p(w^*, x)$ obtained from any given input $x \in \mathcal{X}$ is best at predicting the appropriate label $y \in \mathcal{Y}$ corresponding to x

Optimization Problem

- Loss function: given pair (x, y), parameter vector w and predicted output $p(w, x)$

$$\ell : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to \mathbb{R} : (x, y) \to \ell(p(w, x), y) \tag{4}$$

Measures discrepancy between true output y (supervisor response) to a given input x and predicted output $p(w, x)$ (learning machine response)

$\Rightarrow$ Find parameter vector w^* which minimizes loss incurred from any input-output pair

Learning Problem: Expected and Empirical risk

Expected Risk: $\mathcal{L}(w)$

- Expected risk: $\mathcal{L}(w) = \int \ell(p(w,x),y)dP(x,y) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$ (5)

  $P(x,y) : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \rightarrow [0,1]$ unknown joint probability distribution representing true relationship between inputs and outputs → **Stochastic optimization** with unknown P

- Expected risk minimization: $w^* = \arg\min_{w \in W} \mathcal{L}(w) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$ (6)
- Goal: find function $p(w^*,x)$ which minimizes $\mathcal{L}(w)$ over $\{p(w,x) : w \in W \subseteq \mathbb{R}^p\}$

Learning Problem: Expected and Empirical risk

Expected Risk: $\mathcal{L}(w)$

- Expected risk: $\mathcal{L}(w) = \int \ell(p(w,x),y)dP(x,y) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$ (5)

  $P(x,y) : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to [0,1]$ unknown joint probability distribution representing true relationship between inputs and outputs → **Stochastic optimization** with unknown P
- Expected risk minimization: $w^* = \arg\min_{w \in W} \mathcal{L}(w) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$ (6)
- Goal: find function $p(w^*,x)$ which minimizes $\mathcal{L}(w)$ over $\{p(w,x) : w \in W \subseteq \mathbb{R}^p\}$

Empirical Risk: $\mathcal{L}_n(w)$

- Empirical risk: $\hat{\mathcal{L}}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w,x_i),y_i)$ (7)

  Measures error on given training set $\{X,Y\} = \{(x_i,y_i) : i = 1,\ldots,n\}$ with each (x_i,y_i) selected independently at random from $\{\mathcal{X},\mathcal{Y}\})$ → **Sample average approximation**
- Empirical risk minimization: $\hat{w} = \arg\min_{w \in W} \hat{\mathcal{L}}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w,x_i),y_i)$ (8)

Learning Problem: Expected and Empirical risk

Expected Risk: $\mathcal{L}(w)$

- Expected risk: $\mathcal{L}(w) = \int \ell(p(w,x),y)dP(x,y) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$ (5)

  $P(x,y) : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to [0,1]$ unknown joint probability distribution representing true relationship between inputs and outputs → **Stochastic optimization** with unknown P
- Expected risk minimization: $w^* = \arg\min_{w \in W} \mathcal{L}(w) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$ (6)
- Goal: find function $p(w^*,x)$ which minimizes $\mathcal{L}(w)$ over $\{p(w,x) : w \in W \subseteq \mathbb{R}^p\}$

Empirical Risk: $\mathcal{L}_n(w)$

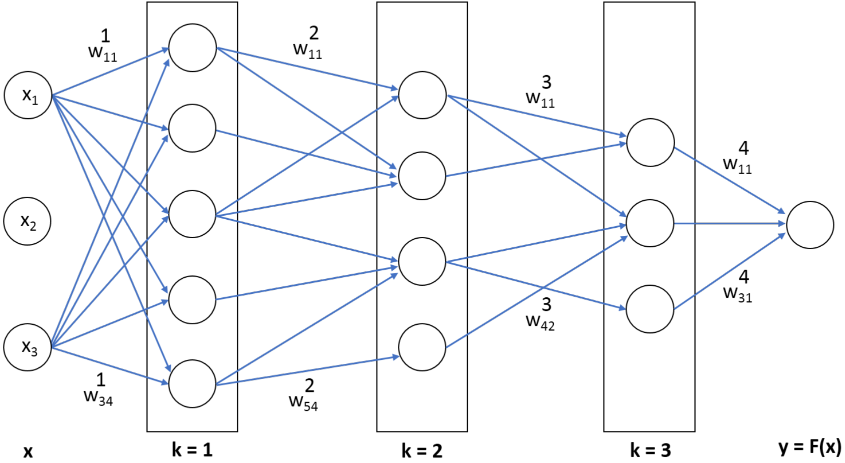
- Empirical risk: $\hat{\mathcal{L}}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w,x_i),y_i)$ (7)

  Measures error on given training set $\{X,Y\} = \{(x_i,y_i) : i = 1,\dots,n\}$ with each (x_i,y_i) selected independently at random from $\{\mathcal{X},\mathcal{Y}\})$ → **Sample average approximation**
- Empirical risk minimization: $\hat{w} = \arg\min_{w \in W} \hat{\mathcal{L}}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w,x_i),y_i)$ (8)

Empirical Risk Minimization (ERM principle): **induction principle**

Approximate the function which minimizes the expected risk (the problem we **want to solve**) by the function which minimizes empirical risk (the problem we **can solve**)

Neural Networks: Example



Learning Problem for Neural Networks

For an unknown target function $f \in \mathcal{F}$ and given training data set $\{X, Y\}$ (randomly selected from distribution $\{\mathcal{X}, \mathcal{Y}\}$)

1. **Selection of NN structure**: depth L and width W
   Interconnection between neurons arranged in one or multiple layers
   Transformation functions each neuron/unit j at layer k performs:
   - **Transfer potential**: weighted sum of input vector $x^{(k-1)}$ ($= x^{(0)}$ for the first layer) which is passed by unit i at layer $k-1$ to neuron/unit j at layer k

     $$\sum_{i=1}^{L_{k-1}} w_{ij}^{(k)} x_i^{(k-1)} \tag{9}$$

   - **Activation function** σ: non-/linear function applied to the transfer potential by each neuron j at layer k to produce

     $$x_j^{(k)} = \sigma(\sum_{i=1}^{L_{k-1}} w_{ij}^{(k)} x_i^{(k-1)}) \tag{10}$$

   - Output $p^{(k)}(w_j^{(k)}, x_j^{(k)})$ of neuron/unit j in layer k defines input vector $x^{(k+1)}$ for next layer $k+1$
2. Minimization of empirical loss $\hat{\mathcal{L}}$ on given training data set $\{X, Y\}$ from distribution $\{\mathcal{X}, \mathcal{Y}\} \rightarrow \hat{w}$
3. Estimation of generalization ability (testing loss) $\hat{\mathcal{L}}(\hat{w})$ on test data set $\{X^t, Y^t\}$ (randomly selected from the same distribution $\{\mathcal{X}, \mathcal{Y}\}$)

Representational capacity of neural networks

UAT characterizes **representational capacity** of neural networks: set of hypotheses a given model is capable of expressing when assigning some value to its parameters

Representational capacity of neural networks

UAT characterizes **representational capacity** of neural networks: set of hypotheses a given model is capable of expressing when assigning some value to its parameters

Main focus

i) Existence of approximations without explicit constructive method or explicit construction methods involving activation functions that are unusable in practice due to their complexity cf. [Maiorov-Pinkus,1999]

ii) Properties of approximations by analyzing their error for certain classes of functions provided that sufficiently many hidden units (N) are available

$\Rightarrow$ Characterizing W and L simultaneously in approximation rate remains open and challenging

iii) Asymptotic estimates have limited applicability when properties of fixed neural structure of finite size need to be understood;

$\Rightarrow$ Very few results address quantitative and non-asymptotic approximation rate of neural networks

iv) Characterizing which functions (class of) NNs can express over entire domain instead of determining their representational capacity for a finite sample of size n

Representational capacity of neural networks

UAT characterizes **representational capacity** of neural networks: set of hypotheses a given model is capable of expressing when assigning some value to its parameters

Main focus

i) Existence of approximations without explicit constructive method or explicit construction methods involving activation functions that are unusable in practice due to their complexity cf. [Maiorov-Pinkus,1999]

ii) Properties of approximations by analyzing their error for certain classes of functions provided that sufficiently many hidden units (N) are available

$\Rightarrow$ Characterizing W and L simultaneously in approximation rate remains open and challenging

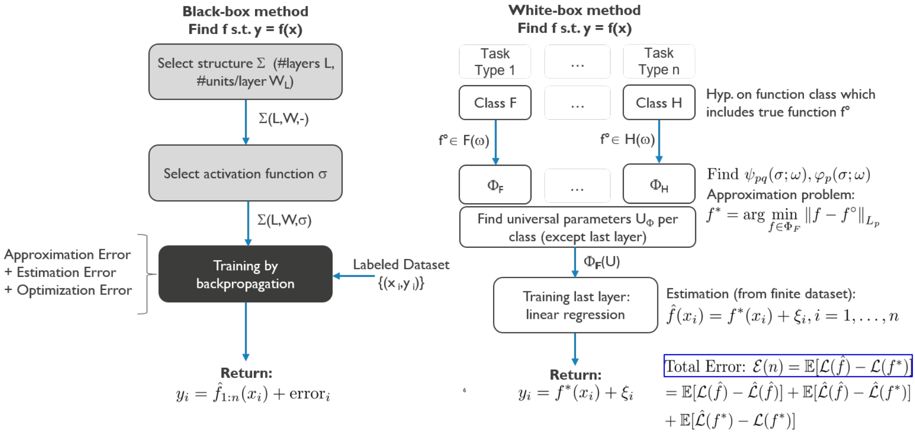
iii) Asymptotic estimates have limited applicability when properties of fixed neural structure of finite size need to be understood;

$\Rightarrow$ Very few results address quantitative and non-asymptotic approximation rate of neural networks

iv) Characterizing which functions (class of) NNs can express over entire domain instead of determining their representational capacity for a finite sample of size n

UAT neither address learnability of neural structure (depth, width) nor trainability of its parameters

Black-box vs. White-box Method



Black-box method
Find f s.t. y = f(x)

Select structure Σ (#layers L, #units/layer W_L)

Σ(L,W,-)

Select activation function σ

Σ(L,W,σ)

Approximation Error
+ Estimation Error
+ Optimization Error

Training by backpropagation

Labeled Dataset
$\{(x_i, y_i)\}$

Return:
$y_i = \hat{f}_{1:n}(x_i) + \text{error}_i$

White-box method
Find f s.t. y = f(x)

Task Type 1 … Task Type n

Class F … Class H

Hyp. on function class which includes true function f°

f° ∈ F(ω)

f° ∈ H(ω)

Φ_F … Φ_H

Find $\psi_{pq}(\sigma;\omega), \varphi_p(\sigma;\omega)$
Approximation problem:
$f^* = \arg\min_{f \in \Phi_F} \|f - f^\circ\|_{L_p}$

Find universal parameters U_Φ per class (except last layer)

$\Phi_F(U)$

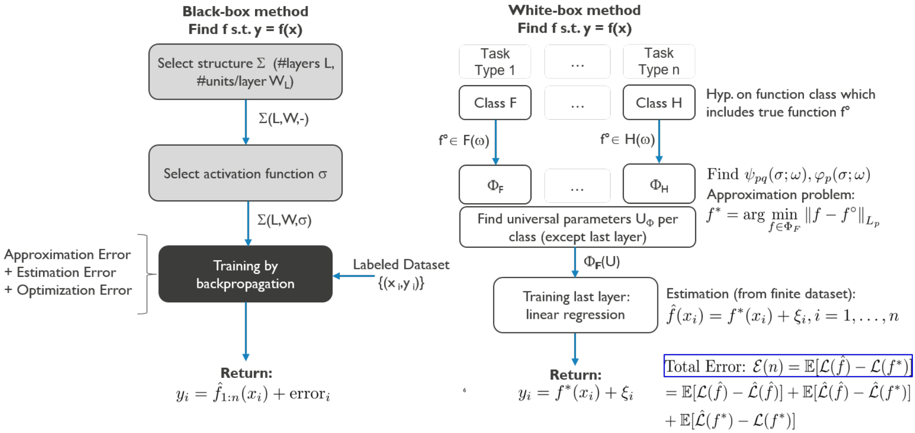
Training last layer: linear regression

Estimation (from finite dataset):
$\hat{f}(x_i) = f^*(x_i) + \xi_i, i = 1, \ldots, n$

Return:
$y_i = f^*(x_i) + \xi_i$

$$\boxed{\text{Total Error: } \mathcal{E}(n) = \mathbb{E}[\mathcal{L}(\hat{f}) - \mathcal{L}(f^*)]}$$
$$= \mathbb{E}[\mathcal{L}(\hat{f}) - \hat{\mathcal{L}}(\hat{f})] + \mathbb{E}[\hat{\mathcal{L}}(\hat{f}) - \hat{\mathcal{L}}(f^*)]$$
$$+ \mathbb{E}[\hat{\mathcal{L}}(f^*) - \mathcal{L}(f^*)]$$

Black-box vs. White-box Method



Black-box method
Find f s.t. y = f(x)

Select structure Σ (#layers L, #units/layer W_L)

Σ(L,W,-)

Select activation function σ

Σ(L,W,σ)

Training by backpropagation

Labeled Dataset $\{(x_i, y_i)\}$

Approximation Error
+ Estimation Error
+ Optimization Error

Return:
$y_i = \hat{f}_{1:n}(x_i) + \text{error}_i$

White-box method
Find f s.t. y = f(x)

Task Type 1 | … | Task Type n

Class F | … | Class H

Hyp. on function class which includes true function f°

$f^\circ \in F(\omega)$

$f^\circ \in H(\omega)$

Φ_F | … | Φ_H

Find $\psi_{pq}(\sigma;\omega), \varphi_p(\sigma;\omega)$
Approximation problem:
$f^* = \arg\min_{f \in \Phi_F} \|f - f^\circ\|_{L_p}$

Find universal parameters U_Φ per class (except last layer)

$\Phi_F(U)$

Training last layer: linear regression

Estimation (from finite dataset):
$\hat{f}(x_i) = f^*(x_i) + \xi_i, i = 1, \ldots, n$

Return:
$y_i = f^*(x_i) + \xi_i$

$$\boxed{\text{Total Error: } \mathcal{E}(n) = \mathbb{E}[\mathcal{L}(\hat{f}) - \mathcal{L}(f^*)]}$$
$$= \mathbb{E}[\mathcal{L}(\hat{f}) - \hat{\mathcal{L}}(\hat{f})] + \mathbb{E}[\hat{\mathcal{L}}(\hat{f}) - \hat{\mathcal{L}}(f^*)]$$
$$+ \mathbb{E}[\hat{\mathcal{L}}(f^*) - \mathcal{L}(f^*)]$$

Main motivations

- Constructive function approximation-based
- Training procedure with tight-error control
- Performance guarantees and verifiable models

Back box Gradient-based Methods: Error

Suppose

- Expected risk minimizer: $w^\star = \arg\min_{w \in W} \mathcal{L}(w) = \mathbb{E}_{(x,y)}[\ell(p(w,x),y)]$
- Empirical risk minimizer: $\hat{w} = \arg\min_{w \in W} \mathcal{L}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w,x_i),y_i)$

Compute minimizer $\hat{w}_\epsilon$ (accuracy ϵ) over n datapoints

- Approximation error: $E_{app} = \mathcal{L}(w^\star)$
- Estimation error: $E_{est}(n) = \mathbb{E}[\mathcal{L}(\hat{w}) - \mathcal{L}(w^\star)]$
- Optimization error: $E_{opt}(n,\epsilon) = \mathbb{E}[\mathcal{L}(\hat{w}_\epsilon) - \mathcal{L}(\hat{w})]$

Goal: $\min_{n,\epsilon} \mathbb{E}[\mathcal{L}(\hat{w}_\epsilon) - \mathcal{L}(w^\star)]$

- Objective function: $\mathbb{E}[\mathcal{L}(\hat{w}_\epsilon) - \mathcal{L}(w^\star)]$

$$= \mathbb{E}[\mathcal{L}(\hat{w}_\epsilon) - \mathcal{L}_n(\hat{w})] + \mathbb{E}[\mathcal{L}_n(\hat{w}) - \mathcal{L}_n(w^\star)] + \mathbb{E}[\mathcal{L}_n(w^\star) - \mathcal{L}(w^\star)]$$

Kolmogorov Representation Theorem

Kolmogorov Representation Theorem (1959)

Every continuous function $f : [0,1]^n \subset \mathbb{R}^n \to \mathbb{R}$ defined on $n-$dimensional space ($n \geq 2$) can be completely determined by sums and superpositions[a] of continuous functions φ_q ($q = 1, \ldots, 2n+1$) of one variable and ψ_{pq} ($p = 1, \ldots, n$, $q = 1, \ldots, 2n+1$) of variable x_p

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \varphi_q\Big(\sum_{p=1}^{n}(\psi_{pq}(x_p)\Big) \qquad (11)$$

with

$2n+1$ continuous outer functions $\varphi_q : \mathbb{R} \to \mathbb{R}$ (dependent of f)

$2n^2 + n$ continuous inner functions $\psi_{pq} : [0,1] \to \mathbb{R}$ (independent of f)

$\Rightarrow$ only outer functions φ_q specific for given function f

[a] Thus, this theorem is often referred to as the Superposition Theorem

Kolmogorov Representation Theorem

Kolmogorov Representation Theorem (1959)

Every continuous function $f : [0,1]^n \subset \mathbb{R}^n \to \mathbb{R}$ defined on $n-$dimensional space ($n \geq 2$) can be completely determined by sums and superpositions[a] of continuous functions φ_q ($q = 1, \ldots, 2n+1$) of one variable and ψ_{pq} ($p = 1, \ldots, n$, $q = 1, \ldots, 2n+1$) of variable x_p

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \varphi_q\Big(\sum_{p=1}^{n} (\psi_{pq}(x_p)\Big) \qquad (11)$$

with

$2n+1$ continuous outer functions $\varphi_q : \mathbb{R} \to \mathbb{R}$ (dependent of f)

$2n^2 + n$ continuous inner functions $\psi_{pq} : [0,1] \to \mathbb{R}$ (independent of f)

$\Rightarrow$ only outer functions φ_q specific for given function f

[a] Thus, this theorem is often referred to as the Superposition Theorem

Applicability to Neural Networks ?

A priori attractive: representation of function requires a fixed number of nodes, polynomially increasing with dimensions of the input space n

Kolmogorov Representation Theorem: Variants

- Lorentz (1962):

  Outer functions $\varphi_q := \varphi$ for all q

  Inner functions $\psi_{pq} := \lambda_p \psi_q$ for rationally independent constants $\lambda_p \leq 1$

Kolmogorov Representation Theorem: Variants

- Lorentz (1962):

  Outer functions $\varphi_q := \varphi$ for all q

  Inner functions $\psi_{pq} := \lambda_p \psi_q$ for rationally independent constants $\lambda_p \leq 1$

- Sprecher (1965):

  Inner functions $\psi_{pq}(x_p) := \lambda^{pq} \psi_q$ with λ constant and ψ_q **Holder continuous**, monotonic increasing functions

Kolmogorov Representation Theorem: Variants

- Lorentz (1962):

  Outer functions $\varphi_q := \varphi$ for all q

  Inner functions $\psi_{pq} := \lambda_p \psi_q$ for rationally independent constants $\lambda_p \leq 1$

- Sprecher (1965):

  Inner functions $\psi_{pq}(x_p) := \lambda^{pq} \psi_q$ with λ constant and ψ_q **Holder continuous**, monotonic increasing functions

- Fridman (1967):

  Inner functions ψ_{pq} could be chosen to be **Lipschitz continuous**

  drawback: decomposition requires $2n + 1$ outer functions and $2n^2 + n$ inner function

Kolmogorov Representation Theorem: Variants

- Lorentz (1962):

  Outer functions $\varphi_q := \varphi$ for all q

  Inner functions $\psi_{pq} := \lambda_p \psi_q$ for rationally independent constants $\lambda_p \leq 1$

- Sprecher (1965):

  Inner functions $\psi_{pq}(x_p) := \lambda^{pq} \psi_q$ with λ constant and ψ_q **Holder continuous**, monotonic increasing functions

- Fridman (1967):

  Inner functions ψ_{pq} could be chosen to be **Lipschitz continuous**

  drawback: decomposition requires $2n + 1$ outer functions and $2n^2 + n$ inner function

- Sprecher (1972):

  Inner functions $\psi_{pq}(x_p) := \lambda^{p-1} \psi(x_p + \epsilon q)$ with ψ Holder continuous

Relation to Neural Networks

[Hecht-Nielsen, 1987]: interpretation of Kolmogorov theorem as three-layer NN

To represent $f : \mathbb{R}^n \to \mathbb{R}^m : (x_1, \dots, x_n) \to f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \varphi\Big(\sum_{p=1}^{n} (\psi_{pq}(x_p)\Big)$

i) n units (processing elements) in the first ($x-$input) layer: fanout units to distribute input x-vector components to the processing elements of next layer

ii) $2n+1$ units in the hidden / middle layer with transfer function ψ_q independent of f (except dim. n)

$$z_q = \sum_{p=1}^{n} \lambda^q \psi(x_p + \epsilon q) + q \qquad (12)$$

where λ real constant, $\epsilon > 0$ rational number ψ continuous real monotonic increasing functions belonging to satisfy the Holder condition $|\psi(x) - \psi(y)| \leq c\,|x - y|^\alpha$ for any $0 < \alpha \leq 1$

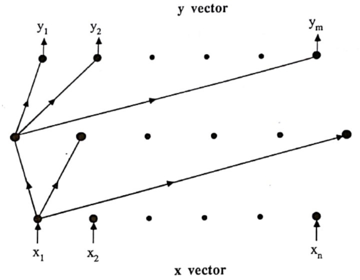
iii) m top layer/output units with transfer function φ dependent of f

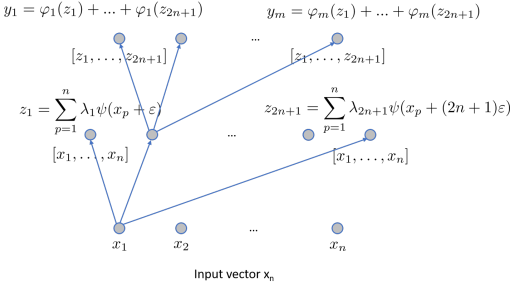
$$y_i = \sum_{q=1}^{2n+1} \varphi_i(z_q), i = 1, 2, \dots, m \qquad (13)$$

where functions $\varphi_i, i = 1, 2, \dots, m$ are real and continuous (dependent on f)

Example

Topology of the Kolmogorov Network[1]





$$y_1 = \varphi_1(z_1) + \ldots + \varphi_1(z_{2n+1})$$

$$y_m = \varphi_m(z_1) + \ldots + \varphi_m(z_{2n+1})$$

$[z_1, \ldots, z_{2n+1}]$ ... $[z_1, \ldots, z_{2n+1}]$

$$z_1 = \sum_{p=1}^{n} \lambda_1 \psi(x_p + \varepsilon)$$

$$z_{2n+1} = \sum_{p=1}^{n} \lambda_{2n+1} \psi(x_p + (2n+1)\varepsilon)$$

$[x_1, \ldots, x_n]$ ... $[x_1, \ldots, x_n]$

x_1 x_2 ... x_n

Input vector x_n

[1] R. Hecht-Nielsen, Kolmogorov's mapping neural network existence theorem, Proc. of Int'l Conf. on Neural Networks, vol.III, pp.11-14, New York: IEEE Press

Limited applicability

Girosi & Poggio (1989)

Interpretation inapplicable for two main reasons

- Inner functions ψ highly nonsmooth $\Rightarrow$ ψ at least as difficult to approximate as f
- Outer functions φ dependent on specific f $\Rightarrow$ not representable in parameterized form

Limited applicability

Girosi & Poggio (1989)

Interpretation inapplicable for two main reasons

- Inner functions ψ highly nonsmooth $\Rightarrow$ ψ at least as difficult to approximate as f
- Outer functions φ dependent on specific f $\Rightarrow$ not representable in parameterized form

Main consequences

1. Kolmogorov's Theorem cannot be used in any constructive way in neural learning context
2. Finding stable and exact representation of function f in terms of two- (or more) layer networks remains at least as difficult

Limited applicability

Girosi & Poggio (1989)

Interpretation inapplicable for two main reasons

- Inner functions ψ highly nonsmooth $\Rightarrow$ ψ at least as difficult to approximate as f
- Outer functions φ dependent on specific f $\Rightarrow$ not representable in parameterized form

Main consequences

1. Kolmogorov's Theorem cannot be used in any constructive way in neural learning context
2. Finding stable and exact representation of function f in terms of two- (or more) layer networks remains at least as difficult

However... finding good and well founded **approximate representations** was left open

Approximation of Representation Theorem

Kurkova's Approximation Theorem [Kurkova 1992]

- Derived from Kolmogorov's Representation Theorem
- NB: original proof of Kolmogorov's theorem is not constructive, i.e. shows existence of a representation but cannot be used in an algorithm for numerical calculations

Approximation of Representation Theorem

Kurkova's Approximation Theorem [Kurkova 1992]

- Derived from Kolmogorov's Representation Theorem
- NB: original proof of Kolmogorov's theorem is not constructive, i.e. shows existence of a representation but cannot be used in an algorithm for numerical calculations

Main idea

Relaxing exactness by an approximation of f (thus, approximate representation) enables estimating number of hidden units as a function of i) desired accuracy and ii) rate of increase of function f (modulus of continuity) being approximated

- One variable functions ϕ replaced by finite linear combinations of affine transformations of a single arbitrary sigmoidal function
- From UAT: sigmoidal functions can approximate any continuous function on any closed interval with arbitrary accuracy

Approximation of Representation: Kurkova Theorem

Let $S(\sigma)$ denote set of all staircase-like functions defined by $x \mapsto \sum_{i=1}^{k} a_i \sigma(b_i x + c_i)$

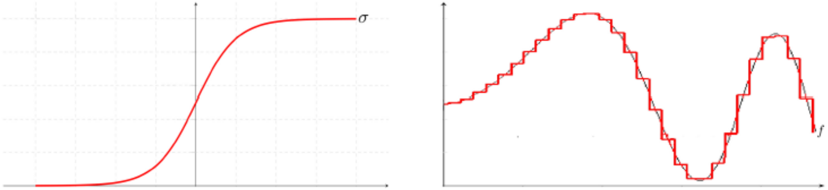


Figure 1: On the left, an activation function $\sigma : \mathbb{R} \to [0,1]$ and on the right a staircase-like function of a type σ which approximates f for the one dimensional case.

Kurkova Approximation Theorem

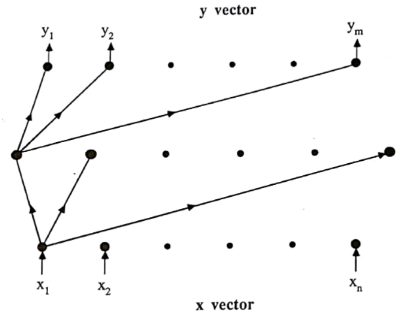
Let a continuous function $f : [0,1]^n \to \mathbb{R}$ and sigmoidal function $\sigma(\in S) : \mathbb{R} \to [0,1]$

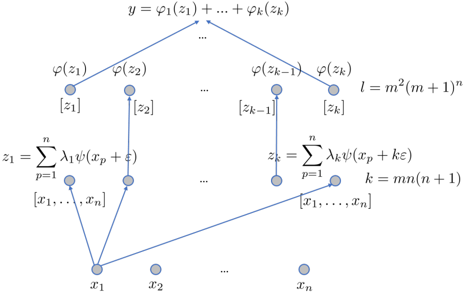
Then, for any $\varepsilon > 0$ there exist $k \in \mathbb{N}$ and functions $\varphi_q; \psi_{pq} \in S(\sigma)$ such that for any $(x_1, \ldots, x_n) \in [0,1]^n$

$$\left| f(x_1, \ldots, x_n) - \sum_{q=1}^{k} \varphi_q \left(\sum_{p=1}^{n} \psi_{pq}(x_p) \right) \right| < \varepsilon \qquad (14)$$

Example

Topology of the Kolmogorov Network[2]





$y = \varphi_1(z_1) + ... + \varphi_k(z_k)$

...

$\varphi(z_1)$ $\varphi(z_2)$ ... $\varphi(z_{k-1})$ $\varphi(z_k)$

$l = m^2(m+1)^n$

$[z_1]$ $[z_2]$ $[z_{k-1}]$ $[z_k]$

$z_1 = \sum_{p=1}^{n} \lambda_1 \psi(x_p + \varepsilon)$

$z_k = \sum_{p=1}^{n} \lambda_k \psi(x_p + k\varepsilon)$

...

$k = mn(n+1)$

$[x_1, \ldots, x_n]$ $[x_1, \ldots, x_n]$

x_1 x_2 ... x_n

Input vector x_n

[2] R. Hecht-Nielsen, Kolmogorov's mapping neural network existence theorem, Proc. of Int'l Conf. on Neural Networks, vol.III, pp.11-14, New York: IEEE Press

Approximation of Representation: Method

Constructive method: two-hidden layer networks with

- Universal set of weights for approximations of functions within a certain continuity class (Lipschitz, $\alpha-$Holder, etc.)

  $\Rightarrow$ Only weights from second hidden layer to output units are specific for the function f being approximated

- These weights appear linearly in parameterized expression

  $\Rightarrow$ **Training** becomes Linear regression problem

Approximation of Representation: Method

Constructive method: two-hidden layer networks with

- Universal set of weights for approximations of functions within a certain continuity class (Lipschitz, $\alpha-$Holder, etc.)

  $\Rightarrow$ Only weights from second hidden layer to output units are specific for the function f being approximated
- These weights appear linearly in parameterized expression

  $\Rightarrow$ **Training** becomes Linear regression problem

Guaranteed method: Kurkova Th. provides upper estimates of number of hidden units needed for $\varepsilon-$approximation of continuous functions

- For every $m \in \mathbb{N}$ such that $m \geq 2n + 1$, $n/(m-n) + v < \varepsilon / \|f\|_\infty$ for some $v > 0$
- Function f can be approximated with accuracy ε by two hidden layer neural network with activation function σ and

  $\rightarrow$ Number of units in first hidden layer: $nm(m+1)$

  $\Rightarrow$ Number of units in second hidden layer: $m^2(m+1)^n$

Approximation of Representation: Method

Constructive method: two-hidden layer networks with

- Universal set of weights for approximations of functions within a certain continuity class (Lipschitz, $\alpha-$Holder, etc.)

  $\Rightarrow$ Only weights from second hidden layer to output units are specific for the function f being approximated

- These weights appear linearly in parameterized expression

  $\Rightarrow$ **Training** becomes Linear regression problem

Guaranteed method: Kurkova Th. provides upper estimates of number of hidden units needed for $\varepsilon-$approximation of continuous functions

- For every $m \in \mathbb{N}$ such that $m \geq 2n+1$, $n/(m-n)+v < \varepsilon/\left\|f\right\|_\infty$ for some $v > 0$
- Function f can be approximated with accuracy ε by two hidden layer neural network with activation function σ and

  $\rightarrow$ Number of units in first hidden layer: $nm(m+1)$

  $\Rightarrow$ Number of units in second hidden layer: $m^2(m+1)^n$

Note: number of hidden units (2nd layer) needed for good approximations of general continuous functions exponential in n

Exercise

Finding good algorithms for constructing function approximators with

- (at least) two-hidden layer structure
- arbitrary activation functions, e.g., sigmoidal

Exercise

Finding good algorithms for constructing function approximators with

- (at least) two-hidden layer structure
- arbitrary activation functions, e.g., sigmoidal

Fact: number of neural units depend on

i) Desired approximation error

ii) Input dimensionality n

iii) Smoothness properties of f being approximated (start from Holder class or even stronger Lipschitz continuous)

Exercise

Finding good algorithms for constructing function approximators with

- (at least) two-hidden layer structure
- arbitrary activation functions, e.g., sigmoidal

Fact: number of neural units depend on

i) Desired approximation error

ii) Input dimensionality n

iii) Smoothness properties of f being approximated (start from Holder class or even stronger Lipschitz continuous)

Goal

Construction procedure to provide neural structure capable of approximating all functions from a certain class with given accuracy where only weights corresponding to output units should be learned

Exercise: Tasks

1. Upper estimates of number of hidden units needed for good approximations of general continuous functions remain very large. For particular classes of functions, better estimates could be obtained

   Identifying classes of functions (smoothness class: Lipschitz, Holder, Sobolev, etc.) to obtain better estimates remains to be formalized

Exercise: Tasks

1. Upper estimates of number of hidden units needed for good approximations of general continuous functions remain very large. For particular classes of functions, better estimates could be obtained

   Identifying classes of functions (smoothness class: Lipschitz, Holder, Sobolev, etc.) to obtain better estimates remains to be formalized

2. As single variable functions φ and ψ_{pq} involved in Kolmogorov's representation can be highly non-smooth and hard to compute, relaxing their rigidity to obtain a structure of function approximation remains to be further explored

Exercise: Tasks

1. Upper estimates of number of hidden units needed for good approximations of general continuous functions remain very large. For particular classes of functions, better estimates could be obtained

   Identifying classes of functions (smoothness class: Lipschitz, Holder, Sobolev, etc.) to obtain better estimates remains to be formalized

2. As single variable functions φ and ψ_{pq} involved in Kolmogorov's representation can be highly non-smooth and hard to compute, relaxing their rigidity to obtain a structure of function approximation remains to be further explored

3. Formulate training problem (i.e., parameter learning) as a regression problem on last layer instead of relying on backpropagation algorithm

   Evaluate and compare different regression methods to solve it depending on the properties of the problem

Exercise: Tasks

1. Upper estimates of number of hidden units needed for good approximations of general continuous functions remain very large. For particular classes of functions, better estimates could be obtained

   Identifying classes of functions (smoothness class: Lipschitz, Holder, Sobolev, etc.) to obtain better estimates remains to be formalized

2. As single variable functions φ and ψ_{pq} involved in Kolmogorov's representation can be highly non-smooth and hard to compute, relaxing their rigidity to obtain a structure of function approximation remains to be further explored

3. Formulate training problem (i.e., parameter learning) as a regression problem on last layer instead of relying on backpropagation algorithm

   Evaluate and compare different regression methods to solve it depending on the properties of the problem

4. Compare resulting models in particular their complexity vs. accuracy

   Guaranteeing performance should not come at the detriment of inflexible or over-constraining models; tradeoff between *Tractable but bad model or Intractable but good model*

Outline and Structure

1. Handling of NP-Hardness
2. Data space properties and Space complexity
3. Function Approximation and Approximate Representation
4. External penalty nonlinear ADM method

External penalty method: Nonlinear ADM

Composite minimization problem

$$\min_{x\in\mathcal{X}} \varphi(x) = f(x) + h(x) + g(c(x) - b)$$

- $(\mathcal{X}, \langle\cdot,\cdot\rangle)$ and $(\mathcal{Y}, \langle\cdot,\cdot\rangle)$ real Hilbert spaces, $b \in \mathcal{Y}$
- $c\colon \mathcal{X} \to \mathcal{Y}$ is a **nonlinear**, differentiable mapping
- $h\colon \mathcal{X} \to \mathbb{R}$ is a differentiable nonconvex function
- $f\colon \mathcal{X} \to]-\infty, +\infty]$ and $g\colon \mathcal{Y} \to]-\infty, +\infty]$ are proper lsc convex functions

Properties algorithmic scheme

- **Full-splitting method**: at each iteration evaluates separately
  - nonsmooth components f, g via their proximity operators
  - smooth component h via its gradient
  - nonlinear operator c via its values
- **No subsolver** (primal) ⇒ Lower computational cost
- Various OP can be reformulated as special cases of this generic model

Comparison: nonlinear ADM vs. alternatives

Composite minimization problem

$$\min_{x\in\mathcal{X}} \ \varphi(x) = f(x) + h(x) + g(c(x) - b)$$

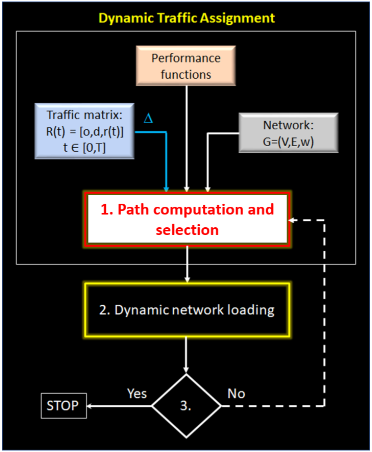
- $(\mathcal{X}, \langle\cdot,\cdot\rangle)$ and $(\mathcal{Y}, \langle\cdot,\cdot\rangle)$ real Hilbert spaces, $b \in \mathcal{Y}$
- $c\colon \mathcal{X} \to \mathcal{Y}$ is a **nonlinear**, differentiable mapping
- $h\colon \mathcal{X} \to \mathbb{R}$ is a differentiable nonconvex function
- $f\colon \mathcal{X} \to]-\infty, +\infty]$ and $g\colon \mathcal{Y} \to]-\infty, +\infty]$ are proper lsc convex functions

Method	**f**	**g**	**h**	**c**	**Splitting**	**Det.**	**Sto.**	**Subsolver**
Nonlinear ADM	cv	cv	nc-s	NL	yes	yes	yes	no
Valkonen 2014	cv	cv	0	NL	yes	yes	no	no
Bolte et al. 2018	0	cv	nc-s	NL	no	yes	no	yes
Linearized ADMs*	cv	cv	nc-s	L	yes	yes	yes	no
Prox-Linear*	cv	cv	0	NL	no	yes	yes	yes
iALMs*	0	cv	0	NL	no	yes	yes	yes

Legend: cv = convex, nc(-s) = non-convex (smooth), (N)L = (Non)Linear

Use case: multi-constrained Dynamic Traffic Assignment (DTA)

1. Method to compute network path-dependent throughput, delay, etc.
2. Method to determine how resulting path flows induce: time-dependent arc-load, load-induced delay, etc.
3. Determine if/when performance criteria are met


Dynamic Traffic Assignment
Performance functions
Traffic matrix: R(t) = [o,d,r(t)] t ∈ [0,T]
Δ
Network: G=(V,E,w)
1. Path computation and selection
2. Dynamic network loading
STOP
Yes
3.
No

Ref: D. Papadimitriou and B.C. Vu, "A penalized nonlinear ADM algorithm applied to the multi-constrained traffic assignment problem, Numerical Algorithms, 92(1), August 2022.

Use case: multi-constrained Dynamic Traffic Assignment (DTA)

1. Method to compute network path-dependent throughput, delay, etc.
2. Method to determine how resulting path flows induce: time-dependent arc-load, load-induced delay, etc.

   Idea: **by means of neural network**

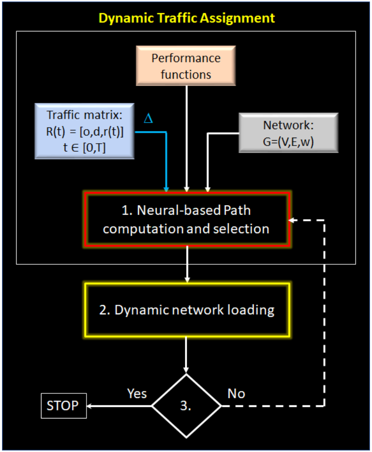
Use case: multi-constrained Dynamic Traffic Assignment (DTA)

1. Method to compute network path-dependent throughput, delay, etc.
2. Method to determine how resulting path flows induce: time-dependent arc-load, load-induced delay, etc.

   Idea: **by means of neural network**

   Challenging

   - available capacity: $c(u,v)$ $\min_{(u,v)\in Path} c(u,v) \rightarrow$ local (ok)
   - load-induced delay: $t(u,v)$ $\sum_{(u,v)\in Path} t(u,v) \rightarrow$ **non-local** (non-decomposable)
   - arc ↔ path dependent decisions
3. Determine if/when performance criteria are met



Dynamic Traffic Assignment
Performance functions
Traffic matrix: R(t) = [o,d,r(t)] t ∈ [0,T]
Δ
Network: G=(V,E,w)
1. Neural-based Path computation and selection
2. Dynamic network loading
STOP
Yes
3.
No

1. Multi-constrained Dynamic Traffic Assignment (DTA)

Given $\varphi_a := \varphi(\ell_a)$ load-dependent **utility function** (nonconcave)

- S-shaped utility function: risk seeking then after certain threshold risk averse
- Bandwidth utility function: inelastic real-time and delay-/rate-adaptive services

1. Multi-constrained Dynamic Traffic Assignment (DTA)

Given $\varphi_a := \varphi(\ell_a)$ load-dependent **utility function** (nonconcave)

- S-shaped utility function: risk seeking then after certain threshold risk averse
- Bandwidth utility function: inelastic real-time and delay-/rate-adaptive services

Given: Graph $G = (V, A)$, Path set $P = \cup_{k=1}^{n} P_k$, and Demand set $K = \{s_k, t_k, d_{st,k}, \tau_k\}_{k=1}^{n}$

Problem

$$\max \sum_{a \in A} \int_0^{\ell_a} \varphi_a(s) ds \tag{15}$$

s.t.

$$\sum_{p \in P_k} x_{stp} = d_{st} \qquad \forall d_{st} : (s,t) \in K \tag{16}$$

$$\sum_{(s,t) \in K} \sum_{p \in P_k} \delta_{stpa} x_{stp} = \ell_a \qquad \forall a \in A \tag{17}$$

$$0 \leq \ell_a \leq \kappa_a \qquad \forall a \in A \tag{18}$$

$$\sum_{a \in A} \delta_{stpa} \Big(\frac{1}{\kappa_a - \ell_a} + t_a \Big) \leq \tau_k \qquad \forall k \in K, p \in P_k, x_{stp} > 0 \tag{19}$$

$$(d_{st} \geq) \, x_{stp} \geq 0 \qquad \forall (s,t) \in K, p \in P \tag{20}$$

S-shaped utility function

Nonconvex (S-Shaped) **disutility function**: measures disutility (φ_a) of using given arc $a \in A$ as function of carried traffic flow or load ℓ_a, $\forall a \in A$

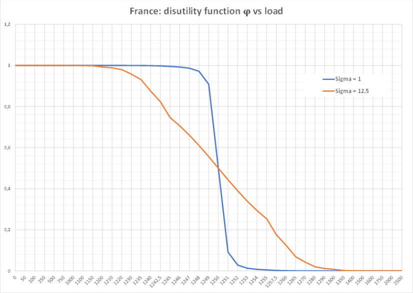
$$\varphi_a(s) = \frac{1}{2}\Big(1 - \frac{((s - \kappa_a/2)/\sigma_a}{\sqrt{1/2 + ((s - \kappa_a/2)/\sigma_a)^2}}\Big) \tag{21}$$

S-shaped utility function

Nonconvex (S-Shaped) **disutility function**: measures disutility (φ_a) of using given arc $a \in A$ as function of carried traffic flow or load ℓ_a, $\forall a \in A$

$$\varphi_a(s) = \frac{1}{2}\Big(1 - \frac{((s - \kappa_a/2)/\sigma_a}{\sqrt{1/2 + ((s - \kappa_a/2)/\sigma_a)^2}}\Big) \tag{21}$$

Topology	Scaling factor σ	Objective Function
austria	1108.00	$\frac{1}{2}\Big(1 - \frac{(s-5544000)/1108}{\sqrt{1/2+((s-5544000)/1108)^2}}\Big)$
atlanta	20.00	$\frac{1}{2}\Big(1 - \frac{(s-2000)/20}{\sqrt{1/2+((s-2000)/20)^2}}\Big)$
cost266	604.80	$\frac{1}{2}\Big(1 - \frac{(s-60480)/604.80}{\sqrt{1/2+((s-60480)/604.80)^2}}\Big)$
france	12.50	$\frac{1}{2}\Big(1 - \frac{(s-1250)/12.5}{\sqrt{1/2+((s-1250)/12.5)^2}}\Big)$
germany50	1	$\frac{1}{2}\Big(1 - \frac{(s-20)}{\sqrt{1/2+((s-20)^2}}\Big)$
giul39	3.20	$\frac{1}{2}\Big(1 - \frac{(s-320)/3.20}{\sqrt{1/2+((s-320)/3.20)^2}}\Big)$
india35	3.00	$\frac{1}{2}\Big(1 - \frac{(s-300)/3.00}{\sqrt{1/2+((s-300)/3.00)^2}}\Big)$
norway	20.00	$\frac{1}{2}\Big(1 - \frac{(s-2000)/20}{\sqrt{1/2+((s-2000)/20)^2}}\Big)$
pioro40	3.11	$\frac{1}{2}\Big(1 - \frac{(s-311)/3.11}{\sqrt{1/2+((s-311)/3.11)^2}}\Big)$



Model Reformulation

Reformulation

- Define variable

$$y_a = \frac{\kappa_a}{\kappa_a - \ell_a}, \ \forall a \in A \tag{22}$$

- Load variable ℓ_a becomes

$$\ell_a = \kappa_a \Big(1 - \frac{1}{y_a}\Big), \forall a \in A \tag{23}$$

- Thus, constraints reformulated as (bilinear $\sim x_{stp} y_a$)

$$\begin{array}{lr}
\sum_{p \in P_k} x_{stp} = d_{st} & \forall d_{st} : (s,t) \in K \\
y_a \Bigg(\sum_{(s,t) \in K} \sum_{p \in P_k} \delta_{stpa} x_{stp} - \kappa_a \Bigg) = -\kappa_a & \forall a \in A \\
\sum_{a \in A} \delta_{stpa} \Big(\frac{y_a}{\kappa_a} + t_a \Big) \leq \tau_k & \forall k \in K, p \in P_k : x_{stp} > 0 \\
(d_{st} \geq) \, x_{stp} \geq 0 & \forall (s,t) \in K, p \in P \\
(M \geq) \, y_a \geq 1 & \forall a \in A
\end{array}$$

Instances

Network Topologies and Properties
From SNDLib http://sndlib.zib.de/problems.overview.action

Topology	**Nodes**	**Arcs**	**Min, Max, Avg Degree**	**Diameter**	**Demands**
austria	65	216	1, 10, 3.32	9	1868
atlanta	15	44	2, 4, 2.93	5	210
cost266	37	114	2, 5 3.08	8	1332
france	25	90	2, 10, 3.60	5	300
germany50	50	176	2, 5, 3.52	9	662
giul39	39	172	3, 8, 4.41	6	1471
india35	35	160	2, 9, 4.57	7	595
norway	27	102	2, 6, 3.78	7	702
pioro40	40	178	4, 5, 4.45	7	780
zib54	54	81	1, 10, 3.00	8	1501

Instances

Network Topologies and Properties
From SNDLib http://sndlib.zib.de/problems.overview.action

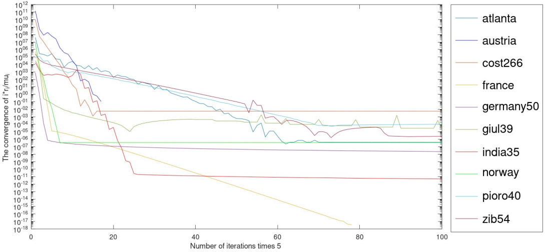
Topology	Nodes	Arcs	Min, Max, Avg Degree	Diameter	Demands
austria	65	216	1, 10, 3.32	9	1868
atlanta	15	44	2, 4, 2.93	5	210
cost266	37	114	2, 5 3.08	8	1332
france	25	90	2, 10, 3.60	5	300
germany50	50	176	2, 5, 3.52	9	662
giul39	39	172	3, 8, 4.41	6	1471
india35	35	160	2, 9, 4.57	7	595
norway	27	102	2, 6, 3.78	7	702
pioro40	40	178	4, 5, 4.45	7	780
zib54	54	81	1, 10, 3.00	8	1501

- Traffic demands (synthetic): variable size d_{st} and max.delay τ_k
- Path classes : for each demand $\{s_k, t_k, d_{st,k}, \tau_k\} \in K$ obtain set P_k of candidate paths

  Example: a) shortest path (hop-count), b) min.weight/least cost path, c) max.available bandwidth/least loaded path, and d) min.residual/unused bandwidth

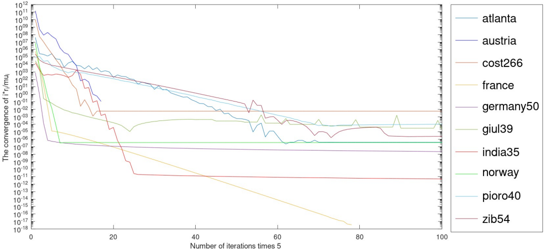
Numerical Method

Nonlinear ADM: convergence vs. number of iterations



Numerical Method

Nonlinear ADM: convergence vs. number of iterations



Comparison with commercial **MILP solver**

- Reformulation Linearization Technique (RLT)-based CR with variable $z_{stpa} = x_{stp} y_a$ such that bilinear constraints

$$\kappa_a y_a - y_a \left(\sum_{(s,t)\in K} \sum_{p \in P_k} \delta_{stpa} x_{stp} \right) = \kappa_a \to \quad \kappa_a y_a - \sum_{(s,t)\in K} \sum_{p \in P_k} \delta_{stpa} z_{stpa} = \kappa_a (\forall a \in A)$$

- PWL approx. of disutility function

Numerical Results

Disutility minimization - Nonlinear ADM (32GB/4)

Topology	**Minimum**	**Feasibility**	**Null Load**	**Computation**
	Obj. value		**Arcs**	**Time (s)**
austria	2099892	7	15	3935
atlanta	**55404**	34	0	198
cost266	**234019**	0.009966	0	913
france	79662	2	0	298
germany50	1478	0.0015	1	**129**
giul39	46001	5	0	1821
india35	**32507**	5	1	471
norway	43623	8	0	679
pioro40	198780	95	1	890
zib54	128756	9	19	1038

Disutility minimization - MILP Solver (32GB/4)

Topology	**Minimum**	**Gap**	**Null Load**	**Comp.**	**Total**
	Obj. value	**Optimal.**	**Arcs**	**Time (s)**	**Time (s)**
austria	**1641854**	0.000%	15	16.54	412.03
atlanta	63887	0.000%	0	59.07	60.56
cost266	317153	0.000%	0	58.44	153.95
france	**69115**	0.000%	0	31.80	38.20
germany50	**1448**	0.005%	4	**6790.86**	6911.11
giul39	**42472**	0.000%	0	851.81	948.17
india35	32969	0.250%	3	60.72	90.45
norway	**42455**	0.000%	0	538.37	563.24
pioro40	**156362**	0.160%	2	136.05	188.38
zib54	**105080**	0.000%	19	105.45	331.31