

# Novel Applications of AI Techniques in Database Management

Johannes Wünsche

July 17, 2018

# Overview of AI Techniques in Database Management

# Overview of AI Techniques in Database Management

1	Natural Language Interfaces	1	Data Management Applications
2	Self-Management of Databases		✓
1	Machine Learning in Databases		
3	Optimization by "Software 2.0"		✓

# Agenda

- 1 Overview of AI Techniques in Database Management
- 2 Short Summary of Natural Language Interfaces, ML in DB and Data Management Applications
- 3 Self-Management of Databases
- 4 Optimization by “Software 2.0”

## Short Summary of Natural Language Interfaces, ML in DB and Data Management Applications

# Natural Language Interfaces & ML in DB

## Natural Language Interfaces

- Idea of using AI to interpret NL questions or requests and react accordingly
- Quite old concept improved by modern hardware and technology

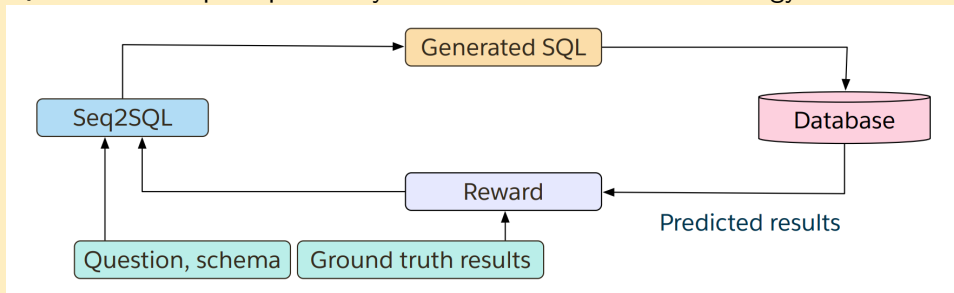


Figure 1: *SEQ2SQL: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE USING REINFORCEMENT LEARNING*, Zhong et al

## Machine Learning in Databases

- Machine learning methods on data
- Reduces data movement by putting libraries in the database
- e.g. 2010s Parameter Server(distributed training), Apache Spark(framework for distribution of tasks for analytics), '*Apache MADlib*, which implements
  - bayes classifier
  - clustering
  - association rules
  - ...
- or 2000s *scikit-learn*(easy to use ML-library)
- 90s *R* and *Weka*

# Data Management Applications

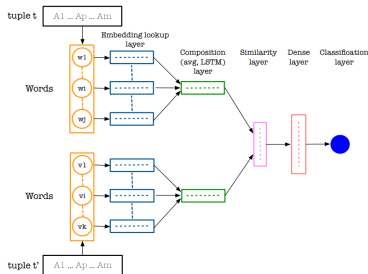


Figure 2: *DeepER-Deep Entity Resolution, Muhammad Ebraheem et al*

## Entity Resolution

- Finding of Records that refer to the same entity
- Required if data does not have a single representation



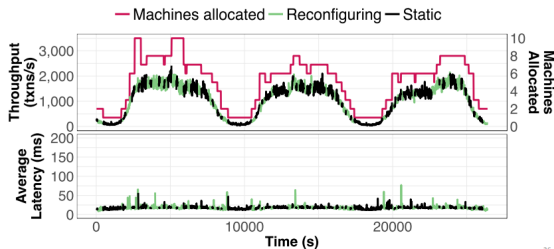
## Self-Management of Databases

# Self-Management of Databases

- *Elastic Scaling of Machine Allocation*
  - avoid latency spikes by action prediction through time-series prediction
  - implementations like P-Store by Taft, MIT

Results – P-Store with SPAR

Avg. Machines Used: 5.05



26

Figure 3: <http://www.hpts.ws/papers/2017/taft.pdf>

# Self-Management of Databases

- *Tuning of parameters of Database Management Systems*
  - like cache amount and frequency of writing to storage
  - implementations like OtterTune *by Database Research Group at Carnegie Mellon University*

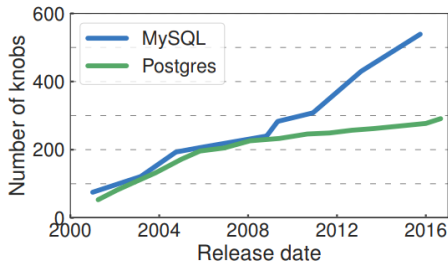
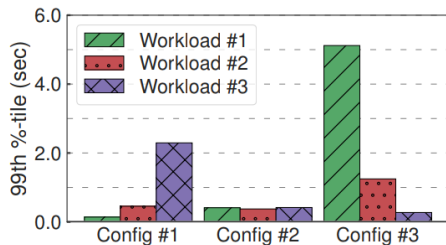


Figure 4: *Automatic Database Management System Tuning Through Large-scale Machine Learning*, Van Aken et al

# Self-Management of Databases

- *Peloton - Self Driving Database*
  - replacement of human database manager

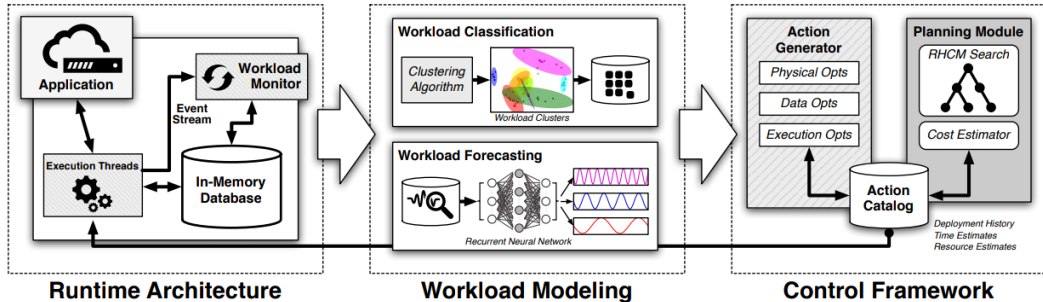


Figure 5: *Self-Driving Database Management Systems, Pavlo et al*

# Overview of AI Techniques in Database Management

1	Natural Language Interfaces	1	Data Management Applications
2	Self-Management of Databases		✓
1	Machine Learning in Databases		
3	Optimization by "Software 2.0"		✓

# Learned Index Structures

## *Learned Index Structures*

- a “model can learn the sort order or structure of lookup keys and use this signal to effectively predict the position or existence of records”
- alternative technology to existing Bloom-Filters or B-Trees

# Learned Index Structures

What's the difference between Learned Index Structures and existing Approaches?

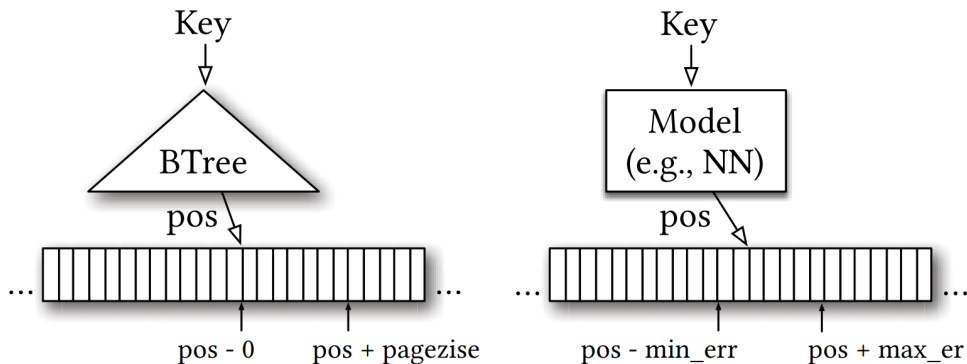


Figure 6: *The Case for Learned Index Structures by Kraska et al*

# Learned Index Structures

- Performance of Index Access can be unintuitively enhanced by predicting the index of a searched instance with a Neural Network or Linear Regression
- Shown to result in equally good or better performance than conventional Index Structures



# Learned Index Structures by Example

## Searching the index of a key

- Similar to every other model used
- Transform key to vector and use as input for trained NN
- Result will be the index of the searched key
- Effective for read-only database

## Inserting or Updating a key

### ■ *Append*

- possible but only in a few cases
- like adding sequential logs in order
- LIS can learn the pattern also for future data
- appending reduced to  $O(1)$
- to compare B-Trees need in any case  $O(\log n)$

### ■ *Insert in the middle*

- theoretically possible, but practically difficult
- can similarly be prelearned
- but moving of data or reservation of space might be required

# Learned Index Structures Result

Type	Config	Map Data			Web Data			Log-Normal Data		
		Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

Figure 7: *The Case for Learned Index Structures by Kraska et al*

# Enhancement of LIS: Recursive Model Index(RMI)

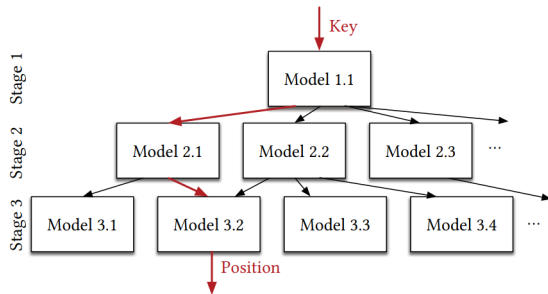


Figure 8: *The Case for Learned Index Structures by Kraska et al*

- Refinement of models after every step
- Easier to achieve than one large model
- After one layer is completed the result will be moved to the according lower level model until a leaf is reached
- Enable mixture of different technologies to optimize the model
  - NN
  - Linear Regression
  - B-Trees
  - ...
- They are also applicable for hashmaps and Bloomfilter replacements

# RMI Example

## 1 Classification

- 1 Use key as input
- 2 Calculate output of the classification neural network
- 3 1 - 12 - 8 - 4 (Example Construction)
- 4 Apply sigmoid to output → highest value model get chosen

## 2 Linear Regression

- 1 1 - 8 - 1 Auto Encoder
- 2 Result is predicted position

# Learned Index Structures Conclusion

## Advantages

- Can take advantage of real world data patterns(ML) → allows for high optimization
- Lower engineering costs
- Can lead to quicker adjustment of models during runtime

## Disadvantages

- Initial work on B-Trees and alike is lower since they do not require additional training
- Some open questions still remain as well as long term performance tests in real world systems

Optimization by “Software 2.0”



# Intro

- Major downsides like difficult optimization of code and human error in classical software development
- Doesn't base on declarative programming and tries to learn the desired functionality with a approximate base net
- Enabled by development of Neural Networks in last 20 years allowing  $>100$  layers deep networks
- Program space is restricted for future training (backpropagation, gradient descent)
- Many real world problems easier to detect desirable behaviour than to write a specific program

# Advantages

## Higher Portability

- Smaller operation set
- Matrix Multiplication and thresholding at zero required
- Small instruction set of chips with pretrained nets allows for cheap and specialised hardware

## Better Performance

- Allows for better performance and correctness predictions because closer implementation in hardware less core primitives are needed
- Modules can be introduced to a single module reducing communication overhead by sacrificing clarity of separation, which is due to the human unlike nature of S2.0 sacrificed beforehand either way
- Well trained neural nets outperform code implementation

# Advantages

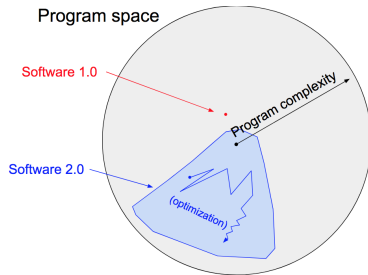


Figure 9: <https://medium.com/@karpathy/software-2-0-a64152b37c35>, Karpathy

## Better Runtime Predictability

- Requires same amount of memory each iteration → low probability of infinite loops or locks
- Speed well adjustable → speed can easily improved by reducing performance

# Disadvantages

## Unintuitivity

- Can be treated as different new paradigm → requires rethinking of development style
- Even though the network may work well, for humans difficult to understand
- Developing S2.0 is unintuitive and not well developed
- Requires manually curating, maintaining, cleaning and labeling of datasets
- May not be applicable easily to all problems

# Disadvantages

## Nonrecognizable errors

- Errors may occur unpredictable
- Can silently fail due to changed biases (hard to track since a large amount of them are being trained)

## Lack of Tools

- No tools currently exist that support the development process like IDE, highlighting and alike as for classical software

## Low Experience

- Effective in some implementations but not as general approach

# Conclusion

## Optimization

- Allows for greater optimization of specific complex problems
- e.g. Cuttlefish achieves 7.5x speedup to other query optimizers *Cuttlefish: A Lightweight Primitive for Adaptive Query Processing*

## Development difficult

- No Tools and unexperienced developers
- Generally low usage experience

Thank you for your attention.

Do you have any questions? Ideas?  
Be free to ask them.

## Sources

KRASKA, Tim, et al. The case for learned index structures. In: Proceedings of the 2018 International Conference on Management of Data. ACM, 2018. S. 489-504.a

KAFTAN, Tomer, et al. Cuttlefish: A Lightweight Primitive for Adaptive Query Processing. arXiv preprint arXiv:1802.09180, 2018.

EBRAHEEM, Muhammad, et al. DeepER—Deep Entity Resolution. arXiv preprint arXiv:1710.00597, 2017.

<https://medium.com/@karpathy/software-2-0-a64152b37c35> - Andrej Karpathy  
Software 2.0

<http://madlib.apache.org/> - Apache MadLIB Information