Block2Vec: A Deep Learning Strategy on Mining Block Correlations in Storage Systems

Dong Dai [1], Forrest Sheng Bao [2], Jiang Zhou [1], and Yong Chen [1]

[1]Dept. of Computer Science, Texas Tech University

[2]Dept. of Electrical and Computer Engineering, University of Akron

August 16, 2016

Introduction and Motivation

- Running applications generate tons of I/O access traces.
- Can we consider I/O traces as languages that applications speak?

```
anarchism originated as a term of abuse first used against early worki
ng class radicals including the diggers of the english revolution and t
he sans culottes of the french revolution whilst the term is still used
```

(a) An English Sentence Example

```
7815409:R 7815265:R 2966507:R 2966507:R 7815411:R 2966460:W 2966286:W
2966286:W 2966460:W 7815535:R 7815597:R 7815473:R 770052:W 770053:W 770
052:W 7815559:R 770053:W 2966718:R 7815559:R 2967007:R 7815221:R 781555
```

(b) A Block Access Log Example

Introduction and Motivation

- Let's talk about the similarity
  - Both human languages and I/O traces are sequence of symbols.
  - Language are constructed by words; I/O traces are built by a combination of Block+Operation.
  - Language has certain rules reflected by how words are organized, I/O traces reveal the inner relationships of data reflected by how the blocks are accessed.
- There are also differences
  - System I/O traces are generated by lots of diverse applications.
  - The vocab size is much larger than normal languages (1B v.s. 1M).
  - More randomness and complexity.
- Map I/O traces as language, how far can we go from here?
  - Our first try, **Block2Vec**, to learn semantic similarity of data blocks.
  - A combination of Natural Language Processing and deep learning.

Block Correlations

Similarity of Blocks

- Correlations between blocks come from several sources:
  - Physically sequential blocks are more likely to be accessed together.
  - Temporarily correlations: like inode blocks and its data blocks.
  - Multiple applications interact with each other (workflow).
- Correlated blocks are:
  - *Accessed relatively close to each other in an access stream*
- The challenges are:
  - How to quantitively measure the correlations of two blocks? As there are too many features:
    - blocks belong to the same file, in the same directory?
    - addresses are sequential?
    - accessed by the same application?
    - created by the same users?

$\rightarrow$ Block Access Model

Block Correlations

Statistic Block Access Model

- The statistical block access model describes the probability of accessing a specific group of blocks together:

$$P(b_1, \ldots, b_{m-1}, b_m)$$

- It can be calculated based on conditional probability.

$$\begin{aligned} P(b_1, \ldots, b_m) &= P(b_1, \ldots, b_{m-1}) \cdot P(b_m | b_1, \ldots, b_{m-1}) \\ &= \prod_{i=1}^{m} P(b_i | b_1, \ldots, b_{i-1}) \end{aligned}$$

- Due to computation complexity, it can be approximated using n proceeding blocks (N-gram).

$$\prod_{i=1}^{m} P(b_i | b_{i-n+1}, \ldots, b_{i-1})$$

Block Correlations

Statistic Model and Vector Representation

- Directly calculating statistic block access model has several limitations, like its cousin: statistic language model.
  - **Curse of dimensionality**
    - a statistic block access model with 10 consecutive block accesses in a disk with blocks of size 100,000 leads to $100000^{10} - 1 = 10^{50} - 1$ possible combinations.
  - **Vanished probability**
    - if a sequence of block accesses never shows before, the statistic model gives its probability as 0, which is not correct as the new combinations are likely to occur again and will occur more frequently with longer context N.
- In this research, for the first time, we propose to use a vector instead of an index to represent a block.

Block Correlations

Vector Representation

- Associate each block with a continuous-valued vector, instead of only representing the physical location through block Id.
- The similarity or correlation of two blocks can be calculated as their vector distance.
- It has several significant advantages:
  - Vector can indicate a tuple of features that characterize the block. For example, the physical location of block, the file it belongs to, the user who owns the file, the creation time, the access time, etc.
  - It overcomes the dimension explosion problem. The number of features (i.e., the vector dimension) is much smaller than the number of blocks.
  - The vector representation supports access sequences that have not yet shown in history, but are similar to existing ones in terms of their features.
- **How to obtain the vector representation for each vector?**

Obtain Vector Representation

Deep Neural Network and Word Embedding

- Deep learning is a rapidly emerging machine learning technique extended from artificial neural network (ANN):
  - The word "deep" comes from the fact that such an extended ANN normally has multiple hidden layers forming a deep hierarchy as compared to traditional ANN.
  - The word "deep" also comes from the fact that it can perform complicated transformations to reach a high-level abstraction of data.
- One of the most exciting uses of deep learning in natural language processing (NLP) is **Word Embedding**[1]
  - It maps words to high-dimensional vectors of real numbers by training a deep artificial neural network
  - We use the similar techniques to obtain vector representations of blocks.

[1] Yoshua Bengio et al. "Neural Probabilistic Language Models". In: *Innovations in Machine Learning*. Springer, 2006, pp. 137–186.

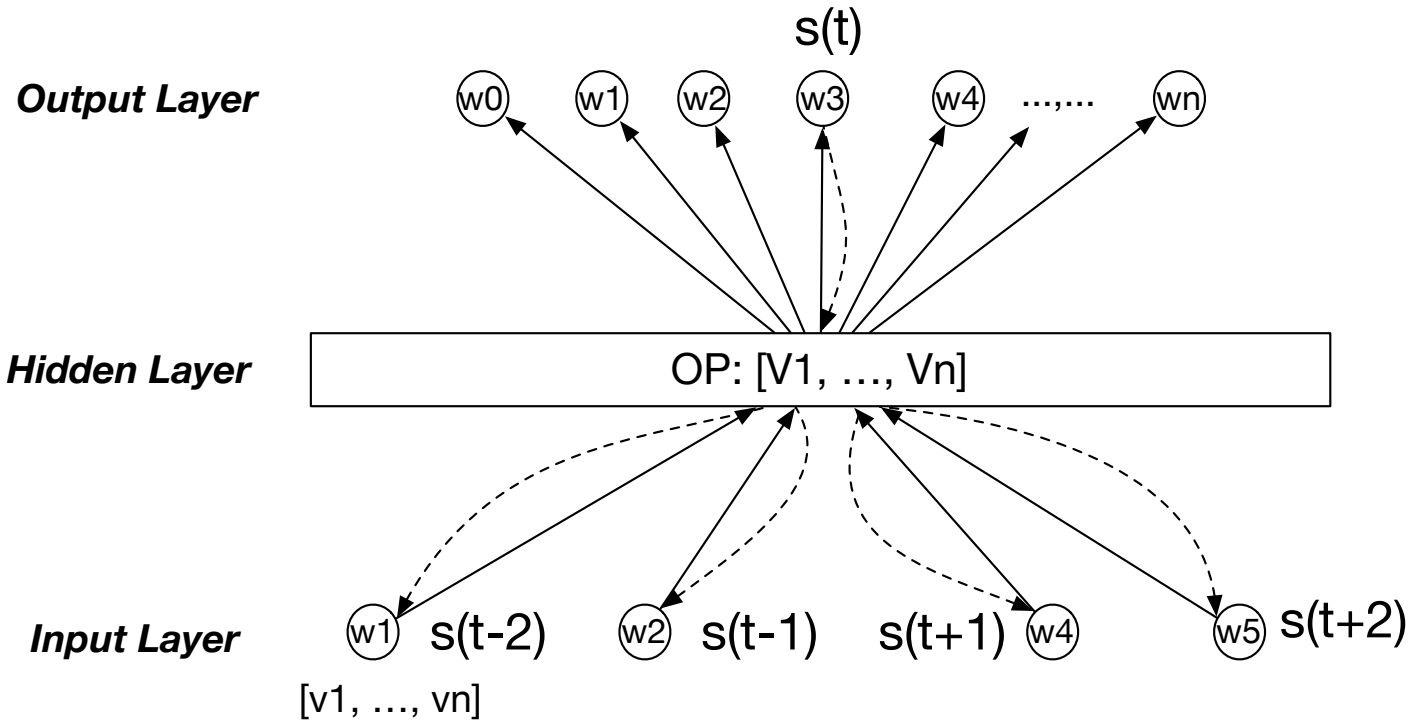


Figure : A neural network example for calculating word embedding for English words (e.g., top 50,000) by solving the "filling-the-blank" problem. The training is done by using numerous 5-word sequences, e.g., "*the cat sits on mat*", from English text. The input is 4 words in the 5-word sequence except the middle word, e.g., "*the cat ___ on mat*". Each neuron in the output layer corresponds to one of the *n* words. A correct prediction will yield the highest probability for the word "*sits*". A wrong prediction will trigger the neural network to update vector representation of words and the weights between neurons.

Block2Vec Design and Implementation

Data Pre-processing: Cutting Window

- We have system-level block I/O access traces. Need to turn them into multiple relevant "sentences" to train the network
  - Block2Vec uses access time distance to divide block access trace.
  - The maximal distance threshold is denoted as cutting window max_{win}.
  - Once a block access happens later than max_{win} after the previous one, we consider a new "sentence" has started.

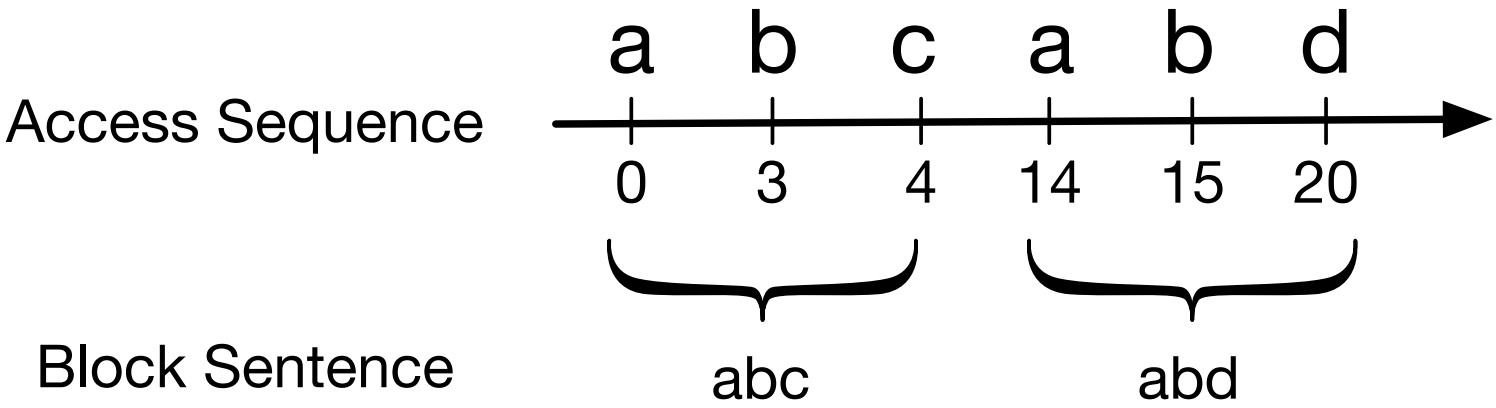


Figure : Splitting block sequences into multiple block sentences. The maximal cutting window is 5 time units in this example.

Block2Vec Design and Implementation

Data Pre-processing: Block Access Translation

- in real world block accesses, each block operation may access various sizes of data

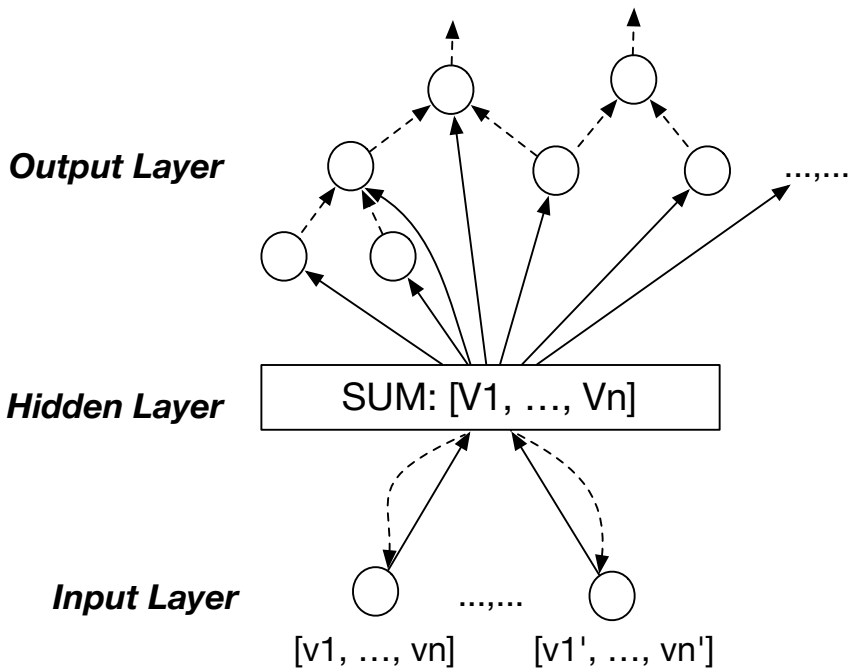
```
128166386533632472,hm,0,Write,11812225024,4096,2099
128166386555128520,hm,0,Read,329748480,1536,68413
128166386555194255,hm,0,Write,3163787264,28672,2679
128166386555196007,hm,0,Write,3154132992,4096,927
128166386555300619,hm,0,Read,11933585408,4096,208813
128166386555454261,hm,0,Read,579637248,32768,55170
128166386555463626,hm,0,Read,580390912,32768,45806
```

- Block2Vec uses an aggregated way to translate block access: only one block Id is included in the final sequence no matter how many bytes are read from that block.
- Block2Vec differentiate read/write block operations by appending an *operation* bit at the end of their identifications.

Block2Vec Design and Implementation

Neural Network Architecture

- Block2Vec's neural network architecture includes:
  - The input layer takes k recent block accesses to train each time; each block is represented as an n-dimensional vector $[v_1, ..., v_n]$.
  - The hidden layer **sums** all input vectors into a single vector.
  - Each dimension of such vector connects to the output layer, which is a Huffman tree.



Block2Vec Design and Implementation

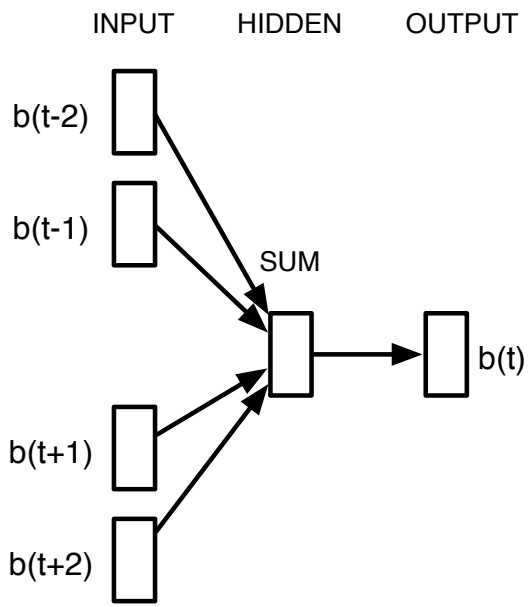
Neural Network Explaination

- Build the output layer as Huffman tree
  - The Huffman tree is built from a full scan of the entire block accesses.
  - All nodes have different path lengths to the root of the tree depending on their occurring frequencies.
- Purge noises
  - Infrequent blocks of fewer than min_f occurrences are ignored as correlations involving these rarely accessed blocks are considered not stable.
  - In this work, we use an empirical value $min_f = 5$ for a good balance between complexity and block coverage.
- Training Process
  - Vector representation of each block and all weights are randomly initialized to generate outputs for each input;
  - If the outputs are not correct, a back-propagation will happen to adjust all weights from the hidden layer to the input layer;

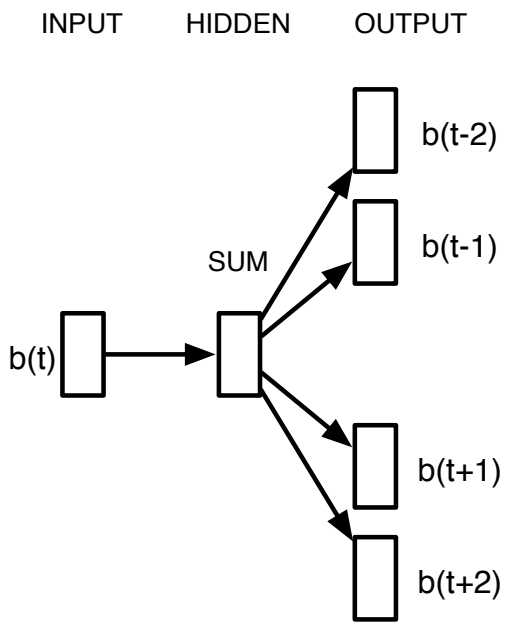
Block2Vec Design and Implementation

Two Training Models

- *Block2Vec* actually contains two different training models, both of which have been widely used in NLP
  - *Continuous Bag-of-Words* (CBOW): it tries to correctly classify the current (middle) block given several future and history blocks.
  - *Skip-gram* model: it tries to predict the context given a single input.



a) CBOW Model Architecture



b) Skip-gram Model Architecture

Block2Vec Design and Implementation

Time Sensitive Training

- Consider the time sensitivity of block accesses
  - Time interval between block access (in max_{win}) is still important: a longer interval indicates higher possibility of irrelevant data accesses.
  - Existing word embedding training (*word2vec*) uses random sampling strategy, which only considers the order of two block accesses.
  - Block2Vec considers the elapsed time between block accesses in the context window.

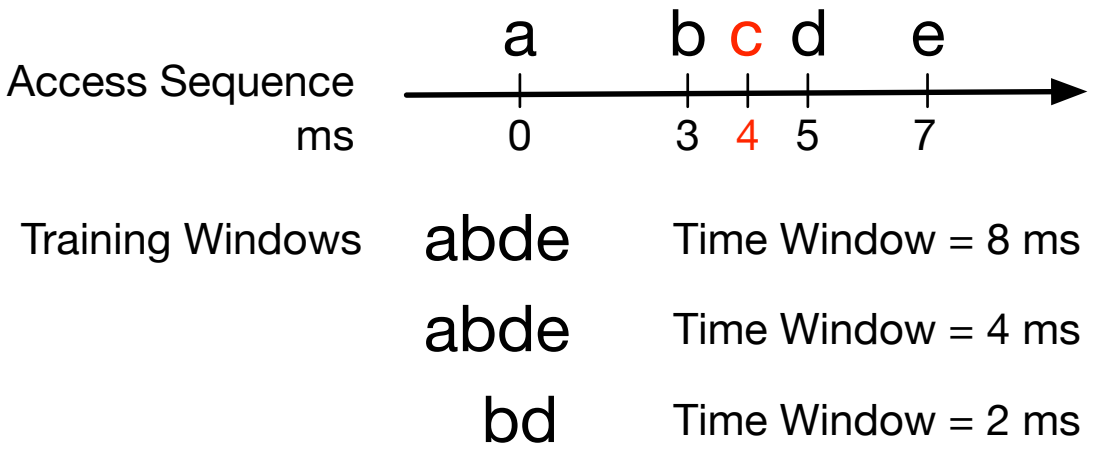
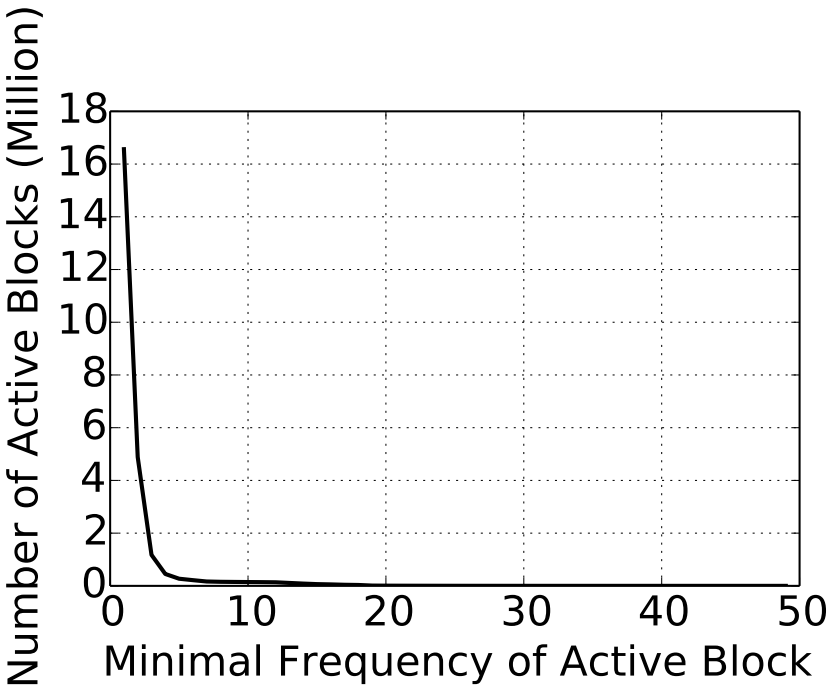


Figure : Time sensitive training in Block2Vec.

Block2Vec Design and Implementation

Efficiency of Block2Vec

- Memory consumption
  - Memory consumption depends on the size of *active blocks*, which have high access frequency (larger than min_f) in the trace.
  - Luckily, the number of active blocks drops significantly as the minimal frequency increases.



Block2Vec Design and Implementation

Efficiency of Block2Vec

- Computation Complexity
  - The training complexity of CBOW model is $\mathcal{O}(N \times D + D \times \log_2 V)$[2].
  - The training complexity of Skip-gram model is $\mathcal{O}(N \times (D + D \times \log_2 V)$
  - Here, N is the size of the context window, D is the dimension of vector, and V is the size of active blocks.
- Computation complexity of probability graph and frequency miner like C-Minder are both $N \times V$.
- Building the Huffman tree is not considered a part of the training phase. It actually costs $O(V \times \log_2 V)$ to build such a tree from scratch. But, this is not needed for each run.

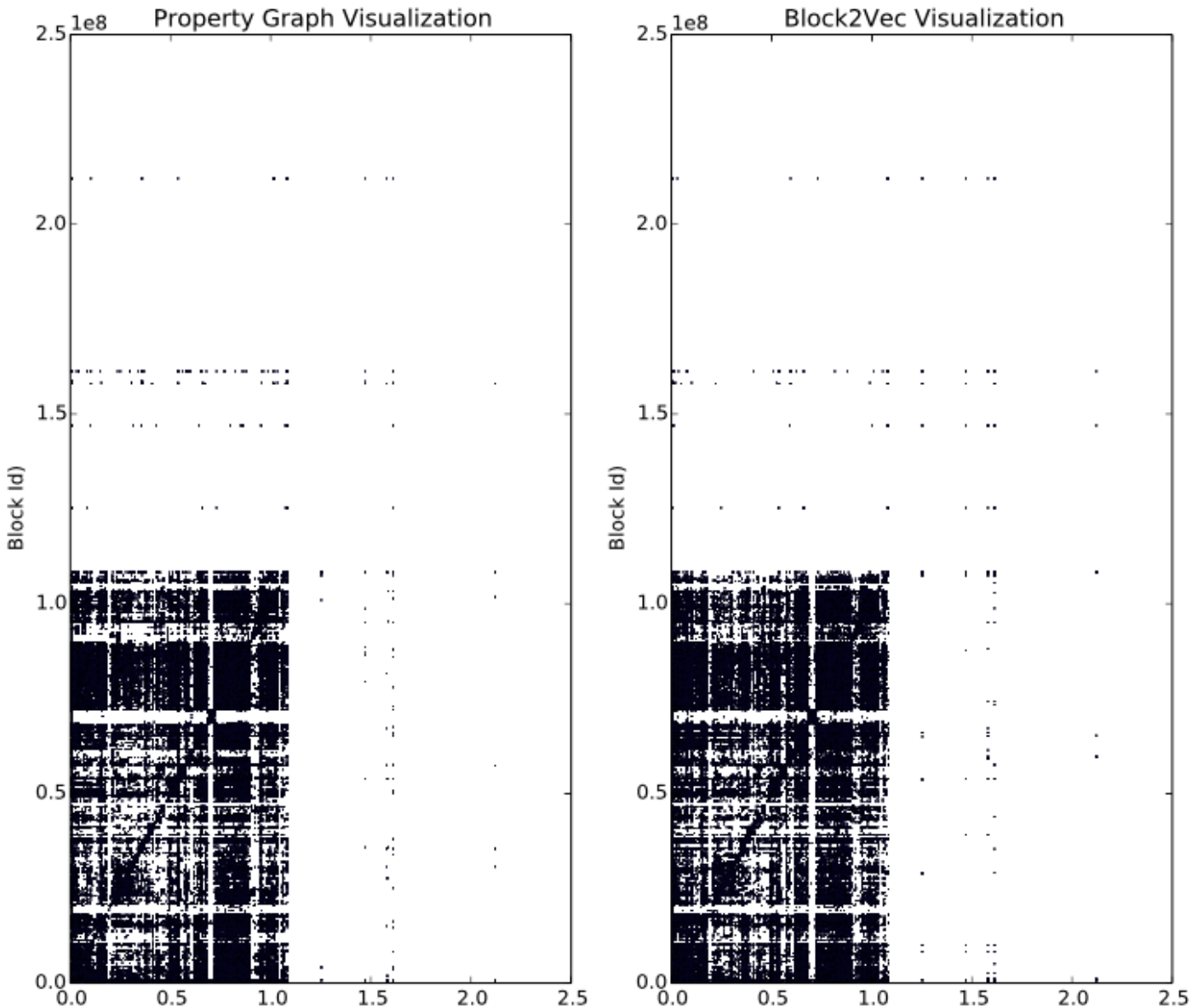
[2] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

Block2Vec Evaluation

Data Set

- Trace-driven simulations with several large disk traces collected in real systems.
- MSR Cambridge Traces (MSR-Cambridge) from SNIA as the test data set in this research.
  - The MSR-Cambridge trace contains 1-week block I/O traces in enterprise servers at MSR-Cambridge.
  - There are totally 36 I/O traces from 36 different volumes on 13 servers. We chose two of them from the servers `Project2` and `Proxy1` as the datasets for all evaluations.
  - `Project2` trace has approximately 29 million block I/O accesses, while `Proxy1` trace has more (around 133 million) block I/O accesses.

Block2Vec Evaluation: Visualization of Block Correlations



Block2Vec Evaluation

Block2Vec Training Performance

- We report the performance of training in different models, parameters and datasets.
  - The hardware platform: 1.7GHz Intel Core i7 CPU, 8GB memory and a 256GB SSD. No GPU or other accelerator.

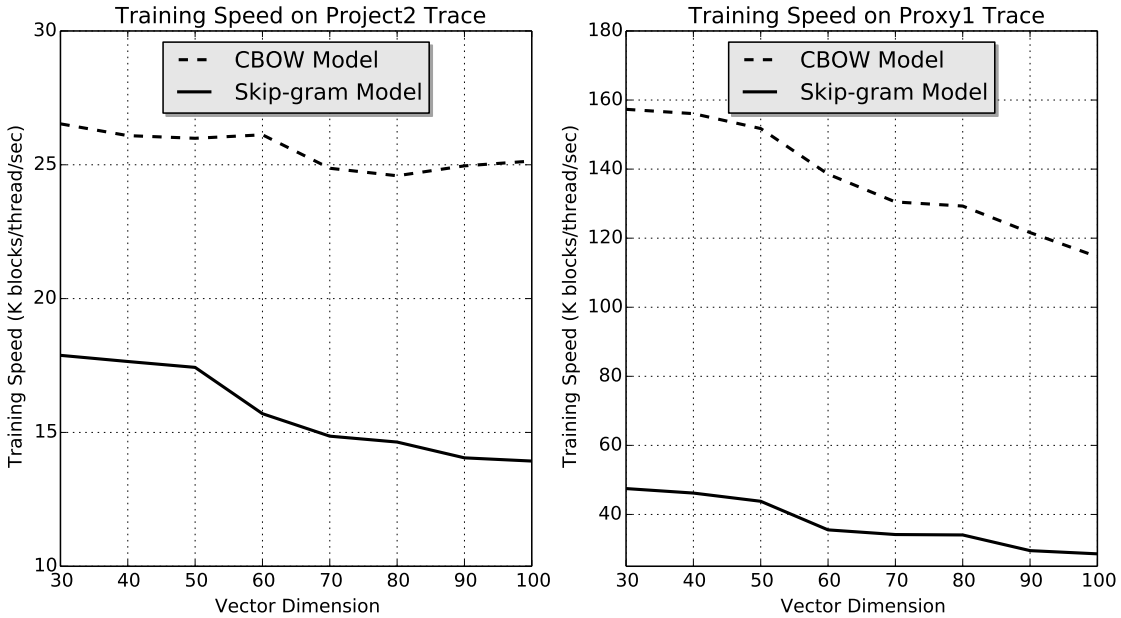


Figure : Training speed v.s. vector dimension on different training models and datasets.

Block2Vec Evaluation

Block2Vec Training Time

- The training time on different datasets and training models are provided in Table 1.
  - The time here is recorded as a single iteration through the training set. We normally run multiple iterations to achieve the best accuracy.
  - Both traces are collected from week-long run of production server, whereas the maximal training time is less than 1 hours.
  - it is feasible to run *Block2Vec* regularly in real system.

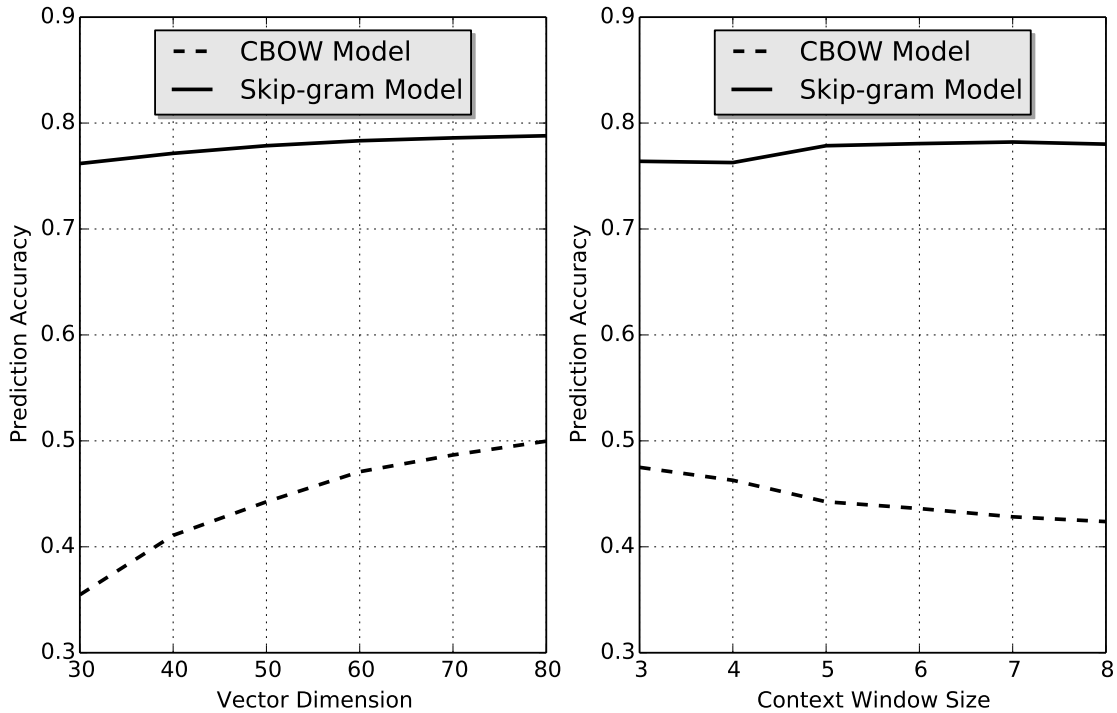
Table : Training Time with vector size 50 and context window 5

	Project2	Proxy1
CBOW Model	120.12 s	881.95 s
Skip-gram Model	183.91 s	2862.5 s

Block2Vec Evaluation

Block2Vec Prediction Accuracy

- The prediction criteria is largely simplified: each time, we predict next block access with 30 candidates and check their accuracy.
- We use `Project2` dataset, whose the first 90% of data trace was used to train the model and then the remaining 10% was for prediction.



Block2Vec Evaluation

Block2Vec Comparison Evaluation

- PG locates block in the graph and returns its top k neighbors.
- SP select next k consecutive blocks as the predictions.

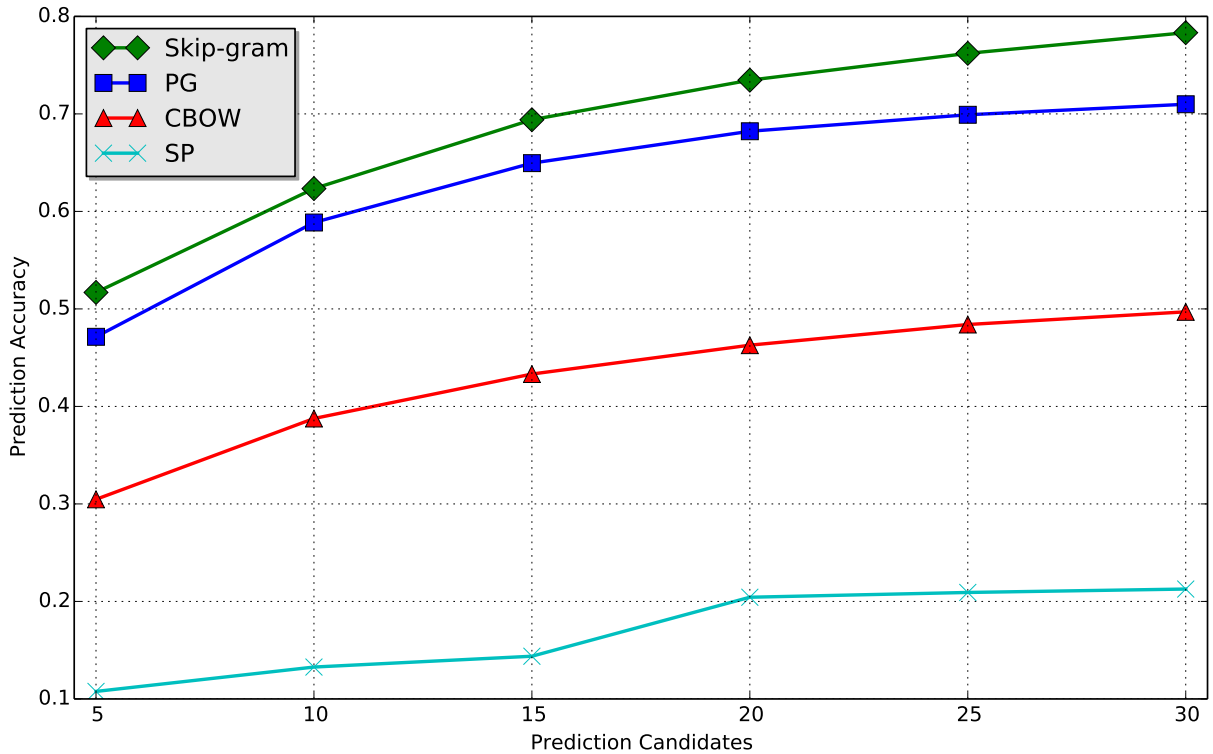


Figure : Training accuracy of Block2Vec (CBOW and Skip-gram), PG (probability graph), and SP (sequential prediction).

Conclusion and Future Work

Conclusion

- Considering block access trace as spoken language is not a bad idea!
- Vector representation of blocks can be efficiently obtained through deep learning.
- Mapping blocks to appropriate blocks improves the accuracy, even in simple task like predicting next block given current block.

Future Work?

- Can we learn more about applications or FS just from I/O traces?
- We believe the answer is Yes, but more work needs to be done.

Question and Answer