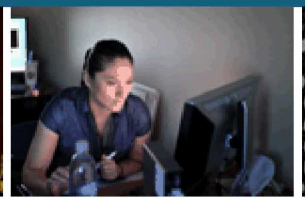
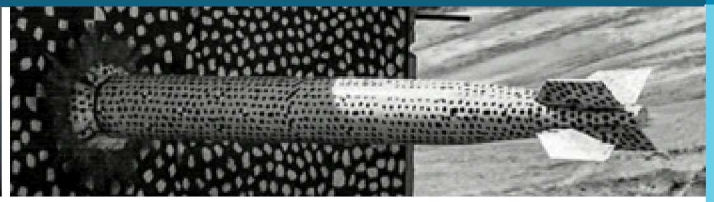
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2019-14021C

Parallel Data-Local Training for Optimizing Word2Vec Embeddings for Word and Graph Embeddings





PRESENTED BY

Gordon E. Moon[1], Denis Newman-Griffis[2], Jinsung Kim[3], Aravind Sukumaran-Rajam[4], Eric Fosler-Lussier[2], P. Sadayappan[3]

[1]Sandia National Laboratories*, [2]The Ohio State University, [3]University of Utah, [4]Washington State University

*This work was done during the Ph.D. of the first author at The Ohio State University

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

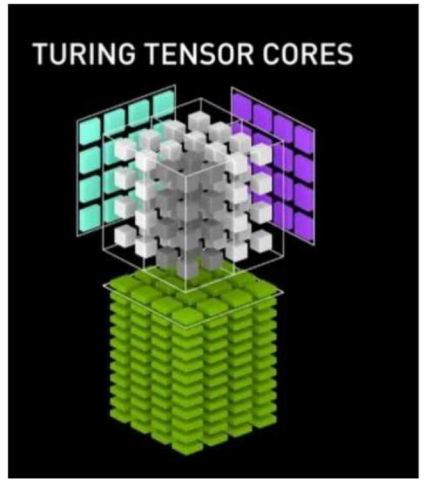
Machine Learning Everywhere

Machine Learning is becoming an integral part of everyday life

Top-Trending Machine Learning Architecture

Problem:

- How to achieve good performance?



Machine Learning Frameworks

Machine Learning frameworks like TensorFlow (Google), PyTorch (Facebook) and CNTK (Microsoft) are facilitating high productivity

Problem with Machine Learning framework:

- There is a significant gap in the performance achievable by machine learning frameworks and the peak compute capability of the current architectures

TensorFlow vs. Our approach

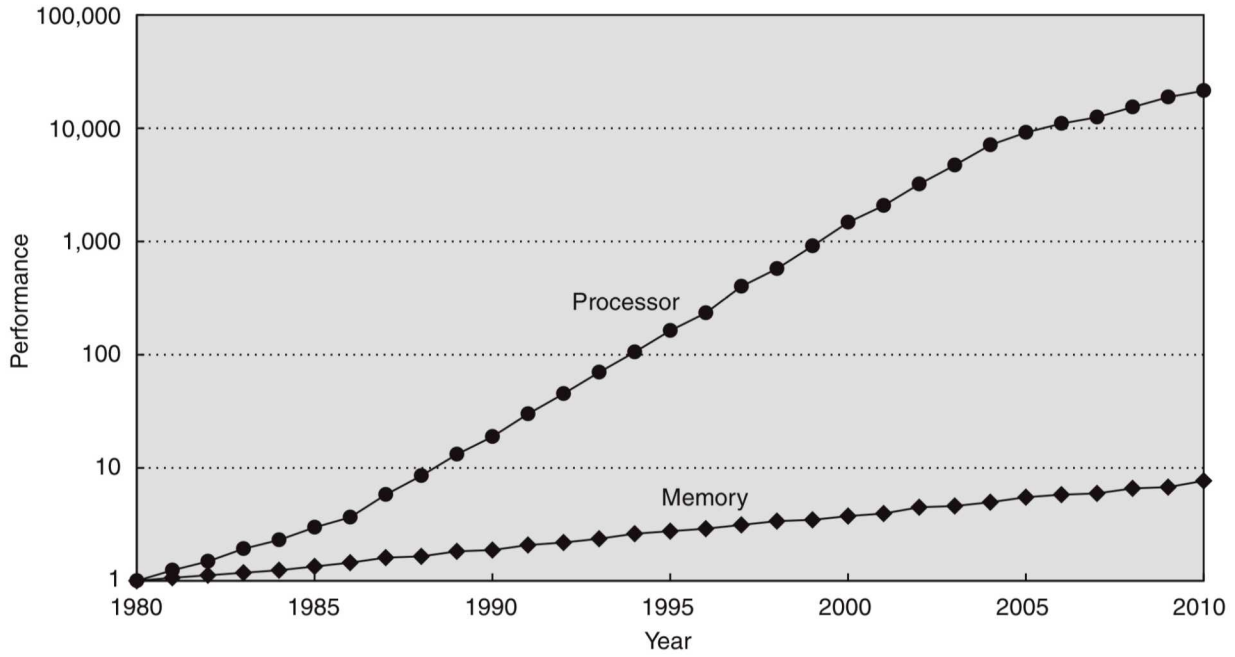
- Running on the same machine with text8 dataset

Word2Vec	TensorFlow	Our approach
Training time per epoch	59.02 (s)	1.02 (s)

FLOPs vs. Data Movement

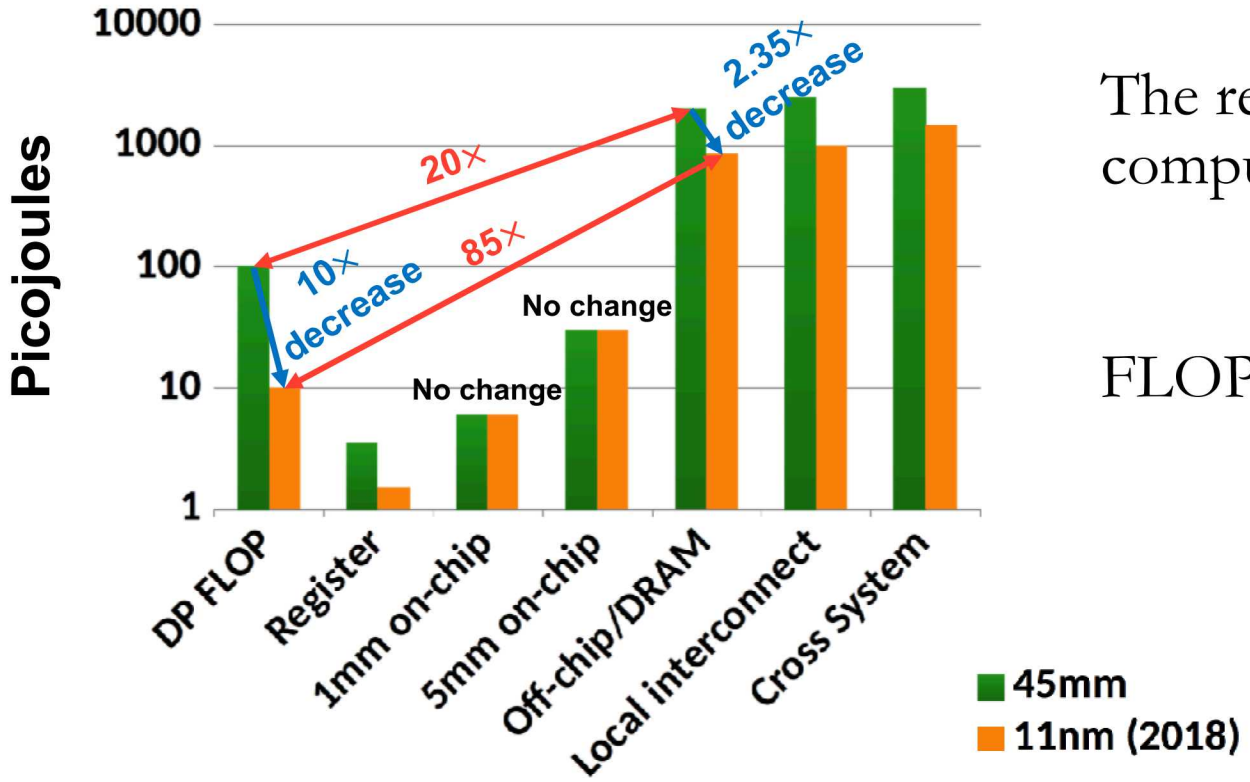
Aspects of Performance

- Processor (number of operations)
- Memory (data movement)



Source: John L. Hennessy (Stanford) and David A. Patterson (UC Berkeley)

Data Movement Cost: Energy Trends



The relative energy cost of data movement vs. computation keeps increasing (20× → 85×)

FLOPs are free; data movement is not

Source: Jim Demmel (UC Berkeley) and John Shalf (LBL)

Solution:

- Minimization of data movement overheads
- Architecture-aware Machine Learning algorithm design

Word Vector Representations in NLP

Four kinds of vector semantic models

- Hard clustering (e.g., Brown clustering)
- Soft clustering (e.g., SVD, NMF, LSA, LDA)
- Neural-network-inspired models
  - (e.g., Skip-Gram and CBOW in Word2Vec, ELMo, BERT)
- Mutual-information weighted word co-occurrence metrics

Source: Dan Jurafsky (Stanford). "Vector Semantics"

Outline

1. Word2Vec Word Embedding
2. Skip-gram based Word2Vec with Negative Sampling
3. Performance Challenges in Parallelizing Word2Vec
4. Parallel Attraction-Repulsion based Word2Vec
5. Data Movement Comparison
6. Performance Evaluation

Overview: Neural Word Embeddings

The intuition:

- Similar words appear in similar contexts*
- More precisely: Similar words have similar distributions of words to their left and right

Neural Word Embeddings

- The idea: Directly generate a dense vector to store "most" of the important information in a low-dimensional embedding space for each word in a fixed vocabulary

word vector for a word "apple"

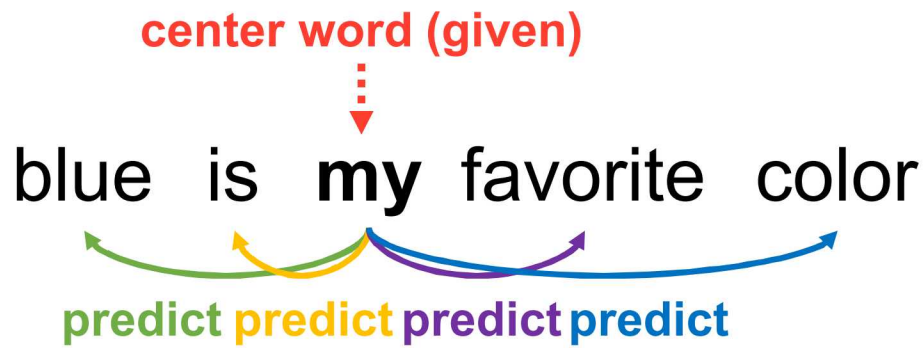
Fruit	0.47
Sport	0.05
Company	0.25
AI	0.12
Health	0.09
⋮	⋮

*Source: Zellig S. Harris. "Distributional Structure"

Overview: Word2Vec Word Embedding

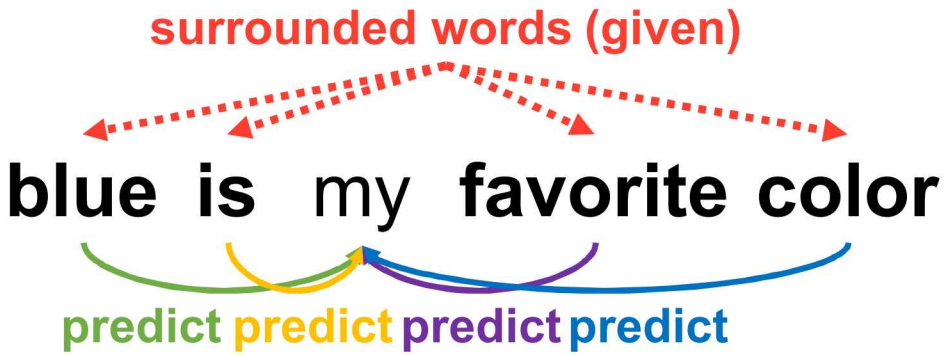
Word2Vec

- Goal: Find word representations that are useful for predicting its surrounding context words
- Context types
  - Skip-Gram (SG)



SG models pairwise probabilities individually

  - Continuous Bag-of-Words (CBOW)

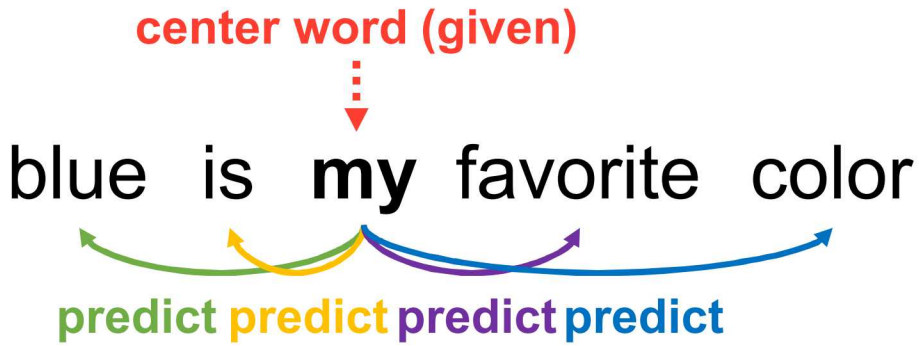


CBOW uses the average of surrounded words

Overview: Word2Vec Word Embedding

Skip-Gram based Word2Vec

- Predicts surrounding "outside" words given the "center" word



- Cost/Objective function:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{-C\le j\le C, j\neq 0} \log p(\text{word}_{n+j}|\text{word}_n)$$

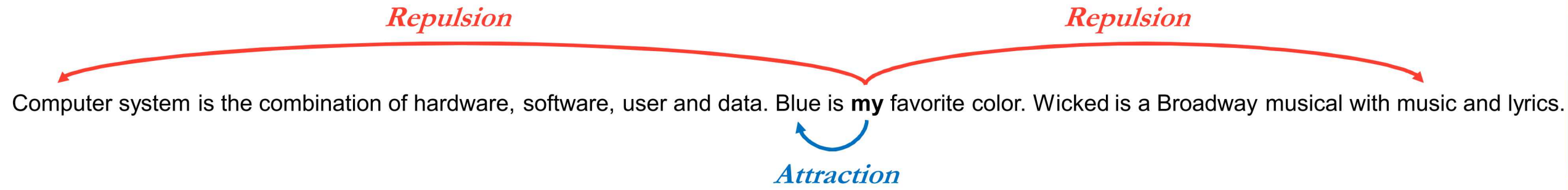
- For a "center" word_n and an "outside" word_{n+j}:

$$\log p(\text{word}_{n+j}|\text{word}_n) = \log \frac{\exp(<\vec{\mathbf{w}}_{n+j}^{out}, \vec{\mathbf{w}}_n^{in}>)}{\sum_{v=1}^{V}\exp(<\vec{\mathbf{w}}_v^{out}, \vec{\mathbf{w}}_n^{in}>)}$$

Overview: Word2Vec Word Embedding

Word2Vec

- Objective type
  - Negative Sampling (NS)



$$\log p(\text{word}_{n+j}|\text{word}_n) = \log \frac{\exp(<\vec{\mathbf{w}}_{n+j}^{out}, \vec{\mathbf{w}}_n^{in}>)}{\sum_{v=1}^{V} \exp(<\vec{\mathbf{w}}_v^{out}, \vec{\mathbf{w}}_n^{in}>)}$$

intractable as the vocabulary size $V > 500,000$ increases

replaced by

tractable! usually $5 \leq T \leq 20$

$$\log \sigma(<\vec{\mathbf{w}}_{n+j}^{out}, \vec{\mathbf{w}}_n^{in}>) + \sum_{t=1}^{T} \log \sigma(-<\vec{\mathbf{w}}_t^{out}, \vec{\mathbf{w}}_n^{in}>)$$

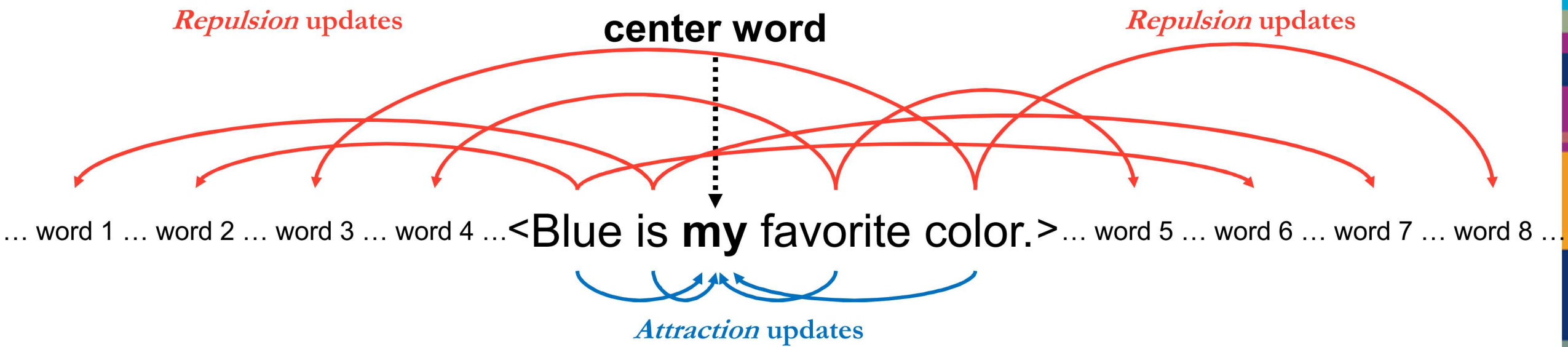
Overview: SG-NS based Word2Vec

Blue is my favorite color.

Overview: SG-NS based Word2Vec

***Attraction* update**: the computations that seek to align a word closer to its neighbors in the windows

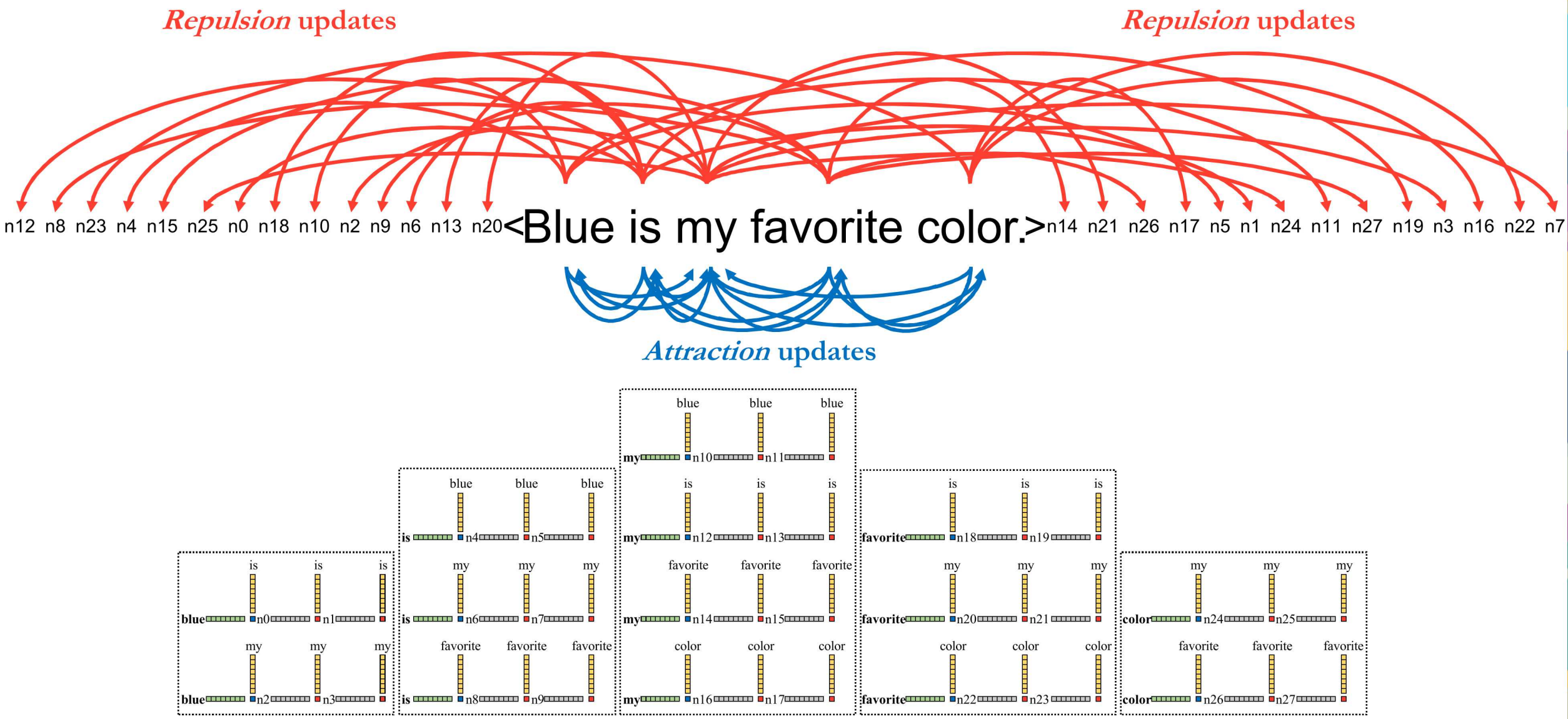
***Repulsion* update**: the negative-sampling computations



where window size = 2 and the number of negative samples = 2

Input word	blue	blue	blue	is	is	is	favorite	favorite	favorite	color	color	color
Target word	**my**	word 2	word 6	**my**	word 1	word 7	**my**	word 4	word 5	**my**	word 3	word 8
Update type	**A**	**R**	**R**	**A**	**R**	**R**	**A**	**R**	**R**	**A**	**R**	**R**

Overview: SG-NS based Word2Vec



Performance Challenges in Parallelizing Word2Vec

The total data movement cost:

$$\mathbf{SL(1 + W(1 + 4K + 8K(T + 1)))}$$

where
S: the total number of sentences over the corpus
L: the number of word tokens in each sentence
W: (2×window_size) − (2×window_startIdx)
T: the number of negative samples

```
repeat
  for sentence = 0 to S – 1 do
    L = number of word tokens in sentence
    for word = 0 to L – 1 do
      center_word = corpus[sentence][word]
      window_startIdx = random_uniform() % window_size
      for context = window_startIdx to 2 × window_size – window_startIdx do
        if context != window_size then
          input_word = corpus[sentence][word - window_size + context]
          Initialize temp[0 : K-1] to 0
          for t = 0 to num_negative_samples do
            if t == 0 then
              target_word = center_word, label = 1
            else
              target_word = random_uniform() % V, label = 0
            sum = 0
            for k = 0 to K – 1 do
              sum += W_in[input_word][k] × W_out[target_word][k]
            end
            gradient = (label – sigmoid(sum)) × learning_rate
            for k = 0 to K – 1 do
              temp[k] += gradient × W_out[target_word][k]
              W_out[target_word][k] += gradient × W_in[input_word][k]
            end
          end
          for k = 0 to K – 1 do
            W_in[input_word][k] += temp[k]
          end
        end
      end
    end
  end
until convergence
```

Data movement is the key bottleneck

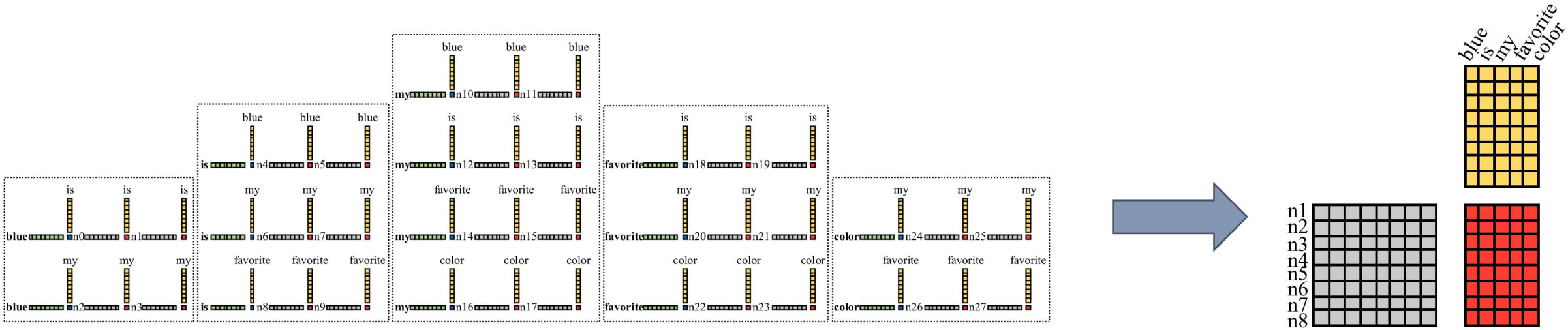
- A large number of dot-product computations between the embedding vectors is inherently memory-bandwidth limited

Minimum time required for data moved:

$$\mathbf{SL(1 + W(1 + 4K + 8K(T + 1)))} \times \textbf{bandwidth of the system}$$

How can we reduce data movement cost?

Performance Challenges in Parallelizing Word2Vec



Given two matrices A, (M×K) and B, (K×N),

- data movement of the ordinary matrix-matrix multiplication with iterative vector-vector multiplications:

$$\mathbf{2MNK + MN}$$

- data movement for efficient tiled matrix multiplication:

$$\frac{\mathbf{2MNK}}{\sqrt{\boldsymbol{\tau}}}$$

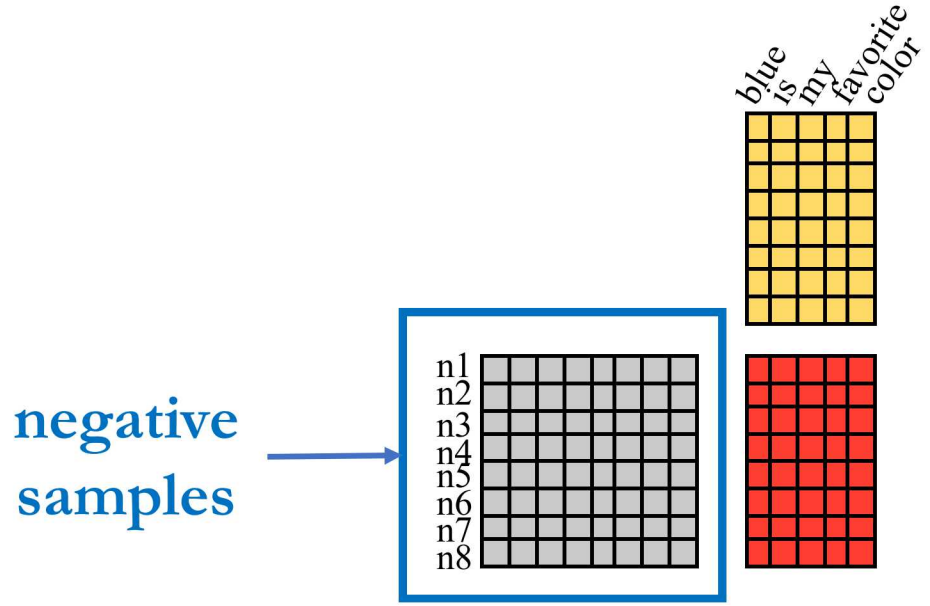
where τ: cache size

Source: Julien Langou. "Communication Lower Bounds for Matrix-Matrix Multiplication". (2015)

Performance Challenges in Parallelizing Word2Vec

The efficiency of matrix-matrix multiplication depends on the size of matrix

- The size of matrix depends on the number of negative samples



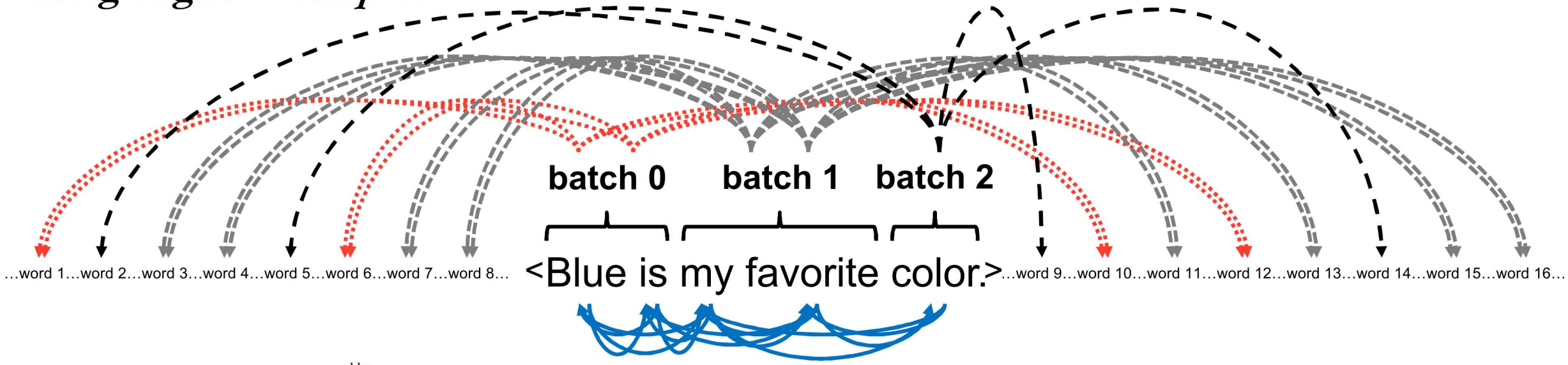
How can we reformulate multiple vector-vector multiplications to matrix-matrix multiplication?

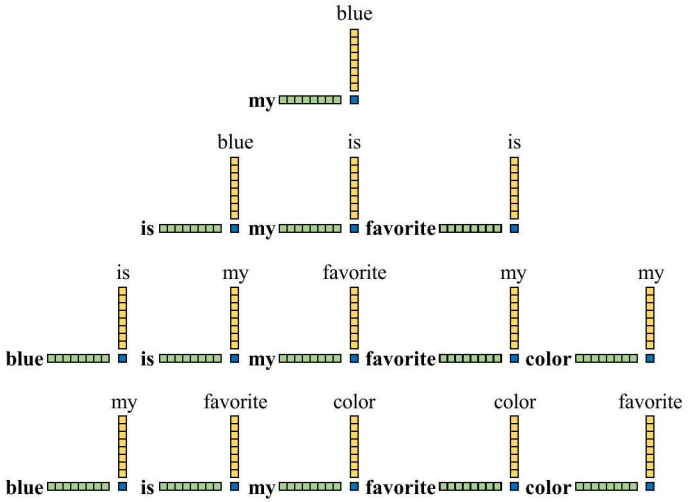
Parallel Attraction-Repulsion based Word2Vec

Blue is my favorite color.

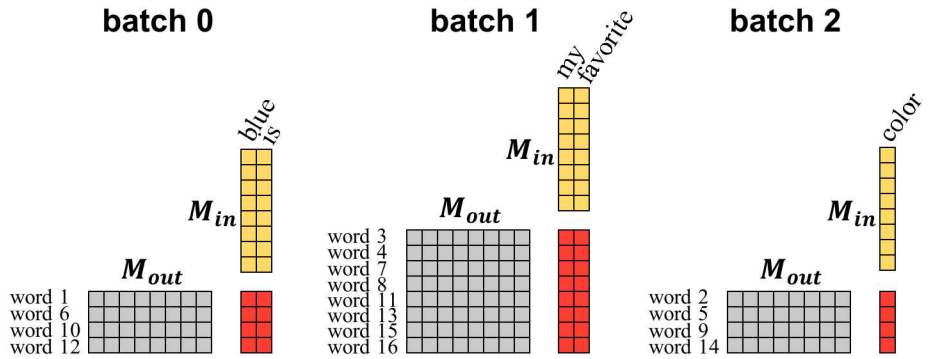
Parallel Attraction-Repulsion based Word2Vec

Sharing negative samples





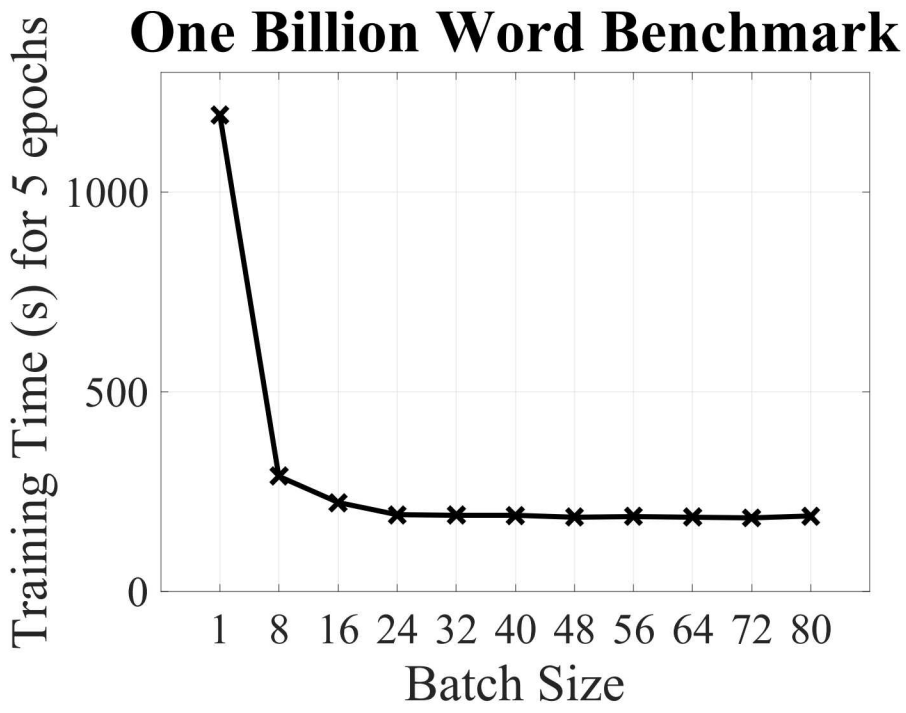
Attraction updates



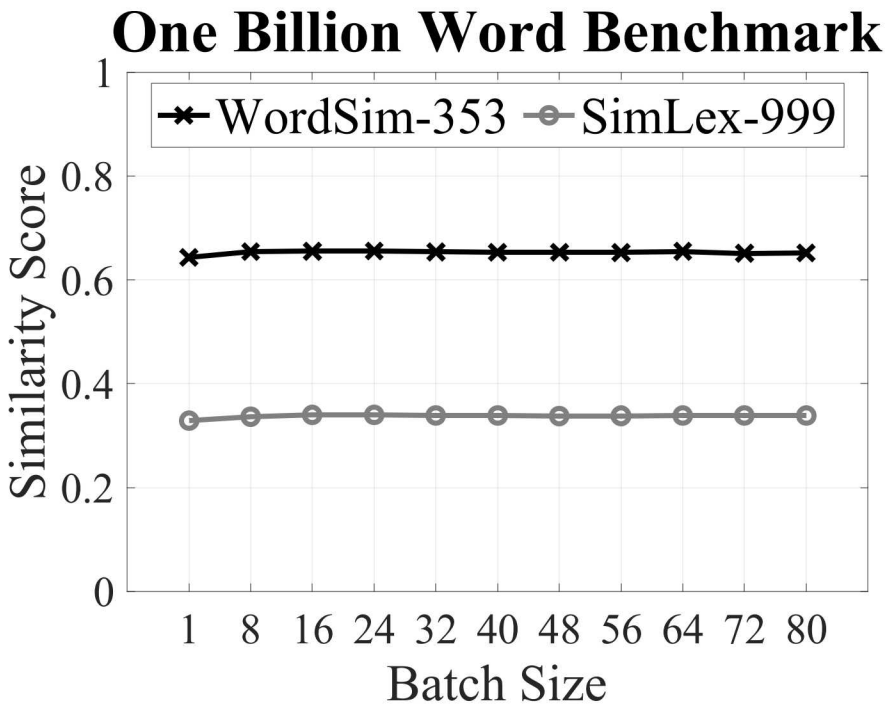
Repulsion updates

Parallel Attraction-Repulsion based Word2Vec

The impact of time and convergence across different mini-batch sizes



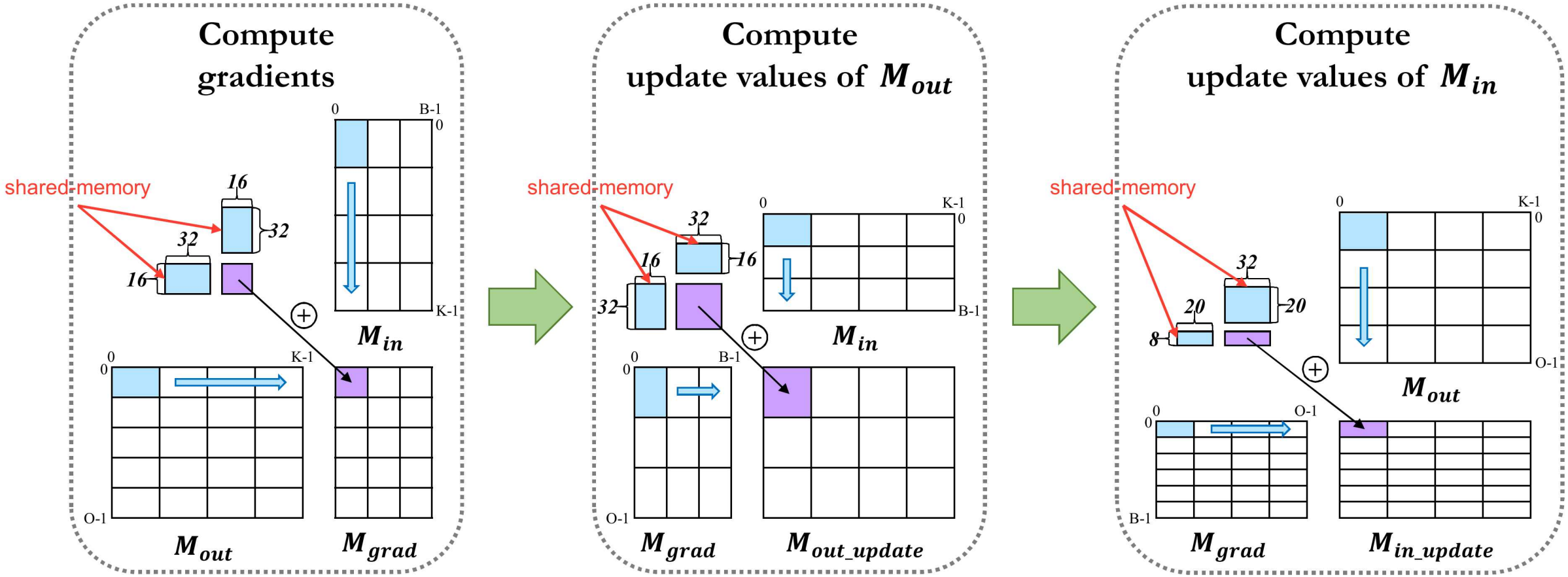
Mini-batch size vs. Training time



Mini-batch size vs. Convergence

Parallel Attraction-Repulsion based Word2Vec

Three matrix-matrix multiplications required for ***Repulsion*** phase



2D-tiled matrix-matrix multiplications using shared-memory

Parallel Attraction-Repulsion based Word2Vec

Data movement analysis

- The total data movement cost:

$$\mathbf{S}\left(\mathbf{L}(\textbf{Attraction phase}) + \frac{\mathbf{L}}{\mathbf{B}}(\textbf{Repulsion phase})\right) =$$

$$\mathbf{S}\left(\mathbf{L}(\mathbf{1}+\mathbf{W}(\mathbf{1}+\mathbf{8K})) + \frac{\mathbf{L}}{\mathbf{B}}\left(\frac{\mathbf{6HBK}}{\sqrt{\boldsymbol{\tau}}} + \mathbf{H} + \mathbf{4BK} + \mathbf{4KH} + \mathbf{2HB}\right)\right)$$

where
S: the total number of sentences over the corpus
L: the number of word tokens in each sentence
W: (2×window_size) − (2×window_startIdx)
K: embedding vector size
B: mini-batch size
H: the number of shared negative samples
τ: cache size

```
#pragma omp parallel num_threads(number of threads)
Distribute S sentences in the corpus into multiple threads
tId = thread id
S_pt = S / number of threads
repeat
  for sentence = tId × S_pt to (tId + 1) × S_pt – 1 do
    L = number of word tokens in sentence
    label = 1                                              Attraction phase
    for word = 0 to L – 1 do
      center_word = corpus[sentence][word]
      window_startIdx = random_uniform() % window_size
      for context = window_startIdx to 2 × window_size – window_startIdx do
        if context != window_size then
          input_word = corpus[sentence][word - window_size + context]
          target_word = center_word
          sum = 0
          for k = 0 to K – 1 do
            sum += W_in[input_word][k] × W_out[target_word][k]
          end
          gradient = (label – sigmoid(sum)) × learning_rate
          for k = 0 to K – 1 do
            temp = gradient × W_out[target_word][k]
            W_out[target_word][k] += gradient × W_in[input_word][k]
            W_in[input_word][k] += temp
          end
        end
      end
    end
    label = 0                                              Repulsion phase
    B = batch_size, num_batch = L / B
    for batch = 0 to num_batch – 1 do
      min_pos = batch × B, max_pos = min_pos + B – 1
      if (min_pos<=window_size–1) || (L–1–min_pos <= window_size–1) then
        rep_window_size = random_uniform() % window_size
      else
        rep_window_size = random_uniform() % (2 × window_size)
      end
      O = num_negative_samples × rep_window_size
      shared_ns[0:O-1] = random_uniform() % V
      memcpy(M_in[0:B-1][0:K-1], W_in[corpus[sentence][min_pos:min_pos+B-1]][0:K-1])
      memcpy(M_out[0:O-1][0:K-1], W_out[shared_ns[0:O-1]][0:K-1])
      M_grad[0:O-1][0:B-1] = sgemm(M_out[0:O-1][0:K-1], M_in[0:B-1][0:K-1])
      M_grad[0:O-1][0:B-1] = (label–sigmoid(M_grad[0:O-1][0:B-1])) × learning_rate
      M_out_update[0:O-1][0:K-1] = sgemm(M_grad[0:O-1][0:B-1], M_in[0:B-1][0:K-1])
      M_in_update[0:B-1][0:K-1] = sgemm(M_grad^T[0:O-1][0:B-1], M_out[0:O-1][0:K-1])
      add(W_in[corpus[sentence][min_pos:min_pos+B-1]][0:K-1], M_in_update[0:B-1][0:K-1])
      add(W_out[shared_ns[0:O-1]][0:K-1], M_out_update[0:O-1][0:K-1])
    end
  end
until convergence
```

Data Movement Comparison

Data movement	Original Word2Vec	Our approach
Attraction **phase**	$S(L(1+W(1+8K)))$	$S(L(1+W(1+8K)))$
Repulsion **phase**	$S(L(W(1+4K+8KT)))$	$S\left(\frac{L}{B}(\frac{6HBK}{\sqrt{\tau}}+H+4BK+4KH+2HB)\right)$
Total data movement	$S(L(1+W(1+4K+8K(T+1))))$	$S\left(L(1+W(1+8K))+\frac{L}{B}(\frac{6HBK}{\sqrt{\tau}}+H+4BK+4KH+2HB)\right)$

For the real One Billion Word Benchmark dataset

S	N	L	K	W	T	H	B	τ
30,607,741	804,269,958	N/S	128	16	5	80	24	35MB

where
S: the total number of sentences over the corpus
N: the total number of word tokens over the corpus
L: the number of word tokens in each sentence
W: (2×window_size) − (2×window_startIdx)
K: embedding vector size
B: mini-batch size
H: the number of shared negative samples
τ: cache size

Data movement	Original Word2Vec (byte)	Our approach (byte)
Attraction **phase**	13190832×10^6	13190832×10^6
Repulsion **phase**	72487241×10^6	19239278×10^5
Total data movement	85665204×10^6	15109321×10^6

97% reduced

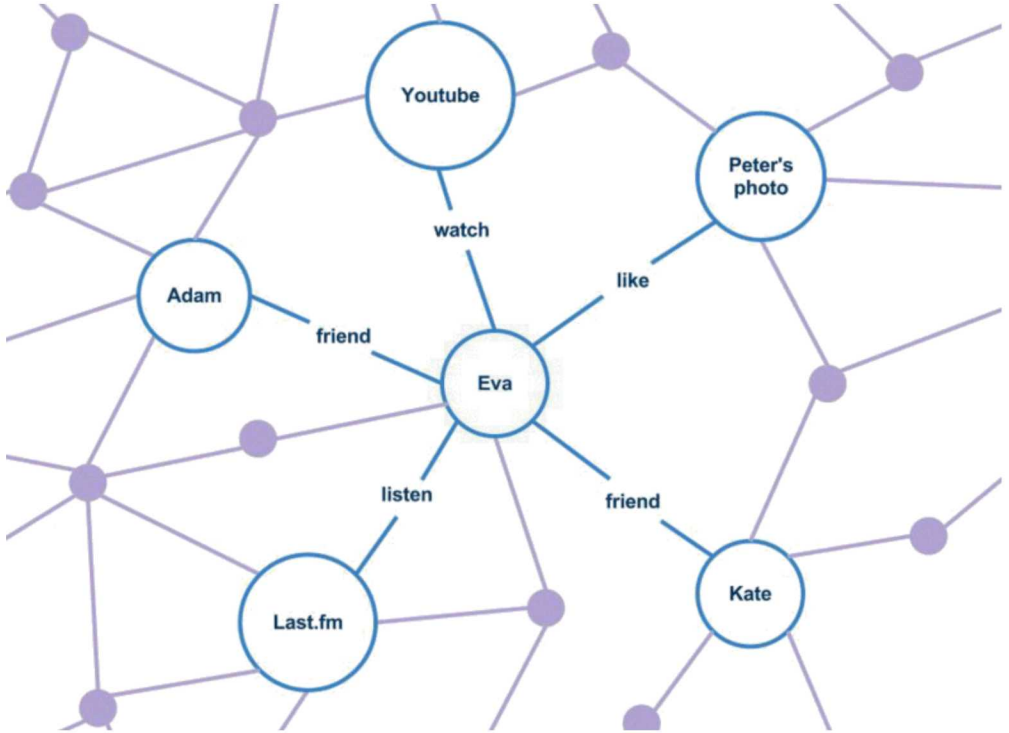
5.67× reduced

Performance Evaluation: Graph Embedding

Node2Vec

Input: Graph G: (V nodes, E edges and W: weights), p: return, q: in-out, R: number of random walks for each node, V: number of unique nodes, L: length of each walk, K: number of hidden units, C: window size
Output: W_{in}: $V \times K$ input node embedding matrix, W_{out}: $K \times V$ output node embedding matrix

$\pi \leftarrow$ Preprocess(G, p, q)
Generate new G': (V, E, π)
Initialize $random\_walks$
for r = 0 **to** $R - 1$ **do**
  for v = 0 **to** $V - 1$ **do**
    Initialize $walk$
    for walk_iter = 0 **to** $L - 1$ **do**
      sample_node $\leftarrow$ Sampling(G', π)
      $walk \leftarrow [walk;$ sample_node$]$
    end for
    $random\_walks \leftarrow [random\_walks;\ walk]$
  end for
end for
// $random\_walks$: pre-generated dataset to use it as an input for Word2Vec
repeat
  $W_{in}, W_{out} \leftarrow$ **SG-NS based Word2Vec**($random\_walks$, K, C)
until convergence



$$\frac{\text{training time for Word2Vec within Node2Vec}}{\text{total training time for Node2Vec}} = \frac{72.635360(s)}{81.505412(s)} = 89.12\% \text{ (BlogCatalog)}$$

Performance Evaluation: Setup

Datasets

	Dataset	V # of unique words OR # of unique nodes	S # of sentences over the corpus OR # of random walks over the graph	N # of word tokens over the corpus OR the sum of the length of the walks in S
Text datasets	text8	71,291	9,385	16,718,843
	1B-Word	555,514	30,607,741	804,269,958
Labeled graph datasets	BlogCatalog	10,313	103,120	8,352,720
	PPI	3,891	38,900	3,150,900
	Wikipedia-2006	4,778	47,770	3,869,370
Unlabeled graph datasets	Facebook	4,040	40,390	3,271,590
	ASTRO-PH	18,773	187,720	15,205,320

Machine configuration

Machine	Details
CPU	Intel(R) Xeon(R) CPU E5-2680 v4 (14 cores and 28 threads), 128 GB RAM, 76.8 GB/s bandwidth, 35 MB L2 cache; ICC 18.0.3
GPU	Tesla P100 PCIE, 56 SMs, 64 cores/MP, 16 GB Global Memory, 732 GB/s bandwidth, 4 MB L2 cache; CUDA 9.2.88

Performance Evaluation: Evaluation Metrics

For the word embeddings learned from Word2Vec

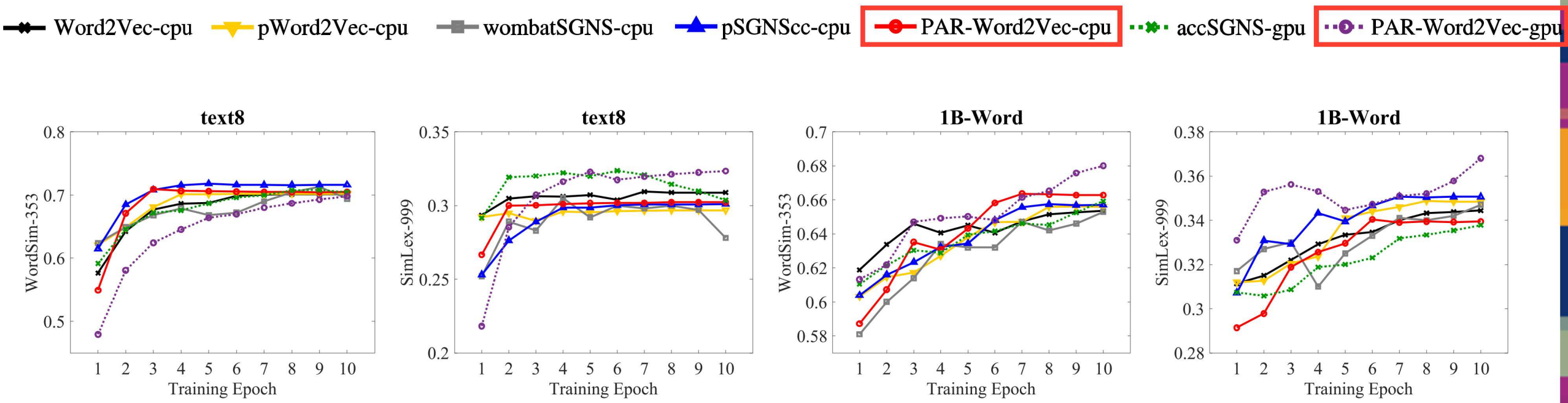
- Intrinsic evaluations
  - WordSim-353: 353 pairs rated for similarity of meaning
  - SimLex-999: 999 pairs rated specifically for similarity
- Extrinsic evaluations
  - Relation extraction: SemEval-2010 shared task, using a CNN
  - Sentiment analysis: positive/negative binary classification of IMDB, using a LSTM

For the node embeddings learned from Word2Vec

- Extrinsic evaluations
  - Multi-label classification, using a logistic regression
  - Link prediction, using a SVM

Performance Comparison: Quality

Intrinsic evaluations for word embeddings



Number of iterations vs. Word similarity scores

higher the better

Performance Comparison: Quality

Intrinsic evaluations for word embeddings

Model	text8		1B-Word	
	WordSim-353	SimLex-999	WordSim-353	SimLex-999
Word2Vec-cpu	0.701 (±0.010)	0.308 (±0.014)	0.653 (±0.004)	0.344 (±0.007)
pWord2Vec-cpu	0.701 (±0.008)	0.296 (±0.008)	0.656 (±0.003)	0.348 (±0.002)
wombatSGNS-cpu	0.694 (±0.013)	0.278 (±0.009)	0.653 (±0.001)	0.350 (±0.002)
pSGNScc-cpu	0.716 (±0.008)	0.301 (±0.009)	0.657 (±0.003)	0.350 (±0.001)
PAR-Word2Vec-cpu	0.705 (±0.008)	0.302 (±0.003)	0.663 (±0.002)	0.341 (±0.003)
accSGNS-gpu	0.704 (±0.004)	0.303 (±0.002)	0.659 (±0.003)	0.337 (±0.002)
PAR-Word2Vec-gpu	0.698 (±0.008)	0.323 (±0.005)	0.680 (±0.004)	0.368 (±0.004)

higher the better

Performance Evaluation: Evaluation Metrics

For the word embeddings learned from Word2Vec

- Intrinsic evaluations
  - WordSim-353: 353 pairs rated for similarity of meaning
  - SimLex-999: 999 pairs rated specifically for similarity
- Extrinsic evaluations
  - Relation extraction: SemEval-2010 shared task, using a CNN
  - Sentiment analysis: positive/negative binary classification of IMDB, using a LSTM

For the node embeddings learned from Word2Vec

- Extrinsic evaluations
  - Multi-label classification, using a logistic regression
  - Link prediction, using a SVM

Performance Comparison: Quality

Extrinsic evaluations for word embeddings

Model	text8		1B-Word	
	Relation Extraction	Sentiment Analysis	Relation Extraction	Sentiment Analysis
Word2Vec-cpu	0.671 (±0.010)	0.795 (±0.006)	0.689 (±0.009)	0.782 (±0.008)
pWord2Vec-cpu	0.669 (±0.006)	0.791 (±0.004)	0.686 (±0.008)	0.779 (±0.007)
wombatSGNS-cpu	0.666 (±0.007)	0.776 (±0.005)	0.691 (±0.010)	0.783 (±0.005)
pSGNScc-cpu	0.666 (±0.009)	0.790 (±0.005)	0.685 (±0.010)	0.784 (±0.006)
PAR-Word2Vec-cpu	0.665 (±0.010)	0.783 (±0.008)	0.691 (±0.008)	0.780 (±0.004)
accSGNS-gpu	0.680 (±0.010)	0.796 (±0.007)	0.689 (±0.006)	0.787 (±0.006)
PAR-Word2Vec-gpu	0.663 (±0.010)	0.807 (±0.004)	0.623 (±0.009)	0.780 (±0.004)

higher the better

Performance Evaluation: Evaluation Metrics

For the word embeddings learned from Word2Vec

- Intrinsic evaluations
  - WordSim-353: 353 pairs rated for similarity of meaning
  - SimLex-999: 999 pairs rated specifically for similarity
- Extrinsic evaluations
  - Relation extraction: SemEval-2010 shared task, using a CNN
  - Sentiment analysis: positive/negative binary classification of IMDB, using a LSTM

For the node embeddings learned from Word2Vec

- Extrinsic evaluations
  - Multi-label classification, using a logistic regression
  - Link prediction, using a SVM

Performance Comparison: Quality

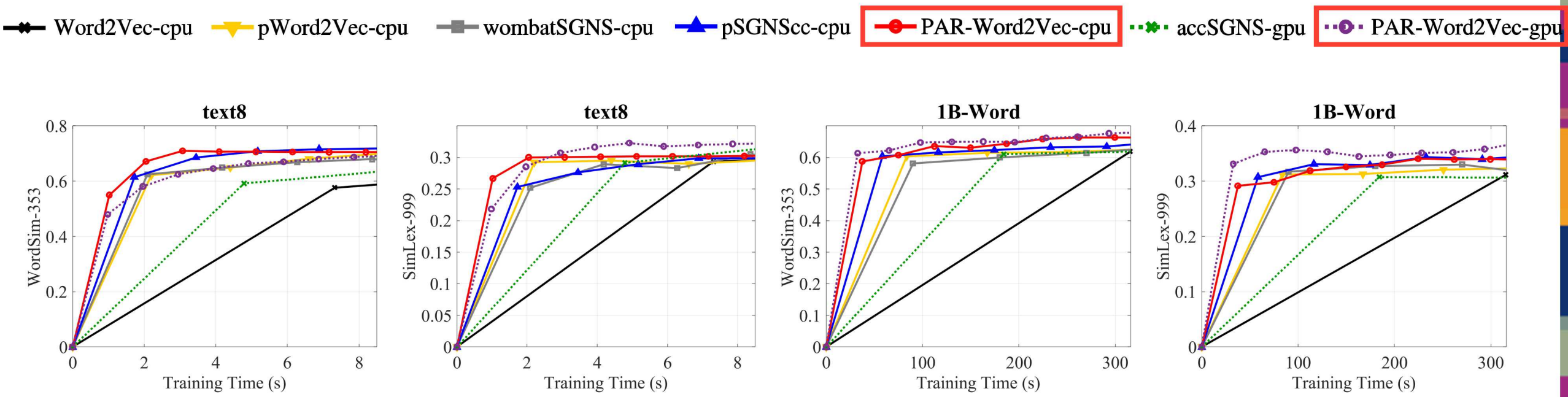
Extrinsic evaluations for node embeddings

Model	BlogCatalog		PPI		Wikipedia-2006		Facebook	ASTRO-PH
	Multi-label Classification						Link Prediction	
	Micro F_1	Macro F_1	Micro F_1	Macro F_1	Micro F_1	Macro F_1	Micro F_1	Micro F_1
Word2Vec-cpu	0.429	0.306	0.213	0.188	0.461	0.081	0.699	0.723
pWord2Vec-cpu	0.422	0.304	0.211	0.187	0.478	0.088	0.691	0.721
wombatSGNS-cpu	0.429	0.310	0.211	0.184	0.464	0.079	0.692	0.718
pSGNScc-cpu	0.422	0.301	0.211	0.184	0.442	0.070	0.686	0.692
PAR-Word2Vec-cpu	0.425	0.304	0.223	0.194	0.480	0.101	0.687	0.723
accSGNS-gpu	0.423	0.297	0.215	0.190	0.460	0.082	0.698	0.721
PAR-Word2Vec-gpu	0.420	0.291	0.217	0.184	0.456	0.071	0.672	0.720
Average std dev.	±0.006	±0.011	±0.006	±0.006	±0.006	±0.008	±0.001	±0.002

higher the better

Performance Comparison: Speedup

PAR-Word2Vec-cpu achieved up to 9× speedup over Word2Vec-cpu



Training time vs. Word similarity scores

higher the better

Performance Comparison: Speedup

PAR-Word2Vec-cpu achieved up to 9× speedup over Word2Vec-cpu

Model	Text Dataset		Labeled Graph Dataset			Unlabeled Graph Dataset	
	text8	1B-Word	BlogCatalog	PPI	Wikipedia-2006	Facebook	ASTRO-PH
Word2Vec-cpu	7.32	315.46	6.43	3.13	2.95	2.55	12.54
pWord2Vec-cpu	2.20	86.63	1.56	0.45	0.55	0.53	2.66
wombatSGNS-cpu	2.09	90.04	1.43	0.47	0.58	0.71	2.88
pSGNScc-cpu	1.72	58.20	1.46	0.70	0.75	0.84	2.77
PAR-Word2Vec-cpu	1.02	37.43	0.83	0.33	0.28	0.31	1.43
accSGNS-gpu	4.79	185.31	2.23	0.66	0.62	1.37	6.44
PAR-Word2Vec-gpu	0.98	32.60	0.72	0.20	0.21	0.27	1.08

Comparison of the training time in seconds per epoch

lower the better

Conclusions

FLOPs are free, but data movement is expensive

- Architecture-aware machine learning algorithm design/implementation is critical

Parallelization of Word2Vec

- Devised an efficient Word2Vec algorithm for multi-core CPUs and GPUs by improving the performance of negative sampling method through increasing data reuse and decreasing data movements

Achieved significant speedup compared to the other existing state-of-the-art implementations

Thank you