Efficient Directional Curvature Optimization for Word Embeddings

Aarush Kulkarni, Sirui (Kris) Sheng, James Wu
Mentor: Dr. Willy Rodriguez

PRIMES Fall-Term Conference
Sunday, October 25, 2026

Introduction

NLP and Word Embeddings

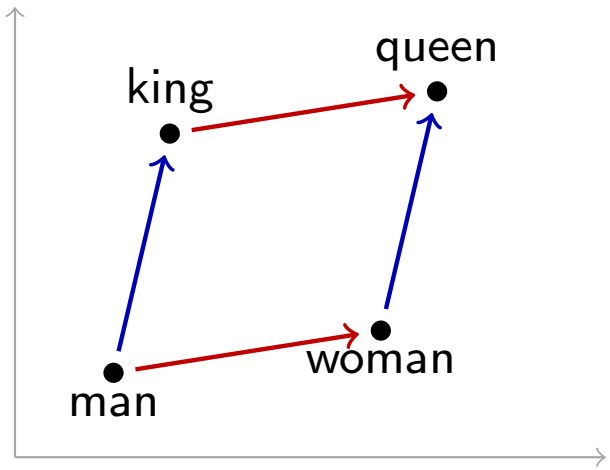
Natural language processing (NLP) is the study of how computers can work with human language.

Definition (Word embedding)

A word embedding assigns to every word in a vocabulary V a vector in $\mathbb{R}^d$, where the dimension d is much smaller than the number of words $|V|$.

- Typically d is between 50 and 600, while $|V|$ is between 10^4 and 10^6.
- Good embeddings encode meaning in their geometry. For example,

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}.$$



- Embeddings are still trained from scratch for medical, legal, and scientific text, so training speed matters.

The word2vec Models

Introduced by Mikolov et al. in 2013:

Definition (word2vec)

word2vec is a family of two models that learn embeddings with a shallow neural network.

The idea is that a word is described by the words around it, the C words on each side of a target word (the "context" words).

Example

In "the young fellow named John loved math" with $C = 2$, the target word is *John* and its context words are *fellow*, *named*, *loved*, and *math*.

- Continuous bag-of-words (**CBOW**) averages the context words and predicts the target word.
- **Skip-gram** uses the target word to predict each context word.

CBOW as an Optimization Problem

Every word gets two vectors, one used when it appears as context and one used when it is the target.

We write v_w and u_w for the two vectors of a word w.

Definition (CBOW model)

Let h be the average of the $2C$ context vectors. The CBOW model gives each word w the probability $p_w \propto e^{\langle u_w, h \rangle}$.

Definition (Loss)

The loss on one example is $-\log p_w$, where w is the correct word. Training minimizes the average loss $L(\theta)$ over all $2|V|d$ numbers θ.

First-Order and Second-Order Optimization

Definition (Direction & distance)

A direction is a vector Δ with $\|\Delta\| = \sqrt{\Delta_1^2 + \cdots + \Delta_n^2} = 1$. The distance is the number $t > 0$ we travel, giving the update $\theta \leftarrow \theta + t\,\Delta$.

Example

$\Delta = (0.6, 0.8)$ has $\|\Delta\| = 1$; with $t = 0.5$, θ moves by $(0.3, 0.4)$.

- **First-order methods** (SGD, Adam) use only the gradient, so the distance must be hand-tuned.
- **Second-order methods** also use the curvature, which tells us how far to go.

The Loss Along a Line

Fix θ and Δ, and write $\ell(t) = L(\theta + t\,\Delta)$ for the loss along the line.

Definition (Rate of decrease & curvature)

The rate of decrease along Δ is $g = -\ell'(0)$, and the curvature along Δ is $\kappa = \ell''(0)$.

We can then do the following second order taylor like expansion to get the optimal step size:

$$\ell(t) \;\approx\; \ell(0) - g\,t + \tfrac{1}{2}\kappa\,t^2 \qquad \Longrightarrow \qquad t \;=\; \frac{g}{\kappa}.$$

Note that this resembles Newton's method!

The RED Algorithm

RED (Rescaled-Exponential-Decay) uses the curvature to choose the distance. We need just one number, κ; the algorithm proceeds as follows.

1. Take Δ from a base optimizer (we use the normalized negative gradient).
2. Measure g and κ along that one direction.
3. Step $\theta \leftarrow \theta + (g/\kappa)\,\Delta$.

Fact (de Gournay–Gossard, 2022)

If the curvature along the line stays below 2κ, this step is guaranteed to decrease the loss.

Motivating Questions

Training word embeddings means minimizing a loss.

Second-order methods also use the curvature of the loss.

Hence, we were curious about the following questions:

Question #1

Can we compute the curvature of word2vec as efficiently as its gradient?

Question #2

How does the curvature behave during training?

Question #3

Does curvature-aware training give better word embeddings?

Theoretical Results

Scores and Their Rates

Definition (Score)

The score of a word w is $s_w = \langle u_w, h \rangle$.

Definition (Rate)

The rate of s_w along Δ is $\dot{s}_w = \frac{d}{dt} s_w(\theta + t\,\Delta)\big|_{t=0}$.

Example

The score of *John* is $s_{\text{John}} = \langle u_{\text{John}}, h \rangle$. If $\dot{s}_{\text{John}} > 0$, the model becomes more confident in *John*.

A Formula for the Curvature

Fact (Kulkarni–Sheng–Wu, 2026; cf. Schraudolph, 2002)

As we move along Δ, the score of every word changes at its own rate $\dot{s}_w$, and

$$\kappa = \underbrace{\mathrm{Var}_p(\dot{s})}_{\text{always } \geq 0} + \underbrace{2\left\langle \sum_w (p_w - 1[w = \text{target}])\, \dot{u}_w,\ \dot{h} \right\rangle}_{\text{the only part that can be negative}}$$

where $\dot{s}_w = \langle \dot{u}_w, h \rangle + \langle u_w, \dot{h} \rangle$, and the dot means the rate of change along Δ.

- Both terms are constructed from quantities the gradient already computes.
- A similar formula holds for skip-gram.

On the Computation of Curvature

In general, κ costs a constant number of gradient computations.

Theorem (Kulkarni–Sheng–Wu, 2026)

For CBOW and skip-gram, a one-pass algorithm gives the gradient and κ together in $O\big((|V| + C)d\big)$ time.

1. Average the $2C$ context vectors. $O(Cd)$
2. Score every word in the vocabulary. $O(|V|d)$
3. Construct the gradient and κ from the scores. $O(|V|d)$

Answer to Question #1

Yes! Same order as the gradient alone, and this is optimal.

Curvature Bounds

Let B be the largest norm of any word vector.

Theorem (Kulkarni–Sheng–Wu, 2026)

For CBOW with $2C$ distinct context words, every direction satisfies

$$-\frac{1}{\sqrt{C}} \;\le\; \kappa \;\le\; B^2\Big(\frac{1}{2}+\frac{1}{2C}\Big)+\frac{1}{\sqrt{C}}.$$

Adding a penalty $\frac{\lambda}{2}\|\theta\|^2$ to the loss shifts both bounds up by λ.

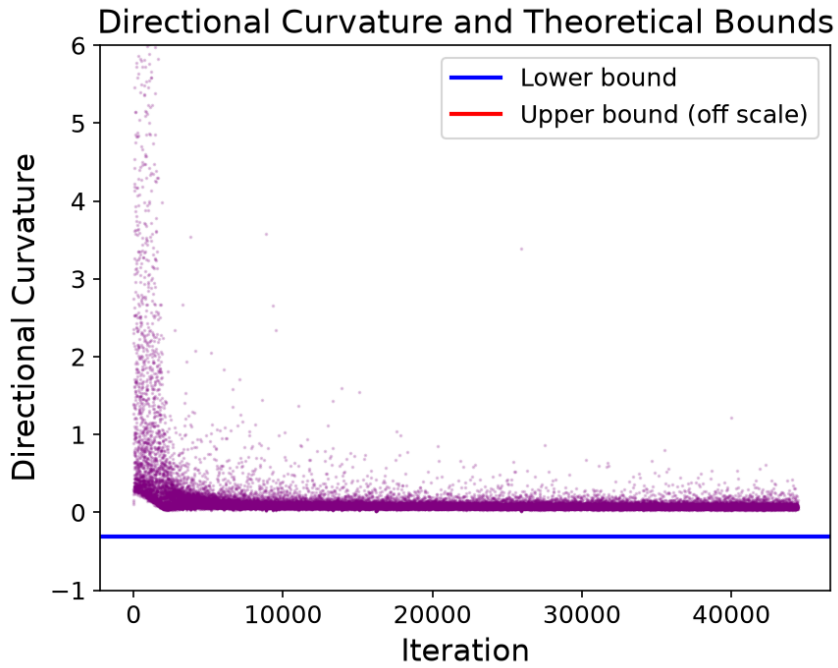
Example

With $C = 10$, we get $-0.32 \le \kappa \le 0.55\,B^2 + 0.32$.

Once $\lambda > 1/\sqrt{C}$, the curvature is positive and bounded.

Curvature Bounds in Practice

Curvature for each batch (the examples used in one step), with $C = 10$ and $\lambda = 10^{-7}$.





Answer to Question #2

The curvature stays bounded, inside the range the theorem predicts.

Experimental Results

Experimental Setup

- We trained the same word2vec model on the Simple English Wikipedia dataset (13 million words).
- Each time, we changed only the optimizer, with 3 runs per optimizer.
- We noted the loss throughout training.
  - An epoch is one pass through the data.
  - The test loss is the loss on text not used for training.
- We evaluated the embeddings on WordSim-353, SimLex-999 (similarity), and the Google Analogy Test Set.
- Our code can be found on GitHub[1] for reproducibility.

[1] https://github.com/ak4774971/PRIMES-2026-Code

The Curvature-Rescaled Step

Every optimizer computes a search vector p_k from the gradient and moves in the direction $\Delta_k = -p_k/\|p_k\|$.

- **Baselines** step a fixed amount, $\theta_{k+1} = \theta_k - \eta\, p_k$.
- **The RED family** travels the distance $t_k = c \cdot \dfrac{g_k}{2\tilde{\kappa}_k}$, where g_k is the rate of decrease along Δ_k, $\tilde{\kappa}_k$ is a running average of recent curvatures, and c is a constant.

Example

If recent curvatures were $0.9,\ 1.1,\ 1.0$, then $\tilde{\kappa}_k \approx 1.0$. With $c = 2$, this is the step $t = g/\kappa$ from before.

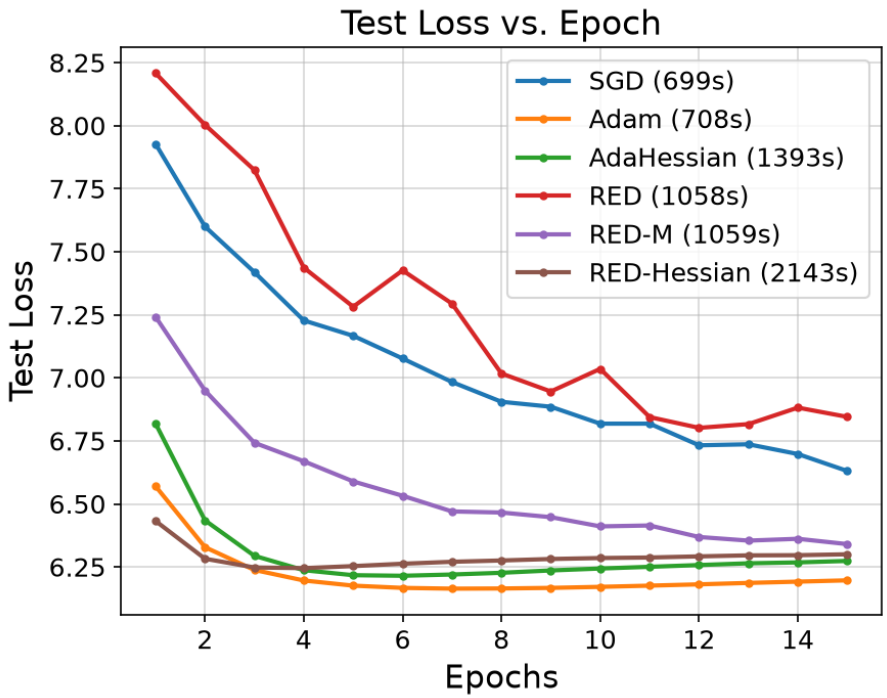
The Six Optimizers

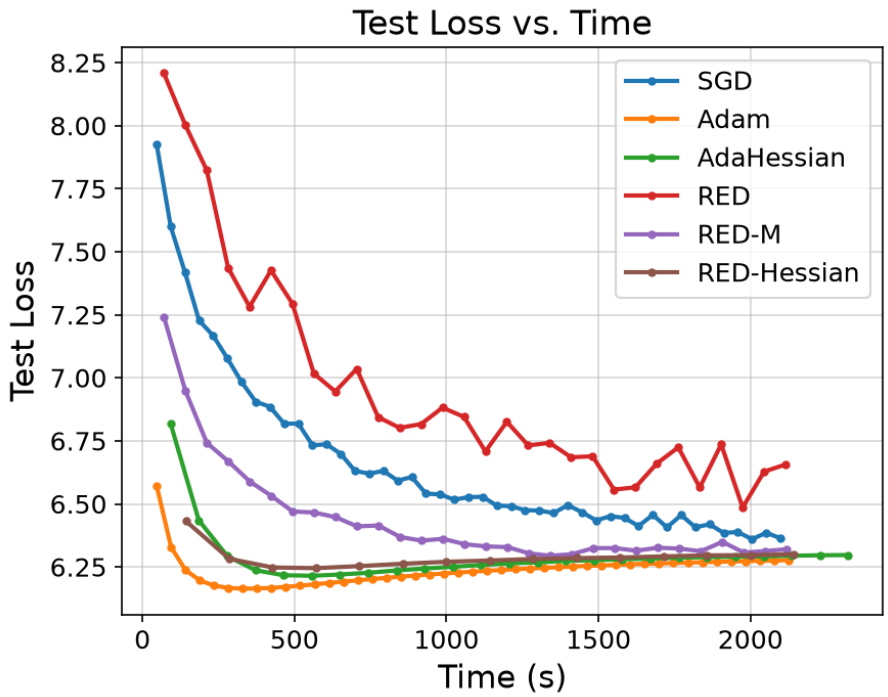
Optimizer	Search vector p_k	Step
SGD [Robbins & Monro, 1951]	gradient	fixed η
Adam [Kingma & Ba, 2015]	momentum, rescaled per coordinate	fixed η
AdaHessian [Yao et al., 2021]	momentum, rescaled by curvature	fixed η
RED [de Gournay & Gossard, 2022]	gradient	t_k
RED-M [Ours]	momentum	t_k
RED-Hessian [Ours]	same as AdaHessian	t_k

Table: Green entries change SGD; red entries are the same as in SGD.

Momentum is a running average of the past gradients.

Results (Loss)





RED reaches a higher test loss than SGD.

Results (Embedding Quality)

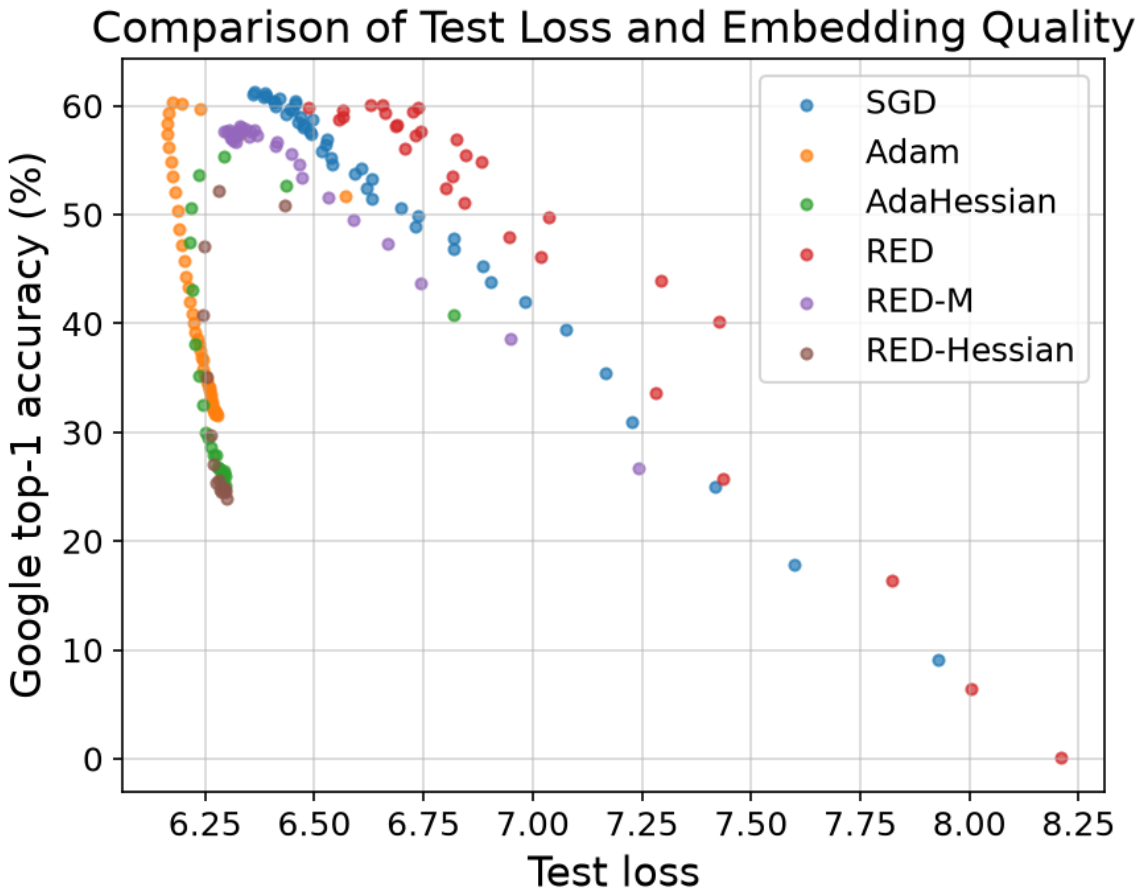
Optimizer	Similarity (ρ)		Google Analogy (%)		
	SimLex-999	WordSim-353	Top-1	Top-3	Top-5
SGD	0.275	0.633	61.08	77.76	82.14
Adam	0.306	0.649	58.44	76.28	81.08
AdaHessian	0.305	0.594	47.40	64.32	70.18
RED	0.267	0.641	59.86	77.10	81.82
RED-M	0.306	0.643	57.66	75.50	80.38
RED-Hessian	0.289	0.523	40.83	57.04	63.39

Table: Best and worst in each column.

Answer to Question #3

Not in general. Adam is best on similarity and SGD on analogy. Plain RED is worse than SGD in four of the five columns.

Lowest Loss ≠ Highest Quality



At the lowest test loss, embedding quality spans a large range.

Conclusion

Conclusion

- The curvature of word2vec costs the same as the gradient.
- The curvature is bounded, and positive under a penalty.
- RED reaches a higher test loss than SGD.
- Adam has the best similarity score, SGD is the best for analogy scores.
- Lowest test loss does not mean the best embeddings.

Acknowledgments

We would like to thank:

- Dr. Willy Rodriguez, our mentor, for providing the reading material and insight needed to start this project, and for his very helpful feedback and guidance throughout our work;
- Dr. Felix Gotti for organizing our practice sessions;
- Dr. Tanya Khovanova, Dr. Slava Gerovitch, Prof. Pavel Etingof, and the MIT PRIMES-USA Program for making this research possible;
- and our families.

References I



A. Botev, H. Ritter, and D. Barber.

Practical Gauss-Newton optimisation for deep learning.

International Conference on Machine Learning, 2017.



F. de Gournay and A. Gossard.

Adaptive scaling of the learning rate by second order automatic differentiation.

arXiv:2210.14520, 2022.



V. Gupta, T. Koren, and Y. Singer.

Shampoo: Preconditioned stochastic tensor optimization.

International Conference on Machine Learning, 2018.



D. P. Kingma and J. Ba.

Adam: A method for stochastic optimization.

International Conference on Learning Representations, 2015.

References II



H. Liu, Z. Li, D. Hall, P. Liang, and T. Ma.

Sophia: A scalable stochastic second-order optimizer for language model pre-training.

International Conference on Learning Representations, 2024.



D. C. Liu and J. Nocedal.

On the limited memory BFGS method for large scale optimization.

Mathematical Programming, 45:503–528, 1989.



T. Mikolov, K. Chen, G. Corrado, and J. Dean.

Efficient estimation of word representations in vector space.

arXiv:1301.3781, 2013.



B. A. Pearlmutter.

Fast exact multiplication by the Hessian.

Neural Computation, 6(1):147–160, 1994.

References III



H. Robbins and S. Monro.

A stochastic approximation method.

Annals of Mathematical Statistics, 22(3):400–407, 1951.



N. N. Schraudolph.

Fast curvature matrix-vector products for second-order gradient descent.

Neural Computation, 14(7):1723–1738, 2002.



N. Vyas, D. Morwani, R. Zhao, I. Shapira, D. Brandfonbrener, L. Janson, and S. Kakade.

SOAP: Improving and stabilizing Shampoo using Adam.

arXiv:2409.11321, 2024.



Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. W. Mahoney.

AdaHessian: An adaptive second order optimizer for machine learning.

AAAI Conference on Artificial Intelligence, 2021.

Thank you! Any questions?