

Directional Curvature-Rescaled Optimization for Word2Vec Training

Aarush Kulkarni¹ Kris Sheng² James Wu³

August 25, 2026

¹Acton-Boxborough Regional High School ²PRISMS ³Lexington High School
PRIMES-USA, MIT

Introduction

Curvature bounds for Word2Vec

Experiments

Conclusion

Introduction

What is Natural Language Processing?

Natural Language Processing (NLP) is the field concerned with getting computers to process, understand, and generate human language.

Some classical NLP tasks:

- Machine translation: “*puedo ir al baño?*” → “*can I go to the bathroom?*”.
- Sentiment analysis: does this review express a positive or a negative opinion?
- Language modelling: predict the next word (behind today's LLMs!)

Sadly language is not numbers

Every one of these tasks is solved by a numerical algorithm, but text is a sequence of symbols.

So, we must first turn language into **numbers**:

1. First we cut text into tokens.
2. Then, we give each token a vector.

Text $\rightarrow$ tokens

A **token** is the atomic unit of text the model manipulates (in this case, a word).

Tokenization maps raw text to a sequence of tokens. For example:

“The black cat sleeps on the warm mat.” $\rightarrow$ [the, black, cat, sleeps, on,
the, warm, mat]

The **vocabulary** V is the set of distinct tokens we keep:

$$V = \{\text{black, cat, mat, on, sleeps, the, warm}\}.$$

Here $n := |V| = 7$, but in practice $n \sim 10^4$ – 10^6 .

Tokens $\rightarrow$ vectors

An “intuitive” algorithm: fix an ordering $V = \{x_1, \dots, x_n\}$ of the vocabulary and encode the word $w = x_i$ by its **one-hot vector** $e_i \in \mathbb{R}^n$, which has a 1 in the slot corresponding to w in the vocabulary and 0 in every other slot. Listing V alphabetically, we get the following data for our example:

black	$\mapsto (1, 0, 0, 0, 0, 0, 0)$	cat	$\mapsto (0, 1, 0, 0, 0, 0, 0)$
mat	$\mapsto (0, 0, 1, 0, 0, 0, 0)$	on	$\mapsto (0, 0, 0, 1, 0, 0, 0)$
sleeps	$\mapsto (0, 0, 0, 0, 1, 0, 0)$	the	$\mapsto (0, 0, 0, 0, 0, 1, 0)$
warm	$\mapsto (0, 0, 0, 0, 0, 0, 1)$		

Our sentence is now a list of eight vectors in $\mathbb{R}^7$ (a vector space), something a numerical algorithm can use.

Why one-hot encoding is not enough

But this encoding is a poor one because:

- The dimension n is huge, and every vector is almost entirely zeros (i.e., it is sparse).
- Every pair of distinct words sits at the same distance $\sqrt{2}$: for example, “cat” is no closer to “dog” than it is to “democracy”.

So the encoding carries no information at all about meaning.

We want short vectors whose geometry carries meaning.

Definition (Word embedding)

Let V be a finite vocabulary and let $d \in \mathbb{N}$ be an integer with $d \ll |V|$, called the **embedding dimension**. A **word embedding** is a map $\mathcal{E}: V \rightarrow \mathbb{R}^d$, $w \mapsto \mathbf{v}_w$, assigning to each word $w \in V$ a vector $\mathbf{v}_w \in \mathbb{R}^d$ [Mikolov et al., 2013].

- Discrete tokens are replaced by **elements of a vector space**, so the tools of geometry (distance, angle, linear combination) become usable [Collobert et al., 2011].
- Similar words end up close together.
- Linguistics even shows up as **vector arithmetic** [Mikolov et al., 2013]:

$$\text{vec}(\text{King}) - \text{vec}(\text{Man}) + \text{vec}(\text{Woman}) \approx \text{vec}(\text{Queen}).$$

But, how do we choose the map $\mathcal{E}$?

Predicting a word from its context

Assumption (The distributional hypothesis)

A word in a text is related to the surrounding words composing its context. [Harris, 1954]

So we set ourselves a **prediction task** where we hide one word, and predict it from its neighbours.

the black cat ? on the warm mat

This is the **CBOW** model (continuous bag of words) [Mikolov et al., 2013], and it is the model we study for the rest of the talk.

The context window

Definition (Context window)

Let $(w_1, \dots, w_T)$ be a sequence of tokens, and let $C \geq 1$ be an integer, called the **context half-window size**. Define $\mathcal{C} := \{-C, \dots, -1, 1, \dots, C\}$. For an index t satisfying $C + 1 \leq t \leq T - C$, the **context window** of the target token w_t is

$$\text{ctx}_C(w_t) := \{w_{t+j} : j \in \mathcal{C}\}.$$

Thus the context consists of the C tokens preceding w_t and the C tokens following w_t .

Like $n = |V|$, the half-window C is a **hyperparameter** we choose, and it reappears in every bound we prove later. Sliding the window along the sentence, each interior position t gives one **data point**: input $\text{ctx}_C(w_t)$, expected output w_t .

The softmax for turning scores into probabilities

Definition (Softmax)

Let $z = (z_w)_{w \in V} \in \mathbb{R}^n$ be a vector of real scores. The **softmax** of z [Bridle, 1990] is the vector $p = (p_w)_{w \in V}$ given by

$$p_w = \frac{e^{z_w}}{\sum_{w' \in V} e^{z_{w'}}} \in (0, 1),$$

so that $\sum_{w \in V} p_w = 1$.

The main observations are that:

- A higher score implies a higher probability.
- The exponential keeps everything positive.
- Dividing by the sum makes the probabilities add up to one.

The CBOW model

Each word gets two vectors: the **input** embedding v_w (kept as “the embedding”) and the **output** embedding u_w .

Definition (CBOW model)

Let w_t be a target token and let $h := \frac{1}{2C} \sum_{j \in \mathcal{C}} v_{w_{t+j}}$ be the average of the embeddings of $\text{ctx}_C(w_t)$. The **CBOW model** with parameters $\Theta = \{v_w, u_w\}_{w \in V}$ assigns to each $w \in V$ the probability

$$\mathbb{P}_{\Theta}(w \mid \text{ctx}_C(w_t)) = \frac{e^{\langle u_w, h \rangle}}{\sum_{w' \in V} e^{\langle u_{w'}, h \rangle}}.$$

The mirror model, skip-gram [Mikolov et al., 2013b], goes from target $\rightarrow$ context; everything that follows has a skip-gram analogue, but we present only CBOW.

A loss function

Averaging the error over every data point in the corpus gives the **negative log-likelihood**, large when the model gives the observed words a low probability:

$$\mathcal{L}(\Theta) = -\frac{1}{T} \sum_{t=1}^T \log \mathbb{P}(w_t \mid \{w_{t+j}\}_{j \in \mathcal{C}}; \Theta).$$

The parameters Θ are all the vectors v_w, u_w , so $\mathcal{L}$ is a function of the embeddings alone.

Solving the prediction task is what produces the embeddings.

Training is optimization

Training means choosing the parameters $\Theta \in \mathbb{R}^p$ that make this error as small as possible.

In other words, it is an **optimization problem** of the following form:

$$\min_{\Theta \in \mathbb{R}^p} \mathcal{L}(\Theta) = \mathbb{E}_{s \sim \mathcal{D}}[\mathcal{L}_s(\Theta)].$$

Here:

- s is a single (context, target) example, and $\mathcal{L}_s$ is the error on that one example.
- $s \sim \mathcal{D}$ means s is drawn from $\mathcal{D}$, the distribution of examples in the corpus.
- The expectation is the average error over the corpus.

The problem is **stochastic**: the corpus is far too large to evaluate that average exactly, so each step uses a random mini-batch as a stand-in for the whole corpus.

How we solve it using gradient descent

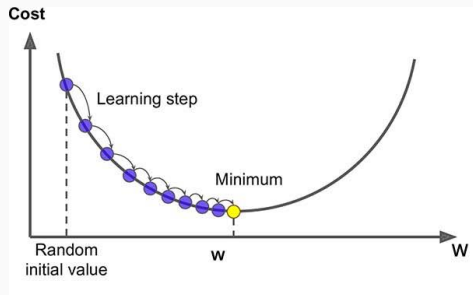


Figure. Step downhill, and stop where the slope vanishes ^a.

^a<https://editor.analyticsvidhya.com/uploads/97106gd4.jpeg>

The gradient $\nabla \mathcal{L}(\Theta)$ points in the direction of steepest increase, so we move against it.

Every method we consider takes the form:

$$\Theta_{k+1} = \Theta_k - \tau_k \dot{\Theta}_k,$$

- $\dot{\Theta}_k$: search direction (which way to move)
- τ_k : learning rate / step size (how far to move)

If τ_k is too small, progress is slow; if too large, the iteration overshoots and diverges.

First-order and Second-order optimization methods

Both families use the update rule from the previous slide, and differ only in the information they use to build the step.

- **First-order methods:** construct $\dot{\Theta}_k$ from gradients only (SGD [Robbins & Monro, 1951], Adam [Kingma & Ba, 2015]).
- **Second-order methods:** also use the curvature $\nabla^2 \mathcal{L}(\Theta_k)$, the second derivative.

Where the ideal step size comes from

How should we choose τ_k ?

- We can approximate $\mathcal{L}$ near Θ_k by its **second-order Taylor expansion** along the search direction:

$$\mathcal{L}(\Theta_k - \tau \dot{\Theta}_k) \approx \mathcal{L}(\Theta_k) - \tau \langle \nabla \mathcal{L}(\Theta_k), \dot{\Theta}_k \rangle + \frac{\tau^2}{2} \langle \nabla^2 \mathcal{L}(\Theta_k) \dot{\Theta}_k, \dot{\Theta}_k \rangle.$$

The right-hand side is a parabola in τ .

- Minimizing that parabola achieves the ideal step

$$\tau_k^* = \frac{\langle \dot{\Theta}_k, \nabla \mathcal{L}(\Theta_k) \rangle}{\langle \nabla^2 \mathcal{L}(\Theta_k) \dot{\Theta}_k, \dot{\Theta}_k \rangle},$$

recovering **Newton's method** when $\dot{\Theta}_k = (\nabla^2 \mathcal{L})^{-1} \nabla \mathcal{L}$, $\tau_k = 1$.

- The numerator is only a gradient. **The denominator involves the Hessian.**

Problem (1) with Hessian: size

$\nabla^2 \mathcal{L} \in \mathbb{R}^{p \times p}$ has p^2 entries, and $p \sim [10^6, 10^9]$:

Parameters p	Hessian entries	Memory
6×10^5 (our small model!)	3.6×10^{11}	≈ 2.9 TB
10^6	10^{12}	≈ 8 TB
10^9	10^{18}	≈ 8 <u>exabytes</u>

Storing it is already out of reach, and inverting it requires $O(p^3)$ operations on top.

Problem (2) with Hessian: nonconvexity

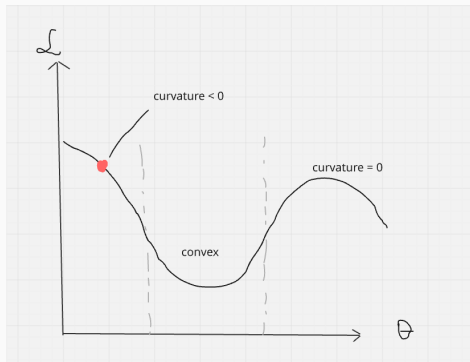


Figure. A loss that is convex only in places; it curves up between the dashed lines and down outside them.

A **convex** loss curves upward everywhere, so the local parabola always opens up.

A neural network loss is **not** convex: **its eigenvalues can vanish or turn negative** [Botev et al., 2017].

Since τ_k^* came from minimizing that parabola:

- At the **red point** the parabola opens downward and has no minimum, so τ_k^* comes out negative and the step runs **uphill**.
- At the dashed lines the curvature passes through 0, the denominator of τ_k^* vanishes, and the step size **grows large**.

Preconditioning and the alternative

The update $\Theta_{k+1} = \Theta_k - \tau_k \dot{\Theta}_k$ has two ingredients: a direction and a step size.

- Every preconditioning method (Shampoo [Gupta et al., 2018], SOAP [Vyas et al., 2025], AdaHessian [Yao et al., 2021], Sophia [Liu et al., 2024], L-BFGS [Liu & Nocedal, 1989], Apollo [Ma, 2021], and so on) devotes its curvature effort on the **direction**, replacing $\dot{\Theta}_k = \nabla \mathcal{L}(\Theta_k)$ by $\dot{\Theta}_k = P_k \nabla \mathcal{L}(\Theta_k)$, where the **preconditioner** P_k approximates $(\nabla^2 \mathcal{L})^{-1}$.
- But P_k is a matrix, approximated, stored, and applied at every step, which is exactly what we wanted to avoid.
- **RED** [de Gournay & Gossard, 2022] leaves the direction alone and uses its effort on the **step size**, rescaling it by a single scalar.

Our central question

Can second-order curvature information accelerate the training of word2vec models, relative to the first-order and adaptive methods used in practice?

Curvature bounds for Word2Vec

The gradient and the Hessian

Write $\mathcal{L}: \mathbb{R}^p \rightarrow \mathbb{R}$ for the loss and $\Theta = (\theta_1, \dots, \theta_p)$ for the parameters.

- The **gradient** is the vector of the p first partial derivatives, $(\nabla \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial \theta_i} \in \mathbb{R}^p$, and points in the direction of steepest increase.
- The **Hessian** is the symmetric matrix of all p^2 second partial derivatives,

$$(\nabla^2 \mathcal{L})_{ij} = \frac{\partial^2 \mathcal{L}}{\partial \theta_i \partial \theta_j} \in \mathbb{R}^{p \times p},$$

and it shows how the slope itself changes.

- The Hessian is the only term in τ_k^* we cannot evaluate directly, and removing that obstacle is what the rest of this section does.

Directional notation

Everything from here on is measured along one fixed **search direction** $\dot{\Theta} = \{\dot{v}_w, \dot{u}_w\}$, a small change of Θ with the same shape.

Definition (Directional derivative)

Let $x: \mathbb{R}^p \rightarrow \mathbb{R}$ be differentiable at Θ . The derivative of x at Θ along $\dot{\Theta}$ is

$$\dot{x} := \langle \nabla x(\Theta), \dot{\Theta} \rangle = \left. \frac{d}{ds} x(\Theta + s\dot{\Theta}) \right|_{s=0}.$$

- $\dot{z}_w, \dot{h}, \dot{u}_w$ are the score, the averaged context vector, and the output embedding, each differentiated along $\dot{\Theta}$; for instance $\dot{h} = \frac{1}{2C} \sum_{j \in \mathcal{C}} \dot{v}_{w_{t+j}}$.
- Two constants enter the bounds: λ , the ℓ_2 regularization strength, and B , a bound on $\|v_w\|$ and $\|u_w\|$ throughout training.

What we will show

Recall that $\nabla^2 \mathcal{L}$ is unusable as a matrix.

But τ_k^* never needed the whole matrix, only the **scalar** $\langle \nabla^2 \mathcal{L} \dot{\Theta}, \dot{\Theta} \rangle$, the curvature along the one direction we are moving in.

For word2vec specifically, this section proves:

1. For CBOW it is a variance plus a small residual.
2. With mild ℓ_2 regularization ¹ the denominator of τ_k^* never vanishes or turns negative, so the rescaling is well-defined throughout training.
3. It requires the same order of work as the gradient.

¹ ℓ_2 regularization is added to the loss to prevent overfitting.

Definition (Directional curvature)

Let $\mathcal{L}: \mathbb{R}^p \rightarrow \mathbb{R}$ be twice differentiable, let $\Theta \in \mathbb{R}^p$, and let $\dot{\Theta} \in \mathbb{R}^p \setminus \{0\}$. The **directional curvature** of $\mathcal{L}$ at Θ along $\dot{\Theta}$ is

$$c_{\mathcal{L}}(\Theta, \dot{\Theta}) := \frac{\langle \nabla^2 \mathcal{L}(\Theta) \dot{\Theta}, \dot{\Theta} \rangle}{\|\dot{\Theta}\|^2}.$$

This is the only number RED needs.

Closed form for CBOW curvature

Theorem (Kulkarni–Sheng–Wu, 2026)

Fix a CBOW training example with target word w_t , and let $p_w := \mathbb{P}_\Theta(w \mid \text{ctx}_C(w_t))$, with $w \in V$. Let W be a V -valued random variable with distribution $\mathbb{P}(W = w) = p_w$. Then, for every parameter direction $\dot{\Theta}$,

$$\langle \nabla^2 \mathcal{L}_{\text{CBOW}}(\Theta) \dot{\Theta}, \dot{\Theta} \rangle = \text{Var}_p(\dot{z}_W) + 2 \langle \mathbb{E}_p[\dot{u}_W] - \dot{u}_{w_t}, \dot{h} \rangle.$$

No Hessian is ever formed; the curvature is a variance of the directional score changes plus a single inner product, both of which we already have.

Curvature is strictly positive and bounded

Theorem (Kulkarni–Sheng–Wu, 2026)

For $\lambda \geq 0$, define the ℓ_2 -regularized loss $\mathcal{J}_\lambda^{\text{CBOW}}(\Theta) := \mathcal{L}_{\text{CBOW}}(\Theta) + \frac{\lambda}{2}\|\Theta\|^2$ and, for every nonzero direction $\dot{\Theta}$, define $c_\lambda^{\text{CBOW}}(\Theta, \dot{\Theta}) := \langle \nabla^2 \mathcal{J}_\lambda^{\text{CBOW}}(\Theta) \dot{\Theta}, \dot{\Theta} \rangle / \|\dot{\Theta}\|^2$. Suppose that there exists $B > 0$ such that $\max_{w \in V} \|u_w\| \leq B$ and $\max_{w \in V} \|v_w\| \leq B$. Then, for every $\dot{\Theta} \neq 0$,

$$\lambda - 2\sqrt{\frac{2}{C}} \leq c_\lambda^{\text{CBOW}}(\Theta, \dot{\Theta}) \leq \lambda + 2B^2(1 + \frac{1}{2C}) + 2\sqrt{\frac{2}{C}}.$$

In particular, if $\lambda > 2\sqrt{2/C}$, then $c_\lambda^{\text{CBOW}}(\Theta, \dot{\Theta}) > 0$ for every $\dot{\Theta} \neq 0$.

The regularizer shifts every curvature up by exactly λ , so the denominator of τ_k^* stays strictly positive and bounded, and both bounds tighten as C grows.

Time complexity of the algorithm

For a general net, RED [de Gournay & Gossard, 2022] evaluates the curvature by automatic differentiation [Pearlmutter, 1994], at $1.5\times$ the time and $2\times$ the memory of one gradient, or $2\times$ the time at equal memory.

Theorem (Kulkarni–Sheng–Wu, 2026)

Let V be a vocabulary of cardinality $n = |V|$, let d be the embedding dimension, and let C be the context half-window size. For a single CBOW training example and a fixed parameter-space direction $\dot{\Theta}$, the pair $(\nabla \mathcal{L}_{\text{CBOW}}(\Theta), \langle \nabla^2 \mathcal{L}_{\text{CBOW}}(\Theta) \dot{\Theta}, \dot{\Theta} \rangle)$ can be computed in $O((n + C)d)$ arithmetic operations.

The curvature reuses h, z, p in a **single pass**, and the lower bound makes the order tight. Moreover, for the full-softmax CBOW model, computing the gradient requires $\Omega(nd)$ operations.

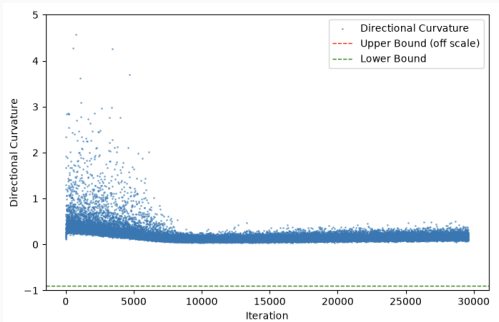
One pass returns both the gradient and the curvature

CBOW, on one training example

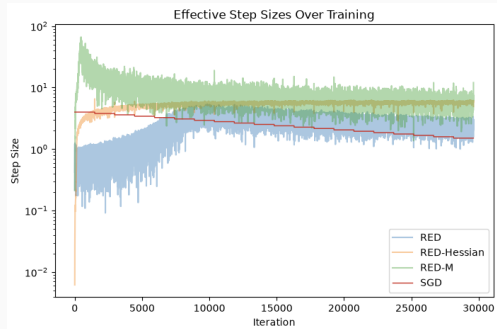
- 1: $\mathbf{h} \leftarrow \frac{1}{2C} \sum_{j \in \mathcal{C}} \mathbf{v}_{w_t+j}, \quad \mathbf{z}_w \leftarrow \langle \mathbf{u}_w, \mathbf{h} \rangle, \quad p_w \leftarrow \text{softmax}(\mathbf{z})_w$
- 2: gradient: $(p_w - [w=w_t]) \mathbf{h}$ and $\frac{1}{2C} (\mathbb{E}_p[\mathbf{u}_W] - \mathbf{u}_{w_t})$
- 3: $\dot{\mathbf{h}} \leftarrow \frac{1}{2C} \sum_{j \in \mathcal{C}} \dot{\mathbf{v}}_{w_t+j}, \quad \dot{\mathbf{z}}_w \leftarrow \langle \dot{\mathbf{u}}_w, \mathbf{h} \rangle + \langle \mathbf{u}_w, \dot{\mathbf{h}} \rangle$
- 4: $\kappa \leftarrow \text{Var}_p[\dot{\mathbf{z}}_W] + 2 \langle \mathbb{E}_p[\dot{\mathbf{u}}_W] - \dot{\mathbf{u}}_{w_t}, \dot{\mathbf{h}} \rangle$
- 5: **return** $\nabla \mathcal{L}_{\text{CBOW}}(\Theta)$ and κ

Lines 1 and 2 are the ordinary gradient. Only the two **highlighted** lines are new, and they reuse $\mathbf{h}, \mathbf{z}, \mathbf{p}$, so the total stays $\boxed{O((n+C)d)}$.

Empirical curvature testing



(a) Measured curvature for every mini-batch against the theoretical bounds. Every batch falls inside them.



(b) Effective step size per update. SGD is flat, while RED and RED-M stretch in flat regions and shorten in sharp ones.

Figure. Empirical curvature and step size behaviour during CBOW training.

Experiments

What are we testing?

The theory says curvature rescaling is well defined and cheap, not that it helps.

So we trained the same word2vec model from scratch six times, changing **only the optimizer**.

Everything else (corpus, vocabulary, architecture, batch size, seed) is held fixed.

We had three questions...

1. Does it converge in fewer epochs?
2. Does it converge in less time?
3. Are the resulting embeddings any good?

Data preprocessing and model parameters

- Simple English Wikipedia [\[simplewiki\]](#)
- Lowercased, 27-symbol alphabet.
- Stop-words removed; top $|V| = 10,000$ tokens.
- We used a subset containing 13 million words
- CBOW examples, window $C = 10$ (with $2C = 20$ neighbors).
- 90/10 train/test split, mini-batches of 8,096 examples.
- Embedding dimension $d = 300$.

The six optimizers

Optimizer	Family	Key idea
SGD [Robbins & Monro, 1951]	First-order	step $\eta = 4.0$, direction $= g_k$
Adam [Kingma & Ba, 2015]	Adaptive first-order	moment m_k + second moment v_k
AdaHessian [Yao et al., 2021]	Adaptive 2nd-order	replace g_k^2 by the Hessian diagonal
RED [de Gournay & Gossard, 2022]	Curvature-rescaled	direction $= g_k$, step $\propto 1/\tilde{L}_k$
RED-M [this work!]	RED + momentum	direction $= \hat{m}_k$, RED step size
RED-Hessian [this work!]	RED + Hessian	AdaHessian direction, RED step size

The last two rows are the optimizers we propose in this work. Both keep the curvature-rescaled step size of RED and change only the search direction.

The RED algorithm

RED keeps the gradient direction $d_k = g_k$ and puts everything into the step size.

Main idea: a flat region $\implies$ long step; sharp region $\implies$ short step.

Procedure:

1. **Measure** the curvature along d_k using our algorithm.
2. **Smooth** it, since one mini-batch is noisy: bias-corrected moving average $\tilde{c}_k$ of c_k with $\beta_3 = 0.9$, then $\tilde{L}_k = \max(\tilde{c}_k, c_k)$.
3. **Rescale** and step, with $\ell = 1$:

$$r_k = \frac{\langle d_k, g_k \rangle}{2\tilde{L}_k \|d_k\|^2},$$

$$\Theta_k = \Theta_{k-1} - \ell r_k d_k.$$

RED-M keeps the curvature rescaling but moves along a momentum direction instead of the raw gradient.

Momentum takes past gradients into account as well; it uses an exponential moving average of the recent gradients instead of a gradient itself.

- Direction: $d_k = \hat{m}_k$, where $m_k = \beta_1 m_{k-1} + (1 - \beta_1)g_k$ and $\hat{m}_k = m_k / (1 - \beta_1^k)$, with $\beta_1 = 0.9$.
- $c_k, \tilde{c}_k, \tilde{L}_k, r_k$ are formed exactly as in RED, now along this d_k .
- Update: $\Theta_k = \Theta_{k-1} - \ell r_k d_k$, again with $\ell = 1$.

Averaging past gradients smooths the direction, and the curvature still decides how far to travel along it.

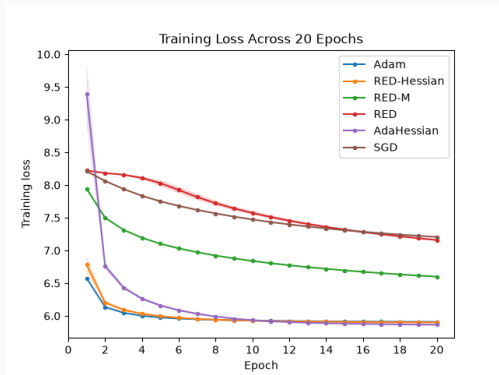
RED and AdaHessian use curvature in two different ways:

- RED uses curvature for its step-size, keeping the original gradient as its direction
- AdaHessian uses curvature for its direction

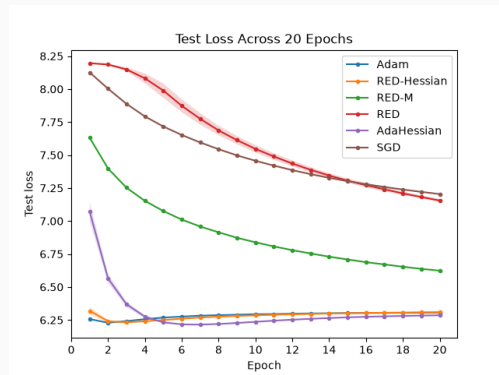
What if we use curvature for both?

- The direction is $d_k = \frac{\hat{m}_k}{\sqrt{\hat{v}_k + \epsilon}}$, where $\hat{v}_k$ is the exponential moving average of the Hutchinson estimate [\[Hutchinson, 1989\]](#) of the Hessian diagonal
- The step size is taken from RED, $r_k = \langle d_k, g_k \rangle / (2\tilde{L}_k \|d_k\|^2)$.
- Our update is $\Theta_k = \Theta_{k-1} - \ell r_k d_k$.

Loss convergence (per epoch)



(a) Training Loss

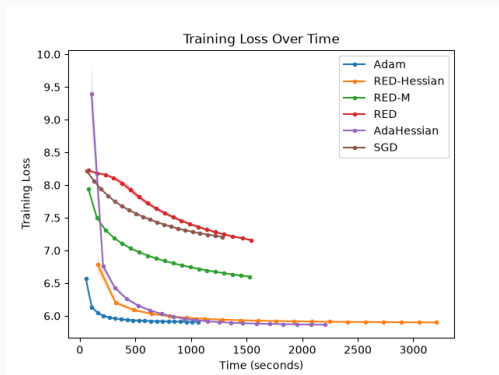


(b) Test Loss

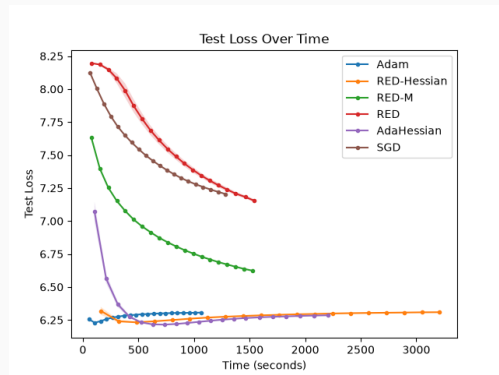
Figure. Training and test loss against epoch for the six optimizers.

Adaptive methods converge the fastest, RED performs comparably to SGD.

Loss convergence (over time)



(a) Training Loss

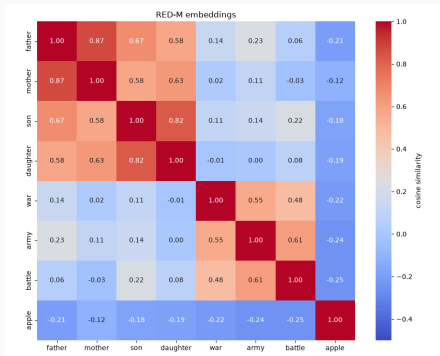


(b) Test Loss

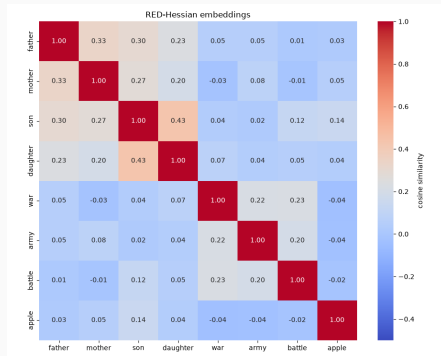
Figure. Training and test loss against wall clock time for the six optimizers.

Adaptive methods still converge quickly, second order methods are not more efficient. 38/52

Heatmap



(a) RED-M cosine similarities



(b) RED-Hessian cosine similarities

Figure. Cosine similarity heatmaps for the RED-M and RED-Hessian embeddings.

The word embeddings from both optimizers show the “distance” between words.

Introducing semantic benchmarks

- The training and testing loss graphs only show how well the models perform on the CBOW prediction task, which does not directly say anything about how good the actual word embeddings are.
- To directly evaluate the quality of word embeddings $V \rightarrow \mathbb{R}^d$, we also evaluated the model on two types of benchmarks:
 - Similarity benchmarks measure how “alike” or “related” two particular words are.
 - Analogy benchmarks test how consistently a relationship between concepts is represented.

Similarity benchmarks

Each benchmark is a list of word pairs, every one carrying a **human** score of how related the two words are. We ask whether our embeddings order those pairs the way people do.

The model's own score for a pair is the **cosine similarity**, the cosine of the angle between the two embedding vectors:

$$\cos(\mathbf{v}_a, \mathbf{v}_b) = \frac{\langle \mathbf{v}_a, \mathbf{v}_b \rangle}{\|\mathbf{v}_a\| \|\mathbf{v}_b\|} \in [-1, 1].$$

We use two datasets that ask different questions:

- SimLex-999 [Hill et al., 2015], strict similarity, on which *cat* and *dog* score **low**,
- WordSim-353 [Finkelstein et al., 2002], general relatedness, on which *cat* and *dog* score **high**.

Scoring a similarity benchmark

We do not compare the raw numbers, since human scores and cosines are not on the same scale. We compare the two **orderings** instead.

Definition (Spearman rank correlation)

Rank the m pairs once by human score and once by cosine similarity, and let d_i be the difference between the two ranks of pair i . Then

$$\rho = 1 - \frac{6 \sum_i d_i^2}{m(m^2 - 1)} \in [-1, 1].$$

So $\rho = 1$ means the two orderings are equal, and $\rho = 0$ means they are unrelated

[Spearman, 1904].

Analogy benchmarks

Here we test whether a relationship between two concepts is represented consistently across the whole vocabulary, a strictly harder demand than pairwise similarity.

Each question has the form “ a is to b as c is to ____?”. We answer it by **vector arithmetic**, computing

$$\text{vec}(b) - \text{vec}(a) + \text{vec}(c)$$

and returning the vocabulary words whose embeddings are closest to it in cosine similarity.

Our metric is **top- k accuracy**: the percentage of questions whose true answer appears among the k nearest words, for $k \in \{1, 3, 5\}$, with the three input words excluded.

The Google Analogy Test Set

The dataset [Mikolov et al., 2013] holds roughly 19,500 such questions across 14 categories, of two kinds:

- **Semantic** questions, about meaning; for example: capital and country, currency, city and state, family.

*Paris : France $\cong$ Tokyo : **Japan***

- **Syntactic** questions, about grammar; for example: plurals, comparatives, superlatives, past tense, adjective to adverb, nationality.

*good : better $\cong$ fast : **faster***

Within 20 epochs the test loss turns back upward for Adam, AdaHessian, and RED-Hessian: those models are overfitting the training split.

Reporting a fixed epoch would therefore punish the fast optimizers for a choice we control.

So every model is scored at its own epoch of minimum test loss, and we also plot how benchmark scores move as the overfitting sets in.

Benchmark results (Similarity)

Optimizer	Similarity (ρ)	
	SimLex-999	WordSim-353
RED-M	0.2171 $\pm$ 0.0035	0.5913 $\pm$ 0.0026
RED-Hessian	0.3099 $\pm$ 0.0132	0.6065 $\pm$ 0.0164
SGD	0.1486 $\pm$ 0.0050	0.4419 $\pm$ 0.0029
Adam	0.2631 $\pm$ 0.0137	0.5795 $\pm$ 0.0144
RED	0.1548 $\pm$ 0.0046	0.4569 $\pm$ 0.0060
AdaHessian	0.3123 $\pm$ 0.0083	0.6118 $\pm$ 0.0053

Figure. Results of the six optimizers on similarity benchmarks (mean $\pm$ SE over 3 runs).

The curvature-shaped directions, AdaHessian and RED-Hessian, lead on both benchmarks, and SGD and RED trail.

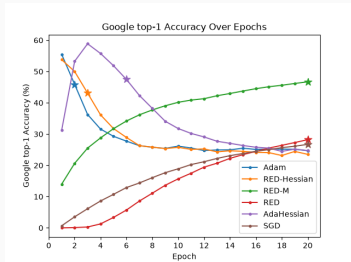
Benchmark results (Analogy)

Optimizer	Google Analogy (%)		
	Top-1	Top-3	Top-5
RED-M	46.90 $\pm$ 0.39	61.93 $\pm$ 0.16	67.44 $\pm$ 0.26
RED-Hessian	45.64 $\pm$ 3.07	63.15 $\pm$ 3.37	69.31 $\pm$ 3.64
SGD	27.13 $\pm$ 0.63	38.20 $\pm$ 0.56	43.14 $\pm$ 0.31
Adam	46.24 $\pm$ 0.36	64.17 $\pm$ 0.42	70.20 $\pm$ 0.13
RED	28.80 $\pm$ 0.56	40.55 $\pm$ 1.21	45.57 $\pm$ 1.07
AdaHessian	47.27 $\pm$ 1.24	65.06 $\pm$ 1.50	71.07 $\pm$ 1.41

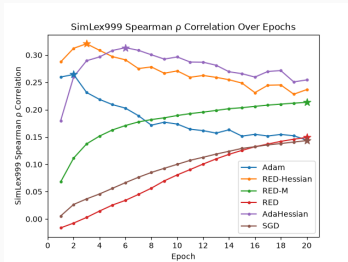
Figure. Results of the six optimizers on analogy benchmarks (mean $\pm$ SE over 3 runs).

Four optimizers cluster near 46% top-1 while SGD and RED sit near 28%; the direction matters more than the rescaled step alone.

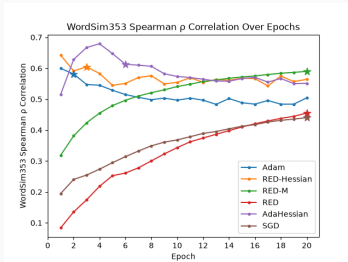
Lowest loss $\neq$ Highest quality



(a) Google Analogy Test Set



(b) SimLex999



(c) WordSim353

Figure. Benchmark score against epoch.

The fast optimizers (Adam, AdaHessian, RED-Hessian) peak early and then decline as they overfit.

Lowest loss $\neq$ Highest quality

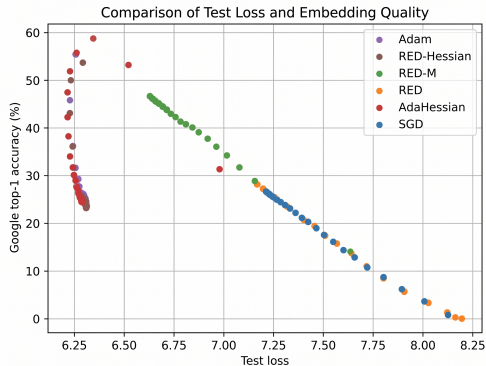


Figure. Google analogy benchmark top-1 accuracy against test loss at each epoch.

At the lowest test loss, embedding quality spans a large range, likely due to overfitting.

Conclusion

- For CBOW, the curvature has a closed form.
- With ℓ_2 regularization $\lambda > 2\sqrt{2/C}$, the denominator of τ_k^* is bounded away from zero, so the RED rescaling is well defined throughout training.
- Curvature and gradient can be computed in one pass in $O((n + C)d)$ time.
- Curvature does not accelerate training enough to justify its additional computational cost.
- Achieving the lowest CBOW loss does not automatically guarantee better embedding quality.

References

- [1] Botev, A., Ritter, H., and Barber, D. (2017). [Practical Gauss–Newton optimisation for deep learning](#). *ICML*.
- [2] Bridle, J. S. (1990). [Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition](#). *Neurocomputing, NATO ASI Series*.
- [3] Collobert, R., et al. (2011). [Natural language processing \(almost\) from scratch](#). *JMLR*.
- [4] de Gournay, F., and Gossard, A. (2022). [Adaptive scaling of the learning rate by second order automatic differentiation](#). *arXiv:2210.14520*.
- [5] Finkelstein, L., et al. (2002). [Placing search in context: the concept revisited](#) (WordSim-353). *ACM TOIS*.
- [6] Gupta, V., Koren, T., and Singer, Y. (2018). [Shampoo: Preconditioned stochastic tensor optimization](#). *ICML*.
- [7] Harris, Z. S. (1954). [Distributional structure](#). *Word*.
- [8] Hill, F., Reichart, R., and Korhonen, A. (2015). [SimLex-999: Evaluating semantic models with \(genuine\) similarity estimation](#). *Computational Linguistics*.
- [9] Hutchinson, M. F. (1989). [A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines](#). *Comm. Statist. Simulation Comput*.
- [10] Kingma, D. P., and Ba, J. (2015). [Adam: A method for stochastic optimization](#). *ICLR*.
- [11] Liu, D. C., and Nocedal, J. (1989). [On the limited memory BFGS method for large scale optimization](#). *Mathematical Programming*.
- [12] Liu, H., et al. (2024). [Sophia: A scalable stochastic second-order optimizer for language model pre-training](#). *ICLR*.
- [13] Ma, X. (2021). [Apollo: An adaptive parameter-wise diagonal quasi-Newton method for nonconvex stochastic optimization](#). *arXiv:2009.13586*.
- [14] Mikolov, T., et al. (2013). [Efficient estimation of word representations in vector space; Distributed representations of words and phrases and their compositionality](#). *ICLR Workshop / NeurIPS*.
- [15] Pearlmutter, B. A. (1994). [Fast exact multiplication by the Hessian](#). *Neural Computation*.
- [16] Robbins, H., and Monroe, S. (1951). [A stochastic approximation method](#). *Annals of Mathematical Statistics*.
- [17] Schraudolph, N. N. (2002). [Fast curvature matrix-vector products for second-order gradient descent](#). *Neural Computation*.
- [18] [Simple English Wikipedia](#). Plain-text corpus, first 10M characters (dataset).
- [19] Spearman, C. (1904). [The proof and measurement of association between two things](#). *American Journal of Psychology*.
- [20] Vyas, N., et al. (2025). [SOAP: Improving and stabilizing Shampoo using Adam](#). *ICLR*.
- [21] Yao, Z., et al. (2021). [AdaHessian: An adaptive second order optimizer for machine learning](#). *AAAI*.

Acknowledgements

The authors

- give a warm thanks to **Dr. Willy Rodriguez**, their mentor, for providing the reading material and insight that launched this project, and for invaluable feedback and guidance throughout our work,
- are very thankful and indebted to the **PRIMES-USA** program for the opportunity to take part in this research,
- wanted to thank the organizers of **SWIM 2026** for the opportunity of presenting our work.