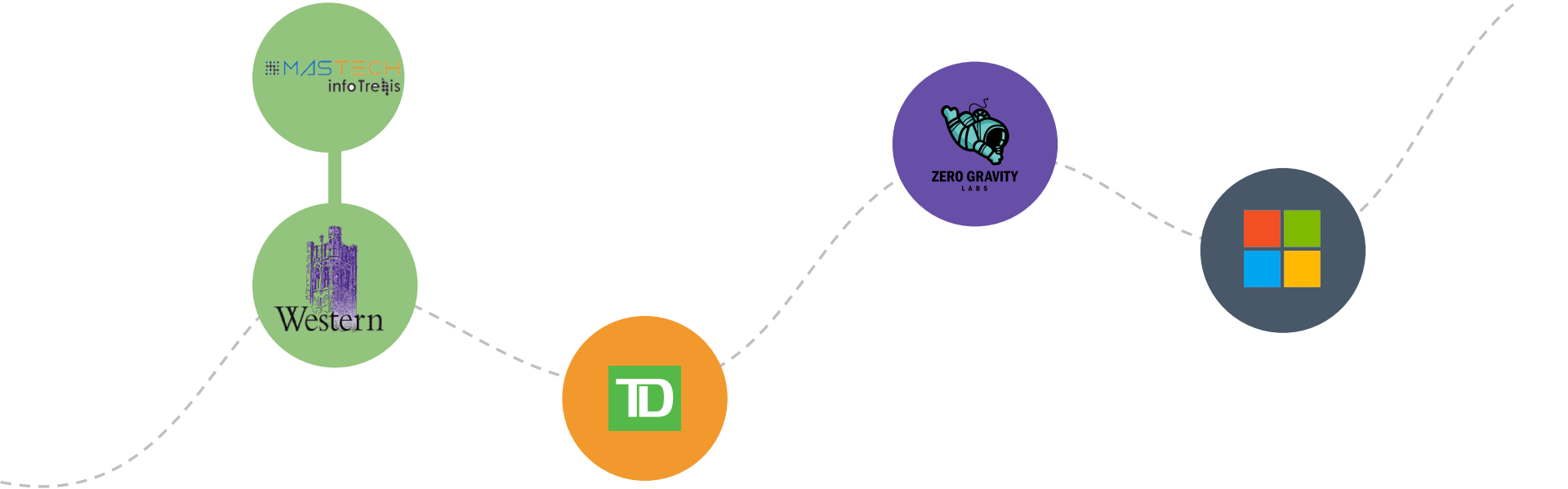
Word2Bits

Quantized Word Vectors

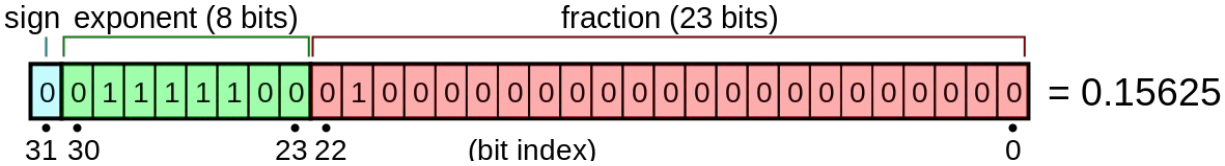
Author: *Maximilian Lam, Stanford University*
Publication date (arXiv): 31 Mar 2018



Taraneh Khazaei

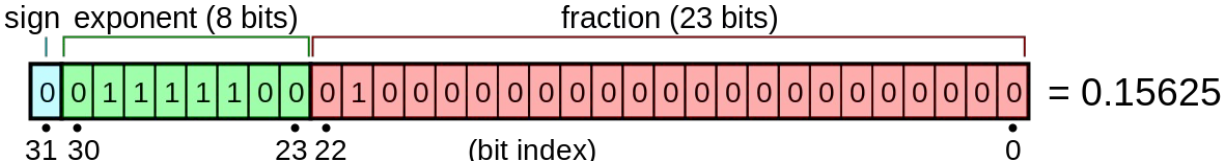
Word Vectors

[-0.038191, -0.24482, 0.72811, ..., -0.145, 0.8278, 0.2706]



Word Vectors

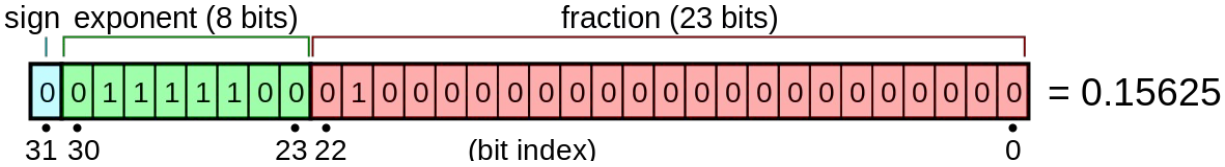
[-0.038191, -0.24482, 0.72811, ..., -0.145, 0.8278, 0.2706]



~3-6 GB Memory/Storage

Word Vectors

[-0.038191, -0.24482, 0.72811, ..., -0.145, 0.8278, 0.2706]



~3-6 GB Memory/Storage

Azure

Overview

- Layer of compression on top of pre-trained vectors
  - Dimensionality reduction
  - Pruning
  - Deep compositional coding

Overview

- Layer of compression on top of pre-trained models
  - Dimensionality reduction
  - Pruning
  - Deep compositional coding

Computationally intensive
Degrade accuracy

Overview

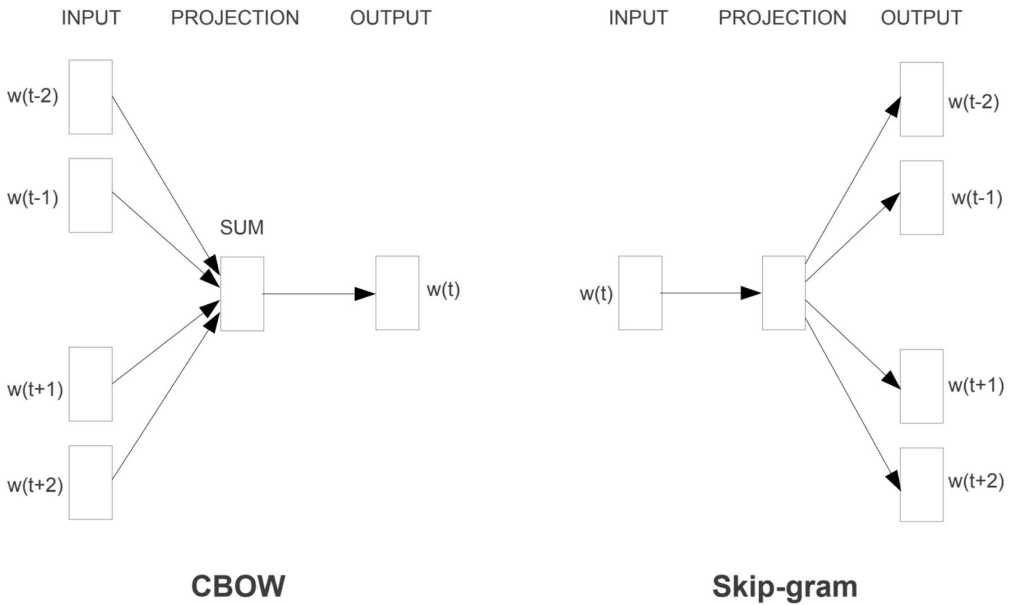
- Learning high quality representations directly
  - Network pruning
  - Deep compression
  - Quantization
    - Physical quantization: training with low-precision arithmetic/floats
    - Virtual quantization: training with full precision but constrained values

Overview

- Learning high quality representations directly
  - Network pruning
  - Deep compression
  - Quantization
    - Physical quantization: training with low-precision arithmetic/floats
    - Virtual quantization: training with full precision but constrained values

Word2Vec

- Skip-Gram: predicts *context* words from the target words
- CBOW: predicts *target* words from bag of words context



Word2Vec

- Skip-Gram: predicts *context* words from the target words
- CBOW: predicts *target* words from bag of words context

Negative Sampling: Each training sample only modify a small percentage of the weights, rather than all of them.

CBOW Loss Function

$$J(u_o, \hat{v}_c) = -log(\sigma(u_o^T \hat{v}_c)) - \sum_{i=1}^{k} log(\sigma(-u_i^T \hat{v}_c))$$

Vector of centre word

$$\hat{v}_c = \frac{1}{2w} \sum_{-w+o \leq i \leq w+o, i \neq o} v_i$$

Vector of context word

CBOW Loss Function

$$J(u_o, \hat{v}_c) = -log(\sigma(u_o^T \hat{v}_c)) - \sum_{i=1}^{k} log(\sigma(-u_i^T \hat{v}_c))$$

Vector of centre word

$$\hat{v}_c = \frac{1}{2w} \sum_{-w+o \leq i \leq w+o, i \neq o} v_i$$

Vector of context word

$$\frac{\partial J(u_o, \hat{v}_c)}{\partial u_o}$$

$$\frac{\partial J(u_o, \hat{v}_c)}{\partial u_i}$$

$$\frac{\partial J(u_o, \hat{v}_c)}{\partial v_i}$$

CBOW Loss Function

$$J(u_o, \hat{v}_c) = -log(\sigma(u_o^T \hat{v}_c)) - \sum_{i=1}^{k} log(\sigma(-u_i^T \hat{v}_c))$$

Vector of centre word

$$\hat{v}_c = \frac{1}{2w} \sum_{-w+o \leq i \leq w+o, i \neq o} v_i$$

Vector of context word

$$\frac{\partial J(u_o, \hat{v}_c)}{\partial u_o}$$

$$\frac{\partial J(u_o, \hat{v}_c)}{\partial u_i}$$

$$\frac{\partial J(u_o, \hat{v}_c)}{\partial v_i}$$

$$u_i + v_i$$

CBOW Loss Function - Quantized

$$Q_{bitlevel}(x) = \text{ quantization function to quantize to } bitlevel \text{ bits}$$

$$u_o^{(q)} = Q_{bitlevel}(u_o) \qquad \hat{v}_c^{(q)} = \sum_{-w+o \leq i \leq w+o, i \neq o} Q_{bitlevel}(v_i)$$

$$J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)}) = -log(\sigma((u_o^{(q)})^T \hat{v}_c^{(q)})) - \sum_{i=1}^{k} log(\sigma((-u_i^{(q)})^T \hat{v}_c^{(q)}))$$

Quantization Function

$$Q_1(x) = \begin{cases} \frac{1}{3} & x \geq 0 \\ -\frac{1}{3} & x < 0 \end{cases}$$

$$Q_2(x) = \begin{cases} \frac{3}{4} & x > \frac{1}{2} \\ \frac{1}{4} & 0 \leq x \leq \frac{1}{2} \\ -\frac{1}{4} & -\frac{1}{2} \leq x < 0 \\ -\frac{3}{4} & x < -\frac{1}{2} \end{cases}$$

Quantization Function

$$Q_1(x) = \begin{cases} \frac{1}{3} & x \geq 0 \\ -\frac{1}{3} & x < 0 \end{cases}$$

$$Q_2(x) = \begin{cases} \frac{3}{4} & x > \frac{1}{2} \\ \frac{1}{4} & 0 \leq x \leq \frac{1}{2} \\ -\frac{1}{4} & -\frac{1}{2} \leq x < 0 \\ -\frac{3}{4} & x < -\frac{1}{2} \end{cases}$$

Hinton's straight-through estimator:

$$\frac{\partial Q_{bitlevel}(x)}{\partial x} = I$$

$$\frac{\partial J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)})}{\partial u_o} = \frac{\partial J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)})}{\partial u_o^{(q)}}$$

$$\frac{\partial J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)})}{\partial u_i} = \frac{\partial J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)})}{\partial u_i^{(q)}}$$

$$\frac{\partial J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)})}{\partial v_i} = \frac{\partial J_{quantized}(u_o^{(q)}, \hat{v}_c^{(q)})}{\partial v_i^{(q)}}$$

Intrinsic Experimentation - Similarity/Analogy

Word Vector Type	Bits per parameter	Dimension	WordSim Similarity	WordSim Relatedness	MEN	M. Turk	Rare Words	SimLex	Google Add / Mul	MSR Add / Mul
Full Precision	32	200	.740	.567	.716	.635	.403	.317	.706/.702	.447/.447
	32	400	.735	.533	.720	.623	.408	.335	**.722**/.734	**.473**/.486
	32	800	.726	.500	.713	.615	.395	.337	.719/.735	.471/**.489**
	32	1000	.741	.529	.745	.617	.400	.358	.664/.675	.423/.434
Thresholded	T1	200	.692	.480	.668	.575	.347	.288	.371/.369	.186/.182
	T1	400	.677	.446	.686	.581	.369	.321	.533/.540	.286/.292
	T1	800	.728	.494	.692	.576	.383	.338	.599/.609	.333/.346
	T1	1000	.689	.504	.694	.551	.358	.342	.521/.520	.303/.305
Quantized	1	800	.772	.653	.746	.612	.417	.355	.619/.660	.395/.390
	1	1000	.768	**.677**	.756	.638	**.425**	.372	.650/.660	.371/.408
	1	1200	**.781**	.628	.765	**.643**	.415	.379	.659/.692	.391/.429
	2	400	.752	.604	.741	.616	.417	.373	.666/.690	.396/.418
	2	800	.776	.634	**.767**	.642	.390	**.403**	.710/.739	.418/.460
	2	1000	.752	.594	.764	.602	.362	.387	.720/**.750**	.436/.482

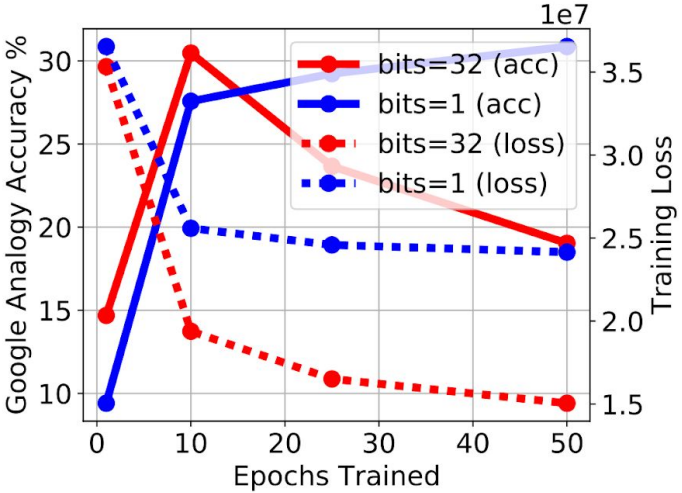
- trained 2017 English Wikipedia , w = 10, k = 12, learning rate = 0.05

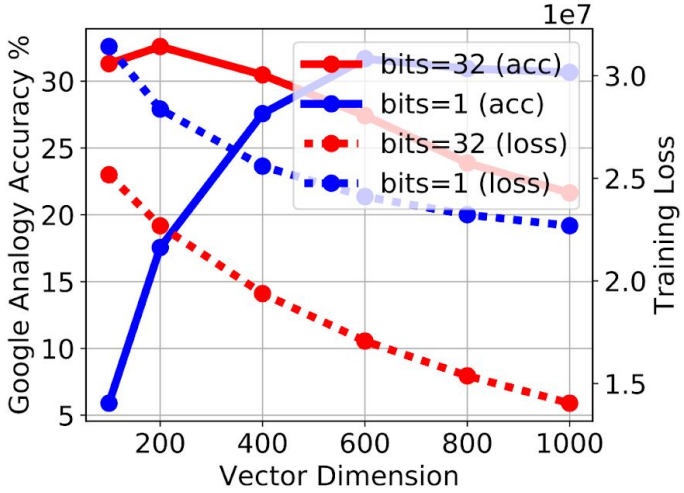
Extrinsic Experimentation - Question Answering

Word Vector Type	Bits per parameter	Dimension	Bytes per word	F1
Full Precision	32	200	800	75.25
	32	400	1600	75.28
	32	800	3200	75.31
	32	1000	4000	9.99
Quantized	1	800	100	76.64
	1	1000	125	76.84
	1	1200	150	76.50
	2	400	100	**77.04**
	2	800	200	76.12
	2	1000	250	75.66

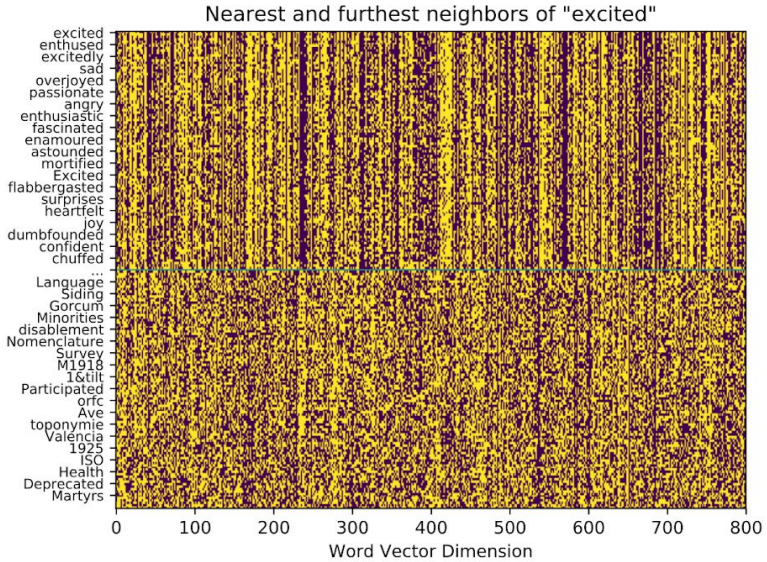
- trained 2017 English Wikipedia , w = 10, k = 12, learning rate = 0.05
- Facebook DrQA for the Stanford Question Answering task (SQuAD)

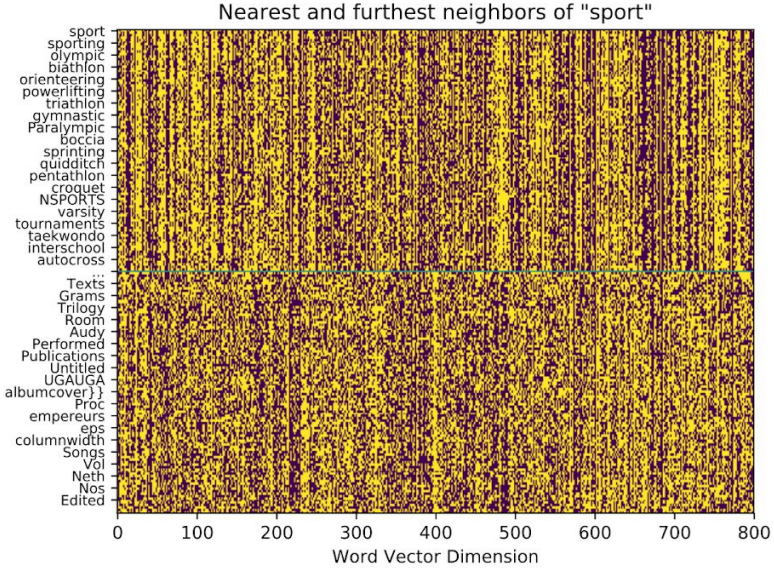
Word2Bits and Regularization





Word2Bits Visualization





In a nutshell...

- 1-bit quantized vector for *"king"* looks something like:

  [0.33333334**,** -0.33333334**,** 0.33333334**, ...,** -0.33333334**,** 0.33333334**]**

- It is possible to train high quality quantized vectors that take considerably less storage/memory.
- The code can be found here: https://github.com/agnusmaximus/Word2Bits