Word2Bits - Quantized Word Vectors

Maximilian Lam
maxlam@stanford.edu

Abstract

Word vectors require significant amounts of memory and storage, posing issues to resource limited devices like mobile phones and GPUs. We show that high quality quantized word vectors using 1-2 bits per parameter can be learned by training Word2Vec with a quantization function. We furthermore show that training with the quantization function acts as a regularizer. We train word vectors on full english Wikipedia (2017) and evaluate them on standard word similarity and analogy tasks and on question answering (SQuAD). Our quantized word vectors not only take 10-16x less space than full precision (32 bit) word vectors but also outperform them on word similarity tasks and question answering.

1 Introduction

Word vectors are extensively used in deep learning models for natural language processing. Each word vector is typically represented as a 300-500 dimensional vector, with each parameter being 32 bits. As there are millions of words, word vectors may take up to 3-6 GB of memory/storage – a massive amount relative to other portions of a deep learning model[1]. These requirements pose issues to memory/storage limited devices like mobile phones and GPUs.

Furthermore, word vectors are often re-trained on application specific data for better performance in application specific domains[1]. This motivates directly learning high quality compact word representations rather than adding an extra layer of compression on top of pretrained word vectors which may be computational expensive and degrade accuracy.

Recent trends indicate that deep learning models can reach a high accuracy even while training in the presence of significant noise and perturbation[1]. It has furthermore been shown that high quality quantized deep learning models for image classification can be learned at the expense of more training epochs[1]. Inspired by these trends we ask: can we learn high quality word vectors such that each parameter is only one of two values, or one of four values (quantizing to 1 and 2 bits respectively)?

To that end we propose learning quantized word vectors by introducing a quantization function into the Word2Vec loss formulation – we call our simple change Word2Bits. While introducing a quantization function into a loss function is not new, to the best of our knowledge it is the first time it has been applied to learning compact word representations.

In this report we show that

- It is possible to train high quality quantized word vectors which take 10x-16x less storage/memory than full precision word vectors. Experiments on both intrinsic and extrinsic tasks show that our learned word vectors perform comparably or even better on many tasks.
- Standard Word2Vec may be prone to overfitting; the quantization function acts as a regularizer against it.

2 Related Work

Word vectors are continuous representations of words and are used by most deep learning NLP models. Word2Vec, introduced by Mikolov's groundbreaking papers[1], is a unsupervised neural network algorithm for learning word vectors from textual data. Since then, other groundbreaking algorithms (Glove, FastText) [1] have been proposed to learn word vectors using other properties of textual data. As of 2018 the most widely used word vectors are Glove, Word2Vec and FastText. This work focuses on how to learn memory/storage efficient word vectors through quantized training – specifically our approach extends Word2Vec to output high quality quantized word vectors.

Learning compact word vectors is related to learning compressed neural networks. Finding compact representations of neural networks date back to the 90's and include techniques like network pruning[1], knowledge distillation[1], deep compression[1] and quantization[1]. More recently, algorithmic and hardware advances have allowed training deep models using low precision floating-point and arithmetic operations[1] – this is also referred to as quantization. To distinguish between quantized training with low precision arithmetic/floats from quantized training with full precision arithmetic/floats but constrained values we term the first physical quantization and the latter virtual quantization.

Our technical approach follows that of neural network quantization for image classification[1], which does virtual quantization by introducing a sign function (a 1 bit quantization function) into the training loss function. The actual technique of backpropagating through a discrete function (the quantization function) has been thoroughly explored by Hinton[1] and Bengio[1].

Application wise, various techniques exist to compress word embeddings[1]. These approaches involve taking pre-trained word vectors and compressing them using dimensionality reduction[1], pruning[1], or more complicated approaches like deep compositional coding[1]. Such techniques add an extra layer of computation to compress pre-trained embeddings and also may degrade word vector performance[1].

To the best of our knowledge, current traditional methods of obtaining compact word vectors involve adding an extra layer of computation to compress pretrained word vectors[1] (as described previously). This may incur computational costs which may be expensive in context of retraining word vectors for application specific purposes and may degrade word vector performance[1]. Our proposed approach of directly learning quantized word vectors from textual data may amend these issues and is an alternative method of obtaining compact high quality word vectors. Note that these traditional compression methods may still be applied on the learned quantized word vectors.

3 Word2Bits - Quantized Word Embeddings

3.1 Background

Our approach utilizes the Word2Vec formulation of learning word vectors. There are two Word2Vec algorithms: Skip Gram Negative Sampling (SGNS) and Continuous Bag of Words (CBOW)[1] – our virtual quantization technique utilizes CBOW with negative sampling. The CBOW negative sampling loss function minimizes

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

where

$$u_o = \text{ vector of center word with corpus position } o$$

$$\hat{v}_c = \frac{1}{2w-1} \sum_{-w+o \leq i \leq w+o, i \neq o} v_i \text{ where } v_i \text{ is vector for context word, } w \text{ is window size, a hyperparameter}$$

$$k = \text{ number of negative samples, a hyperparameter}$$

Intuitively, minimizing this loss function optimizes vectors of words that occur in similar contexts to be "closer" to each other, and pushes vectors whose contexts are different "away". Specifically CBOW with negative sampling tries to predict the center word from context words.

Technically, to optimize this loss function, for each sentence we:

- Identify the center word u_o
- Compute the average of the context words $\hat{v}_c = \frac{1}{2w-1} \sum_{-w+o \leq i \leq w+o, i \neq o} v_i$ given window size w
- Draw k negative samples $u_1, u_2, .., u_k$
- Compute loss $J(u_o, \hat{v}_c) = -log(\sigma(u_o^T \hat{v}_c)) - \sum_{i=1}^{k} log(\sigma(-u_i^T \hat{v}_c))$
- Update center word vector u_o with gradient $\frac{\partial J(u_o, \hat{v}_c)}{\partial u_o}$
- Update negative word vector u_i with gradient $\frac{\partial J(u_o, \hat{v}_c)}{\partial u_i}$
- Update context word vector v_i with gradient $\frac{\partial J(u_o, \hat{v}_c)}{\partial v_i}$

Center vectors u_i and context vectors v_j are stored full precision. The final word vectors are the sums of the context and center vectors $u_i + v_i$ for each corresponding word. The resulting vectors are full precision.

3.2 Word2Bits Approach

To learn quantized word vectors we introduce virtual quantization into the CBOW loss function:

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

where

$$u_o^{(q)} = Q_{bitlevel}(u_o)$$

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

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

The following quantization functions were used (chosen based on what worked best)

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

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

Since $Q_{bitlevel}$ is a discrete function, its derivative is undefined at some points and 0 at others. To solve this we simply set the derivative of $Q_{bitlevel}$ to be the identity function:

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

This is also known as Hinton's straight-through estimator[1].

Like in the standard algorithm, we optimize $J_{quantized}$ with respect to u_i and v_j over a corpus of text. The final vector for each word is $Q_{bitlevel}(u_i + v_i)$; thus each parameter takes $bitlevel$ bits to represent.

Intuitively, although we are still updating u_i and v_j (full precision vectors), we are now optimizing their quantized counterparts $Q_{bitlevel}(u_i)$ and $Q_{bitlevel}(v_j)$ to capture the same corpus statistics as regular word vectors. While we are still training with full precision 32-bit arithmetic operations and 32-bit floating point values, the final word vectors we save to disk are quantized. The full process is presented below.

Algorithm 1 Word2Bits

procedure MYPROCEDURE
    stringlen ← length of *string*
    $i \leftarrow$ *patlen*
top:
    if $i >$ *stringlen* **then return** false
    $j \leftarrow$ *patlen*
loop:
    if *string*$(i) =$ *path*(j) **then**
        $j \leftarrow j - 1$.
        if *string*$(i) =$ *path*(j) **then**
            what
        $i \leftarrow i - 1$.
        goto *loop*.
        close;
    $i \leftarrow i + \max($*delta*$_1($*string*$(i)),$ *delta*$_2(j))$.
    goto *top*.

4 Experiments and Results

5 Discussion and Future Work

Acknowledgments

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609-616. Cambridge, MA: MIT Press.

- COMPRESSING WORD EMBEDDINGS VIA DEEP COMPOSITIONAL CODE LEARNING - FASTTEXT.ZIP: COMPRESSING TEXT CLASSIFICATION MODELS - FASTTEXT (Bag of Tricks for Efficient Text Classification) - Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance - Improving Distributional Similarity with Lessons Learned from Word Embeddings - Optimal brain damage - Second order derivatives for network pruning: Optimal brain surgeon - Learning both weights and connections for efficient neural networks - Distributed Representations of Words and Phrases and their Compositionality - Efficient Estimation of Word Representations in Vector Space - Binarized Neural Networks - Hinton, Geoffrey. Neural networks for machine learning. Coursera, video lectures, 2012. - GloVe: Global Vectors for Word Representation - Learned in Translation: Contextualized Word Vectors - HALP - Towards Lower Bounds on Number of Dimensions for Word Embeddings - Evaluation methods for unsupervised word embeddings - SQuAD - DrQA - WordSim353 Finkelstein - WordSim Similarity, WordSim Relatedness (Zesch et al., 2008; Agirre et al., 2009 - Bruni et al.s (2012) MEN - Radinsky et al.s (2011) Mechanical Turk - Luong et al.s (2013) Rare Words dataset - Hill et al.s (2014) SimLex-999 - Low precision arithmetic for deep learning. - Predicting parameters in deep learning - Compressing neural networks with the hashing trick - Compression of Neural Machine Translation Models via Pruning