A Word2DM Gradients

The objective function that SGNS optimises at each prediction with regard to model parameters θ is

$$J(\theta) = \log \sigma({v_t}^\top v_c) + \sum_{k=1}^{K} \log \sigma(-{v_t}^\top v_{w_k}) \quad (1)$$

where v_t is the embedding of target word, v_c is the embedding of the context word, and $v_1, v_2, ..., v_K$ are the embeddings of K negative samples. By optimising equation 1 over a large corpus, skip-gram learns word embeddings that encode distributional information.

Maximising equation 1 adjusts the embeddings of words occurring in the same context to be more similar and adjusts the embeddings of words that don't occur together to be less similar. This becomes clear when we consider the gradients used to update embeddings during training. We briefly recall the details of the gradient calculation so as to refer back to it later in this section. The derivative of equation 1 with respect to the target vector v_t is

$$\frac{\partial J}{\partial v_t} = (1-\sigma({v_t}^\top v_c))v_c - \sum_{k=1}^{K}(1-\sigma({v_t}^\top v_{w_k}))v_{w_k} \quad (2)$$

which is used to update the target vector as follows:

$$v_t \leftarrow v_t + \alpha \frac{\partial J}{\partial v_t} \quad (3)$$

The target vector is updated by adding the scaled context vector to it and subtracting the scaled negatively sampled vectors from it. The vectors are scaled proportionally to how dissimilar they are to the target vector. This ensures that the target vector is "pulled closer" to the true context vector and "pushed away" from the negative context vectors. It is this computationally simple training procedure which makes SGNS effective.

Word2DM extends SGNS to learn density matrices, replacing equation 1 with the following objective function:

$$J(\theta) = \log \sigma(\mathrm{tr}(A_t A_c)) + \sum_{k=1}^{K} \log \sigma(-\mathrm{tr}(A_t A_{w_k})) \quad (4)$$

where A_t and A_c are the density matrices of the target and context words respectively, $A_1, A_2, ..., A_K$ are the density matrices of K negative samples, and θ is the set of weights of the intermediary matrices B_t, B_c and $B_1, B_2, ..., B_K$.

Computing this objective function requires multiple matrix multiplications. For each $\mathrm{tr}(A_t A_c)$ term (including the terms of the K negative samples), the matrices A_t and A_c have to be computed respectively as $A_t = B_t B_t^\top$ and $A_c = B_c B_c^\top$ and then the matrix product $A_t A_c$ has to be computed. This means that, for each target-context prediction, we require $3(K+1)$ matrix multiplications. One of the most attractive features of SGNS is its computational efficiency, which enabled training on very large corpora in reasonable time. The introduction of multiple matrix multiplications into the objective function means that much of this efficiency is lost. In order to reduce the complexity of our model, we make use of the following property and lemma to find a new objective function that is computationally simpler, but equivalent to equation 4.

Property A.1. *The trace of the product of two matrices can be expressed as the sum of the element-wise products of their elements. If A is an $n \times m$ matrix and B is an $m \times n$ matrix, then the trace of the $n \times n$ matrix AB can be computed as*

$$\mathrm{tr}(AB) = \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij} b_{ji}$$

Lemma A.2. *If B_t and B_c are $n \times m$ intermediary matrices, then trace of the matrix product $A_t A_c$ can be written as the sum of the squared elements of an $m \times m$ matrix $C = B_c^\top B_t$:*

$$\mathrm{tr}(A_t A_c) = \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}^2$$

Proof. We can express $\mathrm{tr}(A_t A_c)$ as a trace computation involving intermediary matrices B_t and B_c:

$$\mathrm{tr}(A_t A_c) = \mathrm{tr}(B_t B_t^\top B_c B_c^\top)$$

Then we can use the cyclic property of the trace function to rewrite this as the product of a matrix C and its transpose:

$$\begin{aligned} \mathrm{tr}(A_t A_c) &= \mathrm{tr}(B_c^\top B_t B_t^\top B_c) \\ &= \mathrm{tr}(B_c^\top B_t (B_c^\top B_t)^\top) \\ &= \mathrm{tr}(CC^\top), \quad \text{where } C = B_c^\top B_t \end{aligned}$$

Now we can use property A.1 to express this as the element-wise products of the elements of C and its

transpose:

$$\begin{aligned}\mathrm{tr}(A_t A_c) &= \sum_{i=1}^{m}\sum_{j=1}^{m}[C]_{ij}[C^\top]_{ji} \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}c_{ij} \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}^2\end{aligned}$$

□

This allows us to rewrite equation 4 to find an equivalent objective function that requires fewer computations than straightforward matrix multiplication would. The objective function at each target-context prediction becomes

$$\begin{aligned}J(\theta) = &\log\sigma(\sum_{i=1}^{m}\sum_{j=1}^{m}[B_c^\top B_t]_{ij}^2) \\ &+ \sum_{k=1}^{K}\log\sigma(-\sum_{i=1}^{m}\sum_{j=1}^{m}[B_{w_k}^\top B_t]_{ij}^2).\end{aligned} \tag{5}$$

By using the result of lemma A.2 we have reduced the number of matrix multiplications required for each target-context prediction from $3(K+1)$ to $(K+1)$. Density matrices are trained by maximising equation 5 with respect to the intermediary matrices $B_t, B_c, B_{w_1}, ..., B_{w_K}$ over a large corpus.

The model is trained using stochastic gradient descent. We now derive the gradients used to update B_t during training, and subsequently show that these gradients lead to suboptimal updates to the density matrices during training. Deriving the gradient with respect to B_c and B_{w_k} would proceed similarly. To compute the gradients of equation 5 we first rewrite it in terms of the elements of the $n \times m$ matrices B_t, B_c, and B_{w_k}:

$$\begin{aligned}J(\theta) = &\log\sigma(\sum_{i=1}^{m}\sum_{j=1}^{m}(\sum_{l=1}^{n} b_{li}^c b_{lj}^t)^2) \\ &+ \sum_{k=1}^{K}\log\sigma(-\sum_{i=1}^{m}\sum_{j=1}^{m}(\sum_{l=1}^{n} b_{li}^{w_k} b_{lj}^t)^2),\end{aligned} \tag{6}$$

where b_{pq}^x denotes the pqth element of B_x. We derive the gradient of this objective function with respect to b_{pq}^t, an element of the intermediary target word matrix B_t. In order to use the chain rule in gradient calculations we rewrite $J(\theta)$ as a composite function:

$$J(\theta) = \log\sigma(y(\theta)) + \sum_{k=1}^{K}\log\sigma(z_k(\theta)), \tag{7}$$

where

$$y(\theta) = \sum_{i=1}^{m}\sum_{j=1}^{m}(\sum_{l=1}^{n} b_{li}^c b_{lj}^t)^2 \qquad \text{and}$$

$$z_k(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{m}(\sum_{l=1}^{n} b_{li}^{w_k} b_{lj}^t)^2.$$

The derivative of J with respect to b_{pq}^t can now be computed as follows:

$$\begin{aligned}\frac{\partial J}{\partial b_{pq}^t} =& \frac{\partial \log}{\partial\sigma}\frac{\partial\sigma}{\partial y}\frac{\partial y}{\partial b_{pq}^t} + \sum_{k=1}^{K}\frac{\partial \log}{\partial\sigma}\frac{\partial\sigma}{\partial z_k}\frac{\partial z_k}{\partial b_{pq}^t} \\ =& \frac{1}{\sigma(y)}(1-\sigma(y))\sigma(y)\frac{\partial y}{\partial b_{pq}^t} \\ &+ \sum_{k=1}^{K}\frac{1}{\sigma(z_k)}(1-\sigma(z_k))\sigma(z_k)\frac{\partial z_k}{\partial b_{pq}^t} \\ =& (1-\sigma(y))\frac{\partial y}{\partial b_{pq}^t} + \sum_{k=1}^{K}(1-\sigma(z_k))\frac{\partial z_k}{\partial b_{pq}^t} \\ =& (1-\sigma(y))\sum_{i=1}^{m} 2b_{pi}^c\sum_{l=1}^{n} b_{li}^c b_{lq}^t \\ &- \sum_{k=1}^{K}(1-\sigma(z_k))\sum_{i=1}^{m} 2b_{pi}^{w_k}\sum_{l=1}^{n} b_{li}^{w_k} b_{lq}^t \\ =& (1-\sigma(y))2[B_c B_c^\top B_t]_{pq} \\ &- \sum_{k=1}^{K}(1-\sigma(z_k))2[B_{w_k}B_{w_k}^\top B_t]_{pq}\end{aligned}$$

The last line in the above derivation is obtained by rewriting the summation expressions as equivalent matrix multiplications. We can now write the derivative of J with respect to the full intermediary matrix B_t:

$$\begin{aligned}\frac{\partial J}{\partial B_t} =& (1-\sigma(y(\theta)))2B_c B_c^\top B_t \\ &+ \sum_{k=1}^{K}(1-\sigma(z_k(\theta)))2B_{w_k}B_{w_k}^\top B_t\end{aligned} \tag{8}$$

As opposed to the gradients of Word2Vec (equation 2), the gradients of Word2DM do not lead to simple and easily interpretable training updates. As discussed in the paragraph following equation 3, in Word2Vec the target vector is made more similar to the context vector and less similar to the negative context vectors. Ideally we would like something

similar to occur in Word2DM with density matrices, but equation 8 shows that we lose the intuitive training updates of Word2Vec through the introduction of intermediary matrices. Furthermore, we can show that the gradients of Word2DM sometimes lead to unwanted consequences in training.

Consider the case where the density matrices of a target and context word are highly dissimilar. Recall from equation 4 that the y is in equation 8 is the trace inner product of the density matrices A_t and A_c (the measure we use to quantify semantic similarity). The minimum value of the trace inner product of two density matrices is zero (this follows from the fact that density matrices are positive semi-definite), so two density matrices are highly dissimilar when their trace inner product is close to zero i.e. $y \approx 0$. From equation 5 we can recall how y can be written in terms of the intermediary matrices:

$$y = \sum_{i=1}^{m}\sum_{j=1}^{m}[B_c^\top B_t]_{ij}^2$$

Consider that $y \approx 0$ if and only if the elements of $B_c^\top B_t$ are close to zero in value, since squaring the elements in the summation makes them all positive. We have established the following equivalence:

$$\mathrm{tr(A_tA_c)} \approx 0 \iff B_c^\top B_t \approx \mathrm{O},$$

where O is the $m \times m$ matrix with all zero entries. Consider how this will affect the target-context update during training. The first term of the gradient in equation 8 becomes

$$(1 - \sigma(y(\theta)))2B_cB_c^\top B_t = (1 - \sigma(0))2B_c\mathrm{O} \approx \mathrm{O}$$

so the target-context update becomes ineffective for true contexts. The update should make the density matrix of the target word more similar to that of the context word, but the gradient is so small that it makes this impossible. Moreover, the more dissimilar the target and context density matrices are before the update, the less effective the update will be. This is the opposite of the intended effect (achieved by Word2Vec) in which the magnitude of the target-context update should increase if the target and context representations are less similar. This is an example of how the introduction of intermediary matrices in Word2DM leads to suboptimal training updates. We ensure that our density matrices are positive semi-definite, but lose the guarantee that the algorithm will learn high-quality semantic representations.

B Hyperparameters for Word2DM and Multi-Sense Word2DM

Word2DM additional details We use a dynamic window size i.e. the size of each context window is sampled uniformly between 1 and the maximum window size. We also discard words that occur less than some minimum threshold and subsample frequently occurring words. Negative samples are drawn from a unigram distribution raised to the power of $\frac{3}{4}$. Furthermore, we train two density matrices for each word - one that represents it as a target word and another that represents it as a context word. After training we use the target density matrices as our final density matrices.

Hyperparameters We train our Word2DM and multi-sense Word2DM models on the ukWaC+Wackypedia corpus, consisting of 2.8 billion words. We use a window size of 5, a minimum word count of 50, 5 negative samples per positive context, and a subsampling rate of 1e-5. We train the model for 4 iterations of the ukWaC+Wackypedia corpus, using the Adam optimisation algorithm, a learning rate of 0.001, and 16 sentences per batch.