Advanced Methods in Text Analytics
Exercise 2: Deep Learning Basics and Word Embeddings
Solutions

Daniel Ruffinelli

FSS 2025

1 Fully-Connected Neural Networks

(a) As many as we need, so n input units, m output units, where n is not necessarily equal to m.

(b) Nodes hold values that are computed based on other nodes and some edges, whereas edges hold parameters of the function represented by the network.

(c) Generally, the input of the operation computed by each node is the nodes from the previous layer *that are connected to it by an edge.* Let's call these *input* nodes. The general operation is a linear combination between the value in the input nodes and the corresponding weight vector $\boldsymbol{w}_i$ made up of the values on the edges connecting the input vectors. Then, an *activation* function f is applied to the output of this linear combination. f may or may not be a linear function. Concretely, let $h_i^{(k)}$ be the (output) value of the i-th unit in the k-th hidden layer in the network. We have:

$$h_i^{(k)} = f\left(\sum_{j=1}^{L^{k-1}} h_j^{(k-1)} w_{i,j}^{(k)}\right),$$

where L^{k-1} is the size of the $(k-1)$-th layer and $w_{i,j}^{(k)}$ is the j-th component of $\boldsymbol{w}_i$. Note that we can write this more compactly in vectorized form:

$$h_i^{(k)} = f(\boldsymbol{h}_{(k-1)}^\top \boldsymbol{w}_k),$$

where both $\boldsymbol{h}, \boldsymbol{w}$ are column vectors, $\boldsymbol{h}_{(k-1)}$ is the vector with values in the $(k-1)$-th layer, and f is applied element-wise.

(d) The activation function. For example, ReLU, *tanh*, the logistic function.

(e) $\boldsymbol{H}_1 = f(\boldsymbol{X}\boldsymbol{W}^T) \in \mathbb{R}^{N\times p}$ where $\boldsymbol{W} \in \mathbb{R}^{p\times d}$ is the matrix constructed by stacking the weight vectors of each hidden unit in the first layer as rows, and f is an activation function that is applied element-wise.

2 Backpropagation

(a) $\frac{\partial}{\partial \boldsymbol{w}} L$ where L is our loss function and $\boldsymbol{w}$ the model parameters we want to adjust during learning.

(b) We use standard derivation rules followed by some rewriting to obtain the desired result.

$$\begin{aligned}
\sigma(x)' &= \frac{d\sigma(x)}{d(x)} \\
&= \frac{d(\frac{1}{1+e^{-x}})}{dx} \\
&= -\frac{1}{(1+e^{-x})^2} \cdot \frac{d(1+e^{-x})}{dx} \\
&= -\frac{1}{(1+e^{-x})^2} \cdot (e^{-x}) \cdot \frac{d(-x)}{dx} \\
&= \frac{1}{(1+e^{-x})^2} \cdot (e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\
&= \frac{1}{(1+e^{-x})} - \frac{1}{(1+e^{-x})^2} \\
&= \sigma(x) - \sigma(x)^2 \\
&= \sigma(x) \cdot (1 - \sigma(x))
\end{aligned}$$

(c) In our setting, the network produces two numbers $y_1, y_2 \in [0,1]$, each corresponding to the probability of one class. The given labels match this setting, since they are given by a pair of binary numbers. E.g. label $[0,1]$ means the example corresponds to the 2nd class, not the first. Using squared error loss, we want the model to match its outputs to the given labels during training, so for label $[0,1]$, the model should match y_1 to 0 and y_2 to 1. This is achieved with the loss we were given:

$$L = -\frac{1}{2}\frac{1}{N}\sum_{e=1}^{N}\sum_{l=1}^{2}(t_{e,l} - y_{e,l})^2.$$

where N is the number of training examples (which we will keep as N for clarity during derivations). Also, we add the 1/2 factor to simplify gradient computations. This is commonly done because is does not affect optimization as the results are proportional to the exact results.

(i) Let h_j be the value of hidden unit j, and $w_{i,j}^{(k)}$ the weight from input feature i in layer $k-1$ to neuron j in layer k. Our network architecture tells us that

$$\begin{aligned}
y_1 &= \sigma(w_{1,1}^{(2)} \cdot h_1 + w_{2,1}^{(2)} \cdot h_2) \\
y_2 &= \sigma(w_{1,2}^{(2)} \cdot h_1 + w_{2,2}^{(2)} \cdot h_2).
\end{aligned}$$

We derive the required gradient w.r.t. $w^{(2)}_{1,2}$ for the update rule as follows:

$$\begin{aligned}
\frac{\partial L}{\partial w^{(2)}_{1,2}} &= -\frac{1}{2}\frac{1}{N}\sum_{e=1}^{N}\sum_{l=1}^{2}\frac{\partial}{\partial w^{(2)}_{1,2}}\left(t_{e,l}-y_{e,l}\right)^2 \\
&= -\frac{1}{N}\sum_{e=1}^{N}\left((-1)(t_{e,1}-y_{e,1})\cdot\frac{\partial y_{e,1}}{\partial w^{(2)}_{1,2}}+(t_{e,2}-y_{e,2})\cdot\frac{\partial y_{e,2}}{\partial w^{(2)}_{1,2}}\right) && \text{(}\frac{\partial y_{e,1}}{\partial w^{(2)}_{1,2}}\text{ evaluates to 0)} \\
&= -\frac{1}{N}\sum_{e=1}^{N}(t_{e,2}-y_{e,2})\cdot\frac{\partial y_{e,2}}{\partial w^{(2)}_{1,2}} && \text{(use gradient from (b))} \\
&= -\frac{1}{N}\sum_{e=1}^{N}(t_{e,2}-y_{e,2})\cdot y_{e,2}\cdot(1-y_{e,2})\cdot\frac{\partial(w^{(2)}_{1,2}\cdot h_1+w^{(2)}_{2,2}\cdot h_2)}{\partial w^{(2)}_{1,2}} && \text{(chain rule)} \\
&= -\frac{1}{N}\sum_{e=1}^{N}(t_{e,2}-y_{e,2})\cdot y_{e,2}\cdot(1-y_{e,2})\cdot h_1 \\
&= -\frac{1}{N}\sum_{e=1}^{N}\delta_{e,2}\cdot h_1,
\end{aligned}$$

where $\delta_{e,2} = (t_{e,2}-y_{e,2})\cdot y_{e,2}\cdot(1-y_{e,2})$. Similarly, our network architecture tells us that:

$$\begin{aligned}
h_1 &= \sigma(w^{(1)}_{1,1}\cdot x_1 + w^{(1)}_{2,1}\cdot x_2 + w^{(1)}_{3,1}\cdot x_3) \\
h_2 &= \sigma(w^{(1)}_{1,2}\cdot x_1 + w^{(1)}_{2,2}\cdot x_2 + w^{(1)}_{3,2}\cdot x_3).
\end{aligned}$$

Then, we obtain our gradient w.r.t. $w^{(1)}_{2,2}$ as follows:

$$\begin{aligned}
\frac{\partial L}{\partial w^{(1)}_{2,2}} &= -\frac{1}{2N}\sum_{e=1}^{N}\sum_{l=2}^{2}\frac{\partial}{\partial w^{(1)}_{2,2}}\left(t_{e,l}-y_{e,l}\right)^2 \\
&= -\frac{1}{N}\sum_{e=1}^{N}\left((t_{e,1}-y_{e,1})\cdot\frac{\partial y_{e,1}}{\partial w^{(1)}_{2,2}}+(t_{e,2}-y_{e,2})\cdot\frac{\partial y_{e,2}}{\partial w^{(1)}_{2,2}}\right) \\
&= -\frac{1}{N}\sum_{e=1}^{N}\left(\delta_{e,1}\cdot\frac{\partial(w^{(2)}_{1,1}\cdot h_1+w^{(2)}_{2,1}\cdot h_2)}{\partial w^{(1)}_{2,2}}+\delta_{e,2}\cdot\frac{\partial(w^{(2)}_{1,2}\cdot h_1+w^{(2)}_{2,2}\cdot h_2)}{\partial w^{(1)}_{2,2}}\right) \\
&= -\frac{1}{N}\sum_{e=1}^{N}\left(\delta_{e,1}\cdot\frac{\partial w^{(2)}_{2,1}\cdot h_2}{\partial w^{(1)}_{2,2}}+\delta_{e,2}\cdot\frac{\partial w^{(2)}_{2,2}\cdot h_2}{\partial w^{(1)}_{2,2}}\right) && \text{(}\frac{\partial h_1}{\partial w^{(1)}_{2,2}}=0\text{)} \\
&= -\frac{1}{N}\sum_{e=1}^{N}\left(\delta_{e,1}\cdot w^{(2)}_{2,1}\cdot h_2\cdot(1-h_2)\cdot x_2+\delta_{e,2}\cdot w^{(2)}_{2,2}\cdot h_2\cdot(1-h_2)\cdot x_2\right) \\
&= -\frac{1}{N}\sum_{e=1}^{N}\cdot h_2\cdot(1-h_2)\cdot x_2\sum_{l=1}^{2}\delta_{e,l}\cdot w^{(2)}_{2,l}.
\end{aligned}$$

We can now write our update rules for each of the weights:

$$\begin{aligned}
w^{(2)}_{1,2} &:= w^{(2)}_{1,2} + \frac{1}{N}\sum_{e=1}^{N}(t_{e,2} - y_{e,2}) \cdot y_{e,2} \cdot (1 - y_{e,2}) \cdot h_1 \\
w^{(1)}_{2,2} &:= w^{(1)}_{2,2} + \frac{1}{N}\sum_{e=1}^{N} \cdot h_2 \cdot (1 - h_2) \cdot x_2 \sum_{l=1}^{2} \delta_{e,l} \cdot w^{(2)}_{2,l}
\end{aligned}$$

(ii) We compute our forward pass with $w^{(i)}_{j,k} = 1$ for all possible values of i, j, k and the two given training examples:

$$\boldsymbol{x} = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix}, \boldsymbol{h} = \begin{pmatrix} 0.73 \\ 0.73 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 0.81 \\ 0.81 \end{pmatrix} \qquad \text{Example 1}$$

$$\boldsymbol{x} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}, \boldsymbol{h} = \begin{pmatrix} 0.98 \\ 0.98 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 0.88 \\ 0.88 \end{pmatrix} \qquad \text{Example 2}$$

And finally, we compute our updates:

$$\begin{aligned}
w^{(2)}_{1,2} &:= 1 + \frac{1}{2}((1 - 0.81) \cdot 0.81 \cdot (1 - 0.81) \cdot 0.73 + (0 - 0.88) \cdot 0.88 \cdot (1 - 0.88) \cdot 0.98) \\
&:= 0.965 \\
w^{(1)}_{2,2} &:= 1 + \frac{1}{2}(0 + 1 \cdot (1 - 0.98) \cdot 0.98 \cdot ((1 - 0.88) \cdot 0.88 \cdot (1 - 0.98) \cdot 1 \\
&\qquad\qquad + (0 - 0.88) \cdot 0.88 \cdot (1 - 0.98) \cdot 1)) \\
&:= 0.99987.
\end{aligned}$$

3 Skip-Gram Basics

(a) It's multi-class classification where the number of classes is the number of words in a given vocabulary V. We can describe the task with the expression $p(c_j|w_i)$. In such a setting, it's common to obtain probabilities using the softmax function. Formally, let l_{w_i,c_j} be the logit score produced by a classifier trained for this task given center word w_i and context word c_j. We then have:

$$p(c_j|w_i) = \frac{\exp(l_{w_i,c_j})}{\sum_{k \in V} \exp(l_{w_i,c_k})},$$

where w_i is a word in position i and c_j is a word in a given context of size L, e.g. for L = 2, we have $j \in \{i-2, i-1, i+1, i+2\}$.

(b) The sliding window moves through the sentence to create the following context windows:

(1) The quick brown fox jumped over the lazy dog.

(2) The quick brown fox jumped over the lazy dog.

(3) The quick brown fox jumped over the lazy dog.

Context Window	Source Word	Target Word
1	The	quick
1	The	brown
2	quick	The
2	quick	brown
2	quick	fox
3	brown	The
3	brown	quick
3	brown	fox
3	brown	jumped
4	fox	quick
4	fox	brown
4	fox	jumped
4	fox	over
5	jumped	brown
5	jumped	fox
5	jumped	over
5	jumped	the
6	over	fox
6	over	jumped
6	over	the
6	over	lazy
7	the	jumped
7	the	over
7	the	lazy
7	the	dog
8	lazy	over
8	lazy	the
8	lazy	dog
9	dog	the
9	dog	lazy

Table 1: Positive examples for the original skip-gram task and a context window of size 2.

(4) The quick brown fox jumped over the lazy dog.

(5) The quick brown fox jumped over the lazy dog.

(6) The quick brown fox jumped over the lazy dog.

(7) The quick brown fox jumped over the lazy dog .

(8) The quick brown fox jumped over the lazy dog .

(9) The quick brown fox jumped over the lazy dog .

Here, green marks the center words and yellow marks the context words. Table 1 lists all examples generated for the original skip-gram task using a context window of size 2 and the given sentence.

(c) The skip-gram model learns two word vectors for each word in our vocabulary: one for when words are center (or source) words, and one for when words are context (or target) words. Let $\boldsymbol{W}$ be the matrix constructed by stacking vectors of center representations as rows, and $\boldsymbol{W}'$ the matrix constructed in the same way using context word representations. Then $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{W}'\}$. The size of these matrices, and thus the number of parameters in our skip-gram model, for our given vocabulary and vector size is:

$$\begin{aligned}\boldsymbol{W} &\in \mathbb{R}^{101425\times 300}\\ \boldsymbol{W}' &\in \mathbb{R}^{101425\times 300}.\end{aligned}$$

(d) We obtain vectors for a given word with a simple lookup function, i.e. a 1-to-1 mapping between words in our vocabulary and rows of $\boldsymbol{W}$ and $\boldsymbol{W}'$. To obtain a single (final) representation of a given word, we need to decide how to use the corresponding parameters in $\boldsymbol{W}$ and $\boldsymbol{W}'$. For example, we may discard the representations in $\boldsymbol{W}'$ and simply use $\boldsymbol{w}_{Frodo} = \boldsymbol{W}_{79}$, i.e. the 79th row of matrix $\boldsymbol{W}$. Or we may choose to concatenate the corresponding representations from $\boldsymbol{W}$ and $\boldsymbol{W}'$, i.e. $\boldsymbol{w}_{Frodo} = \boldsymbol{W}_{79} \oplus \boldsymbol{W}'_{79}$. Instead of concatenation, one could also use other operations between vectors, such as computing the sum or average between the two. Concatenation is a safe operation in that it does not change the learned features, at the expense of doubling the dimension size of final vectors.

4 Skip-Gram Training

(a) The likelihood of the entire text corpus T with a context window of size L is given by:

$$L(\boldsymbol{\theta}|T) = \prod_{t=1}^{|T|} \prod_{-L\leq j\leq L, j\neq 0} p(w_{t+j}|w_t, \boldsymbol{\theta}).$$

Note that here we assume all words are *conditionally* independent of each other *given* parameters $\boldsymbol{\theta}$, i.e. the embeddings. In other words, we assume the embeddings encode the dependencies between words. The corresponding negative log-likelihood function using empirical risk minimization is given by:

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{|T|}\sum_{t=1}^{|T|} \sum_{-L\leq j\leq L, j\neq 0} \log p(w_{t+j}|w_t, \boldsymbol{\theta}).$$

During training, we find the values of $\boldsymbol{\theta}$ that minimize this expression using gradient-based optimizers.

(b) We first need to obtain the vector for word *Frodo* when used as center (source) word. We can represent the necessary lookup function with a matrix-vector multiplication between our embedding matrix and the one-hot encoding vector that represents our word. Since *Frodo* is the 2nd word in V, we get the center word representation for *Frodo* as follows:

$$\begin{aligned}\boldsymbol{w}_{Frodo}^T &= [0,1,0,0,0,0]\cdot \boldsymbol{W}\\ &= [0.91, 0.74, 0.27].\end{aligned}$$

Given that our model computes logits using dot products, we need the dot products of $\boldsymbol{w}_{Frodo}$ with all other context vectors $\boldsymbol{w}'$, i.e. each row in $\boldsymbol{W}'$. We obtain this with the following operation:

$$\boldsymbol{l} = \boldsymbol{w}_{Frodo}^T \cdot \boldsymbol{W}'^T = [0.9111, 0.5924, 0.9265, 1.2742, 0.9519, 0.7397].$$

We then compute the log loss (i.e. cross-entropy) as follows:

$$\begin{aligned}
L_{CE}(ring|Frodo) &= -\ln p(ring|Frodo) && \text{(log loss)}\\
&= -\ln \frac{\exp(\boldsymbol{w}_{ring}^T \boldsymbol{w}_{Frodo})}{\sum_{v\in V}\exp(\boldsymbol{w}_v^T \boldsymbol{w}_{Frodo})} && \text{(softmax)}\\
&= -\boldsymbol{w}_{ring}^T \boldsymbol{w}_{Frodo} + \ln \sum_{v\in V}\exp(\boldsymbol{w}_v^T \boldsymbol{w}_{Frodo}) && \text{(log of division)}\\
&= -0.9265 + \ln(\exp(0.9111) + \ldots + \exp(0.7397))\\
&= -0.9265 + \ln(2.49 + 1.81 + 3.58 + 2.59 + 2.10)\\
&= -0.9265 + 2.53\\
&= 1.60.
\end{aligned}$$

Note that we get the desired behavior of a loss function. Namely, the model is penalized (i.e. the loss increases) when it assigns a high probability to negative examples. Conversely, the model is rewarded (i.e. the loss decreases) when it assigns a high probability to the positive example. Given that these probabilities are determined by dot products, we force the model to increase the similarity between vectors of words only when they are found in the same context, i.e. we follow the distributional hypothesis.

(c) For a given positive example (c, w) where c is target word and w is source word, we update the row of $\boldsymbol{W}$ that corresponds to w and all context vectors for all words in V except source word w, i.e. every row but one in $\boldsymbol{W'}$. This is because we use all our vocabulary in the denominator of the softmax function to compute the probability of our given positive example (c, w). Consequently, the cost of computing the probability of a single example, and thus the number of relevant parameters during backpropagation, is proportional to the size of V, which can be very large. This is why the authors used the binary clasification task with negative sampling discussed in the lecture, to minimize training costs. It can be shown that the binary classification task with negative sampling is an approximation to this approach.

5 Word Embeddings as Matrix Factorization

(a) We can explicitly express the sampling of negatives in the objective as follows:

$$L(w, c) = \log \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c) + k \cdot \mathbb{E}_{i\sim P}[\log \sigma(-\boldsymbol{w}_w^t \boldsymbol{w}_i)]. \tag{1}$$

In other words, each negative is weighted by its probability of being sampled. We can write this more explicitly in the following way:

$$L(w, c) = \log \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c) + k \cdot \sum_{i\in V} \frac{\#(i)}{|V|} \log \sigma(-\boldsymbol{w}_w^t \boldsymbol{w}_i). \tag{2}$$

(b) We can do this by applying Equation 2 over the entire training set as follows:

$$L = \sum_{w\in V}\sum_{c\in V} \#(w, c) \left(\log \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c) + k \cdot \sum_{i\in V} \frac{\#(i)}{|V|} \log \sigma(-\boldsymbol{w}_w^t \boldsymbol{w}_i) \right), \tag{3}$$

where $\#(w, c)$ is the number of times tuple (w, c) appears in the training set.

(c) In our objective, each training example (w, c) is represented by $\boldsymbol{w}_w^T\boldsymbol{w}_c$. Using the fact that the gradient is a linear operator, and that $\sum_{w'\in V}\sum_{c'\in V}\#(w', c') = \sum_{w'\in V}\#(w')$, we first write the gradient we need to compute as follows:

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c} &= \sum_{w'\in V}\sum_{c'\in V}\#(w',c')\left(\frac{\partial \log\sigma(\boldsymbol{w}_{w'}^t\boldsymbol{w}_{c'})}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c} + k\cdot\sum_{i\in V}\frac{\#(i)}{|V|}\frac{\partial\log\sigma(-\boldsymbol{w}_{w'}^t\boldsymbol{w}_i)}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c}\right)\\ &= \sum_{w'\in V}\sum_{c'\in V}\#(w',c')\frac{\partial \log\sigma(\boldsymbol{w}_{w'}^t\boldsymbol{w}_{c'})}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c} + k\cdot\sum_{w'\in V}\#(w')\cdot\sum_{i\in V}\frac{\#(i)}{|V|}\frac{\partial\log\sigma(-\boldsymbol{w}_{w'}^t\boldsymbol{w}_i)}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c}.\end{aligned}$$

We then compute this gradient, using the fact that $\sigma(-x) = 1 - \sigma(x)$ and starting with the given hint:

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c} &= \#(w,c)\frac{\partial\log\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c)}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c} + k\cdot\#(w)\cdot\frac{\#(c)}{|V|}\frac{1}{\sigma(-\boldsymbol{w}_w^t\boldsymbol{w}_c)}\frac{\partial(1-\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c))}{\partial \boldsymbol{w}_w^T\boldsymbol{w}_c}\\ &= \#(w,c)\frac{\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c)(1-\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c))}{\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c)} - k\cdot\#(w)\cdot\frac{\#(c)}{|V|}\frac{\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c)(1-\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c))}{\sigma(-\boldsymbol{w}_w^t\boldsymbol{w}_c)}\\ &= \#(w,c)\sigma(-\boldsymbol{w}_w^t\boldsymbol{w}_c) - k\cdot\#(w)\cdot\frac{\#(c)}{|V|}\sigma(\boldsymbol{w}_w^t\boldsymbol{w}_c).\end{aligned}$$

Now, we set this gradient to be equal to zero and solve for $\boldsymbol{w}_w^t\boldsymbol{w}_c$, which we set to $x = \boldsymbol{w}_w^t\boldsymbol{w}_c$ for simplicity:

$$\begin{aligned}\#(w,c)\sigma(-x) &= k\cdot\#(w)\cdot\frac{\#(c)}{|V|}\sigma(x)\\ \frac{\sigma(x)}{1-\sigma(x)} &= \frac{1}{k}\frac{\#(w,c)|V|}{\#(w)\#(c)}\\ \frac{1+e^{-x}}{e^{-x}(1+e^{-x})} &= \frac{1}{k}\frac{\#(w,c)|V|}{\#(w)\#(c)} && \text{(definition of } \sigma)\\ e^x &= \frac{1}{k}\frac{\#(w,c)|V|}{\#(w)\#(c)}\\ x &= \ln\left(\frac{1}{k}\frac{\#(w,c)|V|}{\#(w)\#(c)}\right) && \text{(applied logs)}\\ \boldsymbol{w}_w^t\boldsymbol{w}_c &= \ln\left(\frac{\#(w,c)|V|}{\#(w)\#(c)}\right) - \ln k && \text{(def. of } x, \text{log of division).}\end{aligned}$$

Recall that k is the number of negative examples generated per positive example in the training set (a hyperparameter). If we set $k = 1$, we have

$$\boldsymbol{w}_w^t\boldsymbol{w}_c = \log\left(\frac{\#(w,c)|V|}{\#(w)\#(c)}\right).$$

This expression is known as the pointwise mutual information (PMI) between w and c, a commonly used concept in NLP that was already covered in the basic Text Analytics course. This concept is based on information-theoretic principles and is defined as follows:

$$\mathrm{PMI}(w,c) = \log\frac{p(w,c)}{p(w)p(c)}.$$

In other words, PMI measures the log of the ratio between the joint probability of w and c, i.e. how often they occur together, and their marginal probabilities, i.e. how often they occur independently. This can be estimated by counting relative frequencies in a given corpus like so:

$$\mathrm{PMI}(w, c) = \log \frac{\#(w, c)}{\#(w)\#(c)} \propto \log \frac{\#(w, c)|V|}{\#(w)\#(c)}.$$

Intuitively, PMI tells us about how strongly (or weakly) related two words are by counting how often they are seen together in a corpus and contrasting this with how often they are seen independently. The derivation in this task shows that the optimal solution of the skip-gram approach as expressed by Equation 3 is the PMI matrix of our training corpus. We can thus relate skip-gram to count-based sparse word representations, because this allow us to interpret it as a form of learning a matrix factorization of the PMI matrix of our training corpus. This is similar to how latent semantic analysis (LSA) derives representations from the singular value decomposition of a co-occurrence matrix, as also seen in the basic Text Analytics course. Finally, note that if we set $k > 1$ as commonly done, we get a PMI matrix shifted by $\log k$. For more details, see Levy and Goldberg.