Advanced Methods in Text Analytics

Exercise 02: DL Basics and Word Embeddings

Daniel Ruffinelli

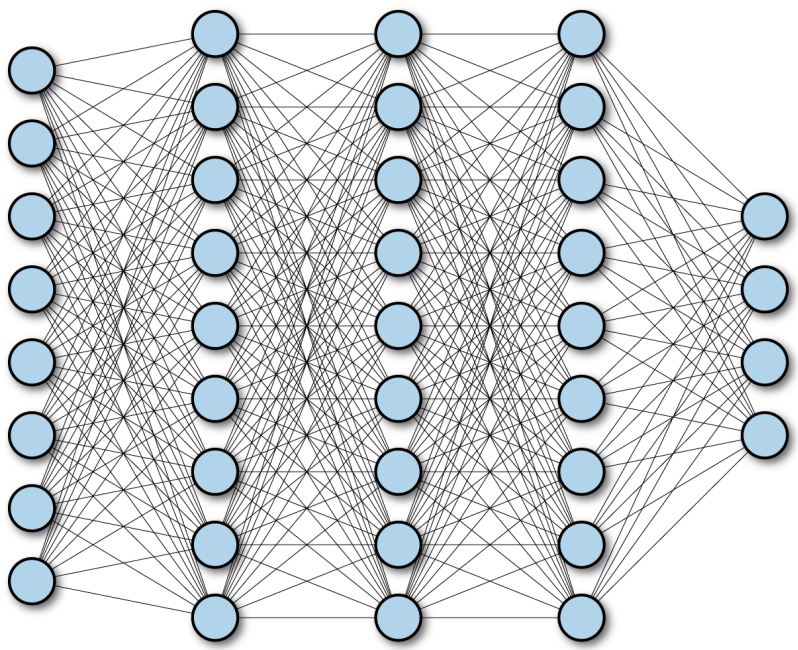
University of Mannheim

FSS 2025

Fully-Connected Neural Networks

Context

- An FNN is a compositional (possibly non-linear) function that transforms an input vector $\boldsymbol{x}$ into a vector of outputs $\boldsymbol{y}$.
- Each node is a unit that holds a numerical value.
- Each edge in the network also holds a numerical value.
- These units are combined to create *layers*, usually an *input* layer, an *output* layer and any number of *hidden* layers.



Fully-Connected Neural Networks

Question a)

- **Question:** How many units should we have in the input layer? And in the output layer?

Fully-Connected Neural Networks

Question a)

- **Question:** How many units should we have in the input layer? And in the output layer?
- **Answer:** As many as we need, so n input units, m output units, where n is not necessarily equal to m.

Fully-Connected Neural Networks

Question b)

- **Question:** What is the difference between the values in nodes and those in edges?

Fully-Connected Neural Networks

Question b)

- **Question:** What is the difference between the values in nodes and those in edges?
- **Answer:** Nodes hold values that are computed based on other nodes and some edges, whereas edges hold parameters of the function represented by the network.

Fully-Connected Neural Networks

Question c)

- What is the *general* operation performed by each unit (node) in the network? What is the input to this operation? Give a formal expression for it.

Fully-Connected Neural Networks

Answer c) (1)

- 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.*
- The 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.
- 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 $\boldsymbol{w}_i$.

Fully-Connected Neural Networks

Answer c) (2)

- 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.

Fully-Connected Neural Networks

Question d)

- **Question:** What is the main difference between different types of artificial neurons? Give two examples of different types of neurons.

Fully-Connected Neural Networks

Question d)

- **Question:** What is the main difference between different types of artificial neurons? Give two examples of different types of neurons.
- **Answer:** The activation function. For example, ReLU, *tanh*, the logistic function.

Fully-Connected Neural Networks

Question e)

- **Question:** Denote by $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ the input matrix constructed by stacking N input vectors $\boldsymbol{x}_i \in \mathbb{R}^d$ as rows. Similarly, denote by $\boldsymbol{H}_1$ the matrix that contains the corresponding representations computed by the first hidden layer, which has p units. What is the size of $\boldsymbol{H}_1$? Give a formal expression for the operation that results in $\boldsymbol{H}_1$ given input $\boldsymbol{X}$.

Fully-Connected Neural Networks

Question e)

- **Question:** Denote by $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ the input matrix constructed by stacking N input vectors $\boldsymbol{x}_i \in \mathbb{R}^d$ as rows. Similarly, denote by $\boldsymbol{H}_1$ the matrix that contains the corresponding representations computed by the first hidden layer, which has p units. What is the size of $\boldsymbol{H}_1$? Give a formal expression for the operation that results in $\boldsymbol{H}_1$ given input $\boldsymbol{X}$.
- **Answer:** $\boldsymbol{H}_1 = f(\boldsymbol{X}\boldsymbol{W}^T) \in \mathbb{R}^{N \times p}$ 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.

Backpropagation

Context

- With backpropagation, we can get gradient information that is useful to update the parameters of a multi-layer neural network during training.
- We start by computing the relevant gradients of the output layer and then continue backwards by computing the gradients of previous layers based on the gradients we computed before.

Backpropagation

Question a)

- **Question:** What are the *relevant gradients* we care about during training?

Backpropagation

Question a)

- **Question:** What are the *relevant gradients* we care about during training?
- **Answer:** $\frac{\partial}{\partial \boldsymbol{w}} L$ where L is our loss function and $\boldsymbol{w}$ the model parameters we want to adjust during learning.

Backpropagation

Question b)

- The update rules of gradient-based optimizers require that we compute gradients of the functions we are optimizing.
- For example, when backpropagating through a layer with sigmoid activations, we will need the gradient of the sigmoid function.
- Show that this gradient is given by:

$$\sigma(x)' = \sigma(x) \cdot (1 - \sigma(x)), \text{where } \sigma(x) = \frac{1}{1 + e^{-x}}$$

Backpropagation

Answer b) (1)

- We use standard derivation rules to get our result.

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

Backpropagation

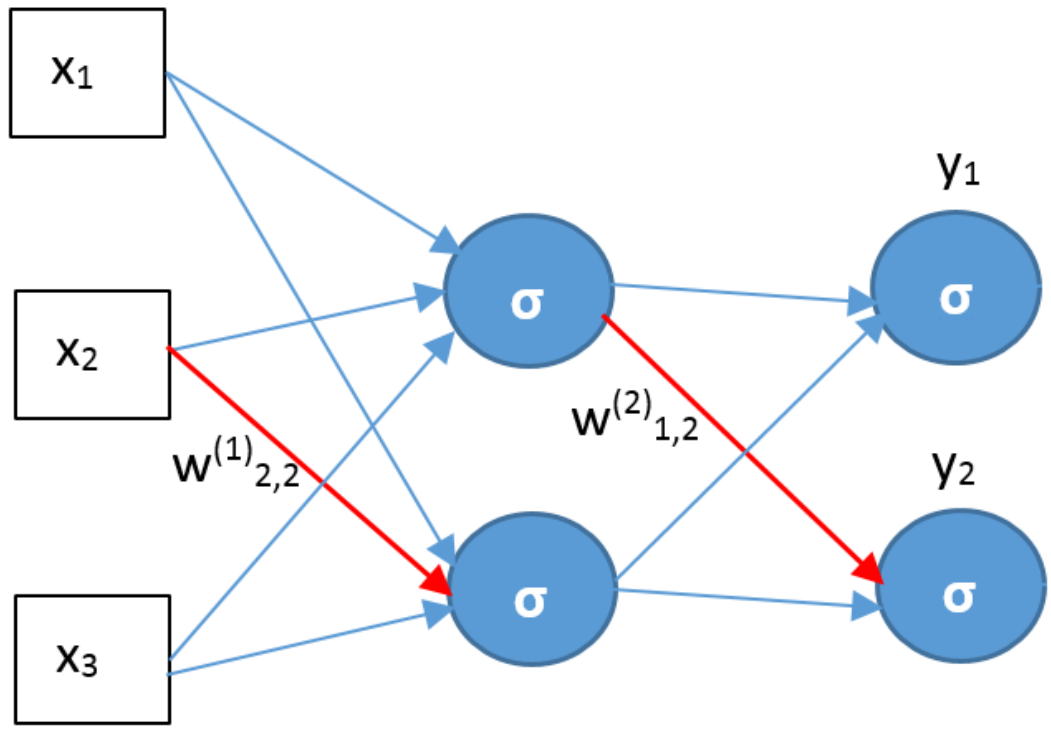
Answer b) (2)

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

Backpropagation

Question c) (Context)

- You are given a toy feed-forward neural network that predicts one of two classes given three input features: x_1, x_2, and x_3.
- The network has one hidden layer and one output layer, each consisting of two neurons.
- The activation function on both layers is the sigmoid function.



Backpropagation

Question c) (i)

- Derive the update rule for the weights marked in red in the image, i.e., for $w_{2,2}^{(1)}$ and $w_{1,2}^{(2)}$.
- Assume you are training your model with the following square error loss:

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

  where N is size of the training set, M is number of labels, $t_{e,l}$ the target label l of example e and $y_{e,l}$ the corresponding prediction made by the model.
- Note that in this toy setting, $M = N = 2$.
- Using square error loss for a classification task is not a common setting, but it is useful for us to construct a simple example to study backpropagation.

Backpropagation

Question c) (ii)

- You are given the toy dataset consisting of the following two training examples: $([-1, 0, 2], [0, 1])$ and $([2, 1, 1], [1, 0])$.
- These examples are of the form $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ is input features, and $\boldsymbol{y}$ is corresponding label.
- Assuming that all weights are initialized to $w = 1$, carry out one update iteration for denoted weights $w_{2,2}^{(1)}$ and $w_{1,2}^{(2)}$, using the two training examples for supervision.
- Use learning rate $\psi = 1$ and the square error loss given above.

Backpropagation

Answer c) (Some context)

- 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: 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, the $1/2$ factor simplifies computations, result unchanged.

Backpropagation

Answer c) (i) (1)

- 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

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

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

Backpropagation

Answer c) (i) (2)

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

where $\delta_{e,2}=(t_{e,2}-y_{e,2})\cdot y_{e,2}\cdot(1-y_{e,2})$.

Backpropagation

Answer c) (i) (3)

- Similarly, our network architecture tells us that:

$$h_1 = \sigma(w_{1,1}^{(1)} \cdot x_1 + w_{2,1}^{(1)} \cdot x_2 + w_{3,1}^{(1)} \cdot x_3)$$
$$h_2 = \sigma(w_{1,2}^{(1)} \cdot x_1 + w_{2,2}^{(1)} \cdot x_2 + w_{3,2}^{(1)} \cdot x_3).$$

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

Backpropagation

Answer c) (i) (4)

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

Backpropagation

Answer c) (i) (5)

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

$$w_{1,2}^{(2)} := w_{1,2}^{(2)} + \frac{1}{N}\sum_{e=1}^{N}(t_{e,2} - y_{e,2}) \cdot y_{e,2} \cdot (1 - y_{e,2}) \cdot h_1$$

$$w_{2,2}^{(1)} := w_{2,2}^{(1)} + \frac{1}{N}\sum_{e=1}^{N} \cdot h_2 \cdot (1 - h_2) \cdot x_2 \sum_{l=1}^{2} \delta_{e,l} \cdot w_{2,l}^{(2)}$$

- Now all that is left is computing the forward pass and then applying our update rules to get our new weight values.

Backpropagation

Answer c) (ii)

- We compute our forward pass with $w_{j,k}^{(i)} = 1$ 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 our updates:

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

Skip-Gram Basics

Context (1)

- Consider the sentence

$$\textit{The quick brown fox jumped over the lazy dog},$$
$$c_1 \quad c_2 \quad w \quad c_3 \quad c_4$$

  where w marks the *center word* and c_i the *context words*.
- Here, we have a *context window* of size 2, meaning we use the 2 previous and 2 subsequent words around w as context.
- In the lecture, skip-gram meant training a logistic regression model on the following binary classification task to learn word embeddings: $p(\text{True}|w, c) = \sigma(l_{w,c})$, where $l_{w,c}$ is the logit score (i.e. unnormalized prediction) produced by a logistic regression model given words w and c.
- Given this task, sentence and context window, some positive examples of the form (source word, target word) would be (*fox, quick*), (*fox, brown*), etc.

Skip-Gram Basics

Context (2)

- However, in their original work on the skip-gram approach, the authors learned word embeddings using a different task: predicting context word c_j given center word w_i in position i of a given sequence of words.
- We refer to this task as the *original task* throughout this exercise.

Skip-Gram Basics

Question a)

- How many classes are there in the original task proposed for skip-gram? And how could we compute probabilities for each class given logit scores from some classifier? Give a formal expression for this original task, including the way probabilities are computed from logits.

Skip-Gram Basics

Answer 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 $j \in \{i-2, i-1, i+1, i+2\}$.

Skip-Gram Basics

Question b)

- Generate all positive training examples for the original skip-gram task using the given sentence above as the entire training set. Use a window size of 2.

Skip-Gram Basics

Answer b) (1)

- 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.
  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 .
- Green marks center words, yellow marks context words
- The following table lists all examples generated for the original skip-gram task using a context window of size 2 and the given sentence.

Skip-Gram Basics

Answer b) (2)

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
⋮	⋮	⋮

Skip-Gram Basics

Question c)

- How is the skip-gram model parameterized? What is the role of each parameter?
- Assume your vocabulary consists of 101.425 words and you want to learn word representations of 300 dimensions each.
- What is the number of parameters in the skip-gram approach in this setting?

Skip-Gram Basics

Answer 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:

$$\boldsymbol{W} \in \mathbb{R}^{101425 \times 300}$$
$$\boldsymbol{W}' \in \mathbb{R}^{101425 \times 300}.$$

Skip-Gram Basics

Question d)

- Assume that word *Frodo* is the 79th word in your vocabulary. How do we obtain a single (final) representation of the word *Frodo* after our model has finished training? What decision do we need to make to obtain such a final representation?

Skip-Gram Basics

Answer 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.

Skip-Gram Training

Context

- We now focus on how to train models using the skip-gram approach.
- Assume throughout that we use dot products to compute logit scores, i.e. $l_{w_i,c_j} = \boldsymbol{w}_i^T \boldsymbol{w}'_j$, and that we use the softmax function to obtain probabilities from logit scores

Skip-Gram Training

Question a)

- To train a skip-gram model, we find the parameters θ, i.e. word embedings, that maximize the likelihood of our training data, i.e. we perform MLE.
- Given text corpus T, give the expression for the dataset likelihood $L(\theta|T)$ as determined by a skip-gram model with a context window of size L.
- Give the corresponding negative log-likelihood expression under the empirical risk minimization framework.
- As in the lecture, assume independence between words and training examples (i.i.d. assumption)

Skip-Gram Training

Answer 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 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.

Skip-Gram Training

Question b)

- Recall that computing the negative log-likelihood is equivalent to using cross-entropy loss.
- Consider the following toy vocabulary V=$\{$*Daenerys*, *Frodo*, *ring*, *dragon*, *Picard*, *ship*$\}$.
- Assume you are training a skip-gram model over a large corpus T with this vocabulary, and that the current values of the parameters of your model are the following:

$$\mathbf{W} = \begin{bmatrix} 0.54 & 0.25 & 0.07 \\ 0.91 & 0.74 & 0.27 \\ 0.58 & 0.00 & 0.05 \\ 0.16 & 0.77 & 0.31 \\ 0.10 & 0.54 & 0.32 \\ 0.59 & 0.99 & 0.06 \end{bmatrix}, \mathbf{W}' = \begin{bmatrix} 0.19 & 0.64 & 0.98 \\ 0.17 & 0.42 & 0.47 \\ 0.55 & 0.24 & 0.92 \\ 0.89 & 0.58 & 0.13 \\ 0.31 & 0.73 & 0.48 \\ 0.49 & 0.20 & 0.54 \end{bmatrix}$$

- Compute the value of the cross-entropy loss corresponding to the single training instance (*Frodo* (source word), *ring* (target word)) when used as positive example.

Skip-Gram Training

Answer b) (1)

- 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 $\boldsymbol{W}'$.

Skip-Gram Training

Answer b) (2)

- 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:

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

Skip-Gram Training

Answer b) (3)

$$L_{CE}(ring|Frodo) = -\boldsymbol{w}_{ring}^{T}\boldsymbol{w}_{Frodo} + \ln \sum_{v \in V'} \exp(\boldsymbol{w}_{v}^{T}\boldsymbol{w}_{Frodo})$$

- 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.

Skip-Gram Training

Question c)

- Assume we train using stochastic gradient-descent, i.e. we update our relevant parameters during training after computing the loss for a single positive example as done in (b).
- Which parameters are updated after every single example? Why?

Skip-Gram Training

Answer 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.

Word Embeddings as Matrix Factorization

Context

- As discussed in Task 1, we obtain the final representation of a given word from its corresponding representations in $\boldsymbol{W}$ and $\boldsymbol{W}'$.
- Let us now consider the perspective that $\boldsymbol{W}$ and $\boldsymbol{W}'$ are factors of another matrix $\boldsymbol{M}$, i.e. $\boldsymbol{M} = \boldsymbol{W}\boldsymbol{W}'^{T}$.
- If we can derive some meaning for $\boldsymbol{M}$, we may be able to interpret what we are learning to factorize with the skip-gram approach.
- As in the lecture, we now take the perspective of skip-gram as the binary classification task $p(\text{True}|w, c) = \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c)$, where w is a source word, c a target (context) word, and $\boldsymbol{w}_w, \boldsymbol{w}_c$ their corresponding word representations.
- Note that in this setting, entry m_{ij} in $\boldsymbol{M}$ is given by $\boldsymbol{w}_i^T \boldsymbol{w}_j$.

Word Embeddings as Matrix Factorization

Question a)

- The (lecture) skip-gram training objective for *a single training example* (w, c) using the cross-entropy loss is the following:

$$L(w, c) = \log \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c) + \sum_{i=1}^{k} \log(-\boldsymbol{w}_w^t \boldsymbol{w}_i), \qquad (1)$$

  where k are the number of negative examples sampled per positive tuple (w, c) in the training set (we omit the negation of the loss for brevity).
- Rewrite Equation 1 so that it explicitly shows the *sampling* of negatives.
- To this end, use the expectation operator and assume negative examples $\boldsymbol{w}_i$ are drawn from the uniform distribution P over vocabulary V.
- That is, the probability of using word c to construct a negative is $\frac{\#(c)}{|V|}$, where $\#(c)$ is the number of times word c appears in the training corpus.

Word Embeddings as Matrix Factorization

Answer a)

- We can explicitly express the sampling of negatives in the objective as follows:

$$L(w, c) = \log \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c) + k \cdot \mathbb{E}_{i \sim P}[\log(-\boldsymbol{w}_w^t \boldsymbol{w}_i)]. \quad (2)$$

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

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

Word Embeddings as Matrix Factorization

Question b)

- Generalize your training objective derived in the previous question to include all training examples (w, c), i.e. the overall training objective L using the entire corpus.

Word Embeddings as Matrix Factorization

Answer b)

- We can do this by applying Equation 3 over the entire training set as follows:

$$L = \sum_{w \in V} \sum_{c \in V} \#(w, c) \left(\log \sigma(\boldsymbol{w}_w^t \boldsymbol{w}_c) + k \cdot \sum_{i \in V} \frac{\#(i)}{|V|} \log(-\boldsymbol{w}_w^t \boldsymbol{w}_i) \right), \tag{4}$$

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

Word Embeddings as Matrix Factorization

Question c)

- During training, we optimize the objective L.
- To see what this optimization process tells us about m_{wc}, show that the optimal value for $\boldsymbol{w}_w^t \boldsymbol{w}_c$ is given by the following expression:

$$\boldsymbol{w}_w^t \boldsymbol{w}_c = \ln \left(\frac{\#(w,c)|V|}{\#(w)\#(c)} \right) - \ln k \tag{5}$$

- To get there, derive an optimal value for $\boldsymbol{w}_w^t \boldsymbol{w}_c$ by computing the gradient of L w.r.t *a single training example* (w, c), setting it to zero, and deriving a value for $\boldsymbol{w}_w^t \boldsymbol{w}_c$.
  Hint: only two terms in L are relevant for this gradient computation.

Word Embeddings as Matrix Factorization

Answer c) (1)

- 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:

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

Word Embeddings as Matrix Factorization

Answer c) (2)

- 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:

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

- We got our answer, now let's interpret this expression.

Word Embeddings as Matrix Factorization

Answer c) (3)

- 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:

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

Word Embeddings as Matrix Factorization

Answer c) (4)

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

- 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:

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

Word Embeddings as Matrix Factorization

Answer c) (5)

- 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 4 is the PMI matrix of our training corpus.
- We can thus relate skip-gram to count-based sparse word representations, which allows 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 (seen in basic Text Analytics course).
- Finally, note that if we set $k > 1$ as commonly done, we get a PMI matrix shifted by $\log k$.