CS 4063 – Natural Language Processing

Lecture Notes – week 5

Muhammad Hannan Farooq

Word2Vec

- **Word2Vec** is a popular neural network-based model introduced by researchers at Google in 2013 that transforms words into high-dimensional vectors of real numbers. These vectors, known as **word embeddings**, capture semantic relationships between words in a continuous vector space. The core idea of Word2Vec is to place semantically similar words close together in the vector space, which helps in many natural language processing (NLP) tasks.
- Word2Vec can be trained using two different approaches:
  - Continuous Bag of Words (CBOW): Predicts a target word based on its context (i.e., surrounding words).
  - Skip-Gram: Predicts the surrounding context words based on a target word.

Skip-gram

- The skip-gram model is a technique used in Natural Language Processing (NLP) for learning distributed representations of words (i.e., word embeddings).
- It is part of the Word2Vec algorithm, which is widely used to create word vectors that capture semantic and syntactic meanings of words based on their contexts in a corpus.

Skip-gram Model

- The primary goal of the skip-gram model is to predict the context words given a target word.
- In simpler terms, given a word in a sentence, it tries to predict which words are likely to surround it within a predefined "window" size.
- This window defines how many context words on either side of the target word will be considered for prediction.

How Skip-gram Works

- **Input Word (Target Word):** A single word from the corpus is chosen.
- **Output (Context Words):** The words surrounding the input word (within a certain window size) are the context words that need to be predicted.

Skip-gram steps

- The model takes a target word as input and predicts the context words that are likely to appear in a specified window size around the target word.
- For example, if the window size is 2, the model will predict the two words before and after the target word.

The skip-gram model learns the relationships between words by maximizing the probability of observing context words based on the target word. During training, the model tries to adjust the word embeddings so that words that frequently occur in similar contexts have similar vectors in the embedding space.

Mathematical Formulation

- Given a word wt (target word) and a context window size of c, the model aims to maximize the following objective function:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log P(w_{t+j}|w_t)$$

Where:

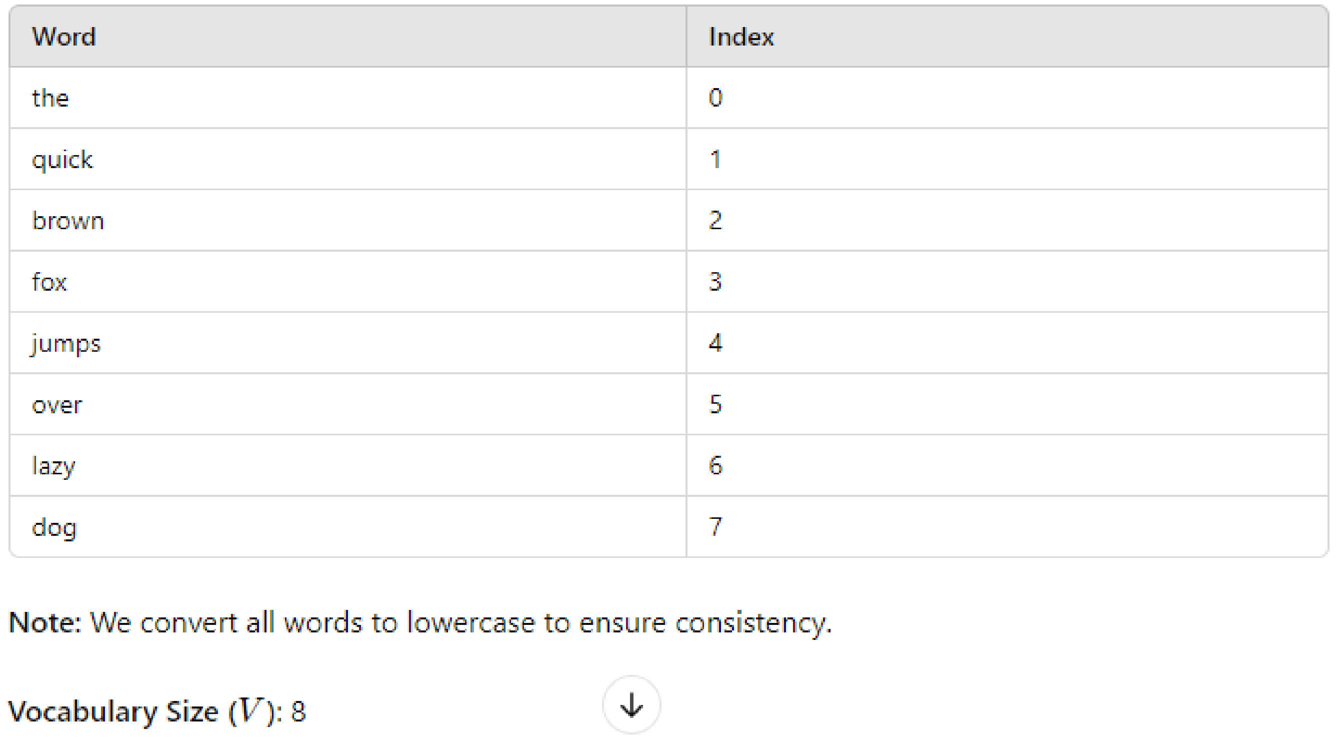
- w_t is the target word at position t.
- w_{t+j} represents the context words within a window size c.
- T is the total number of words in the corpus.
- $P(w_{t+j}|w_t)$ is the conditional probability of a context word given the target word.

Example

- “The quick brown fox jumps over the lazy dog.”
- If the target word is “brown”, and the window size is 2, the context words are “quick” and “fox” (words within two positions on either side of the target word). The skip-gram model tries to predict these context words given the target word.

Step 1 : Preprocessing

- **Corpus:** “The quick brown fox jumps over the lazy dog.”
- **Vocabulary Construction:** Identify the unique words in the corpus. Let's assume each word in the corpus is assigned a unique index.
- **Context Window:** Let’s assume we choose a context window size c=2. This means for every target word, we will look at the 2 words before and the 2 words after the target word.



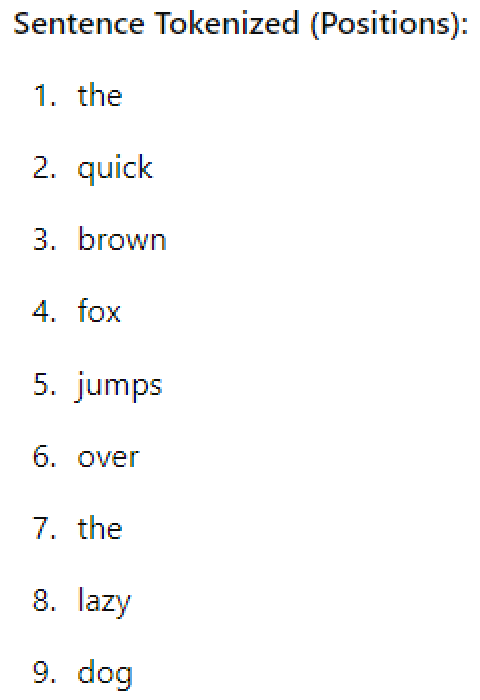
Word	Index
the	0
quick	1
brown	2
fox	3
jumps	4
over	5
lazy	6
dog	7

Note: We convert all words to lowercase to ensure consistency.

Vocabulary Size (V): 8

Step 2: Generating Training Data

- The skip-gram model generates training data by creating word pairs consisting of the target word and its context words.
- We choose a context window size (cc) of 2. This means we'll consider the two words before and after each target word to generate word pairs.
- For the sentence we generate Target-Context Pairs



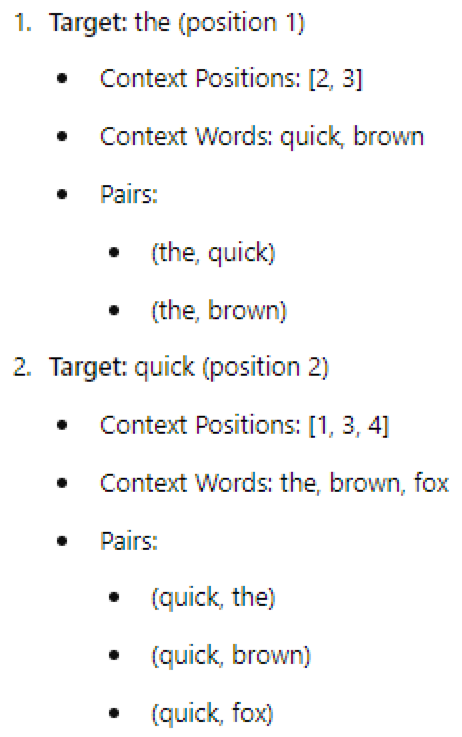
Sentence Tokenized (Positions):

1. the
2. quick
3. brown
4. fox
5. jumps
6. over
7. the
8. lazy
9. dog

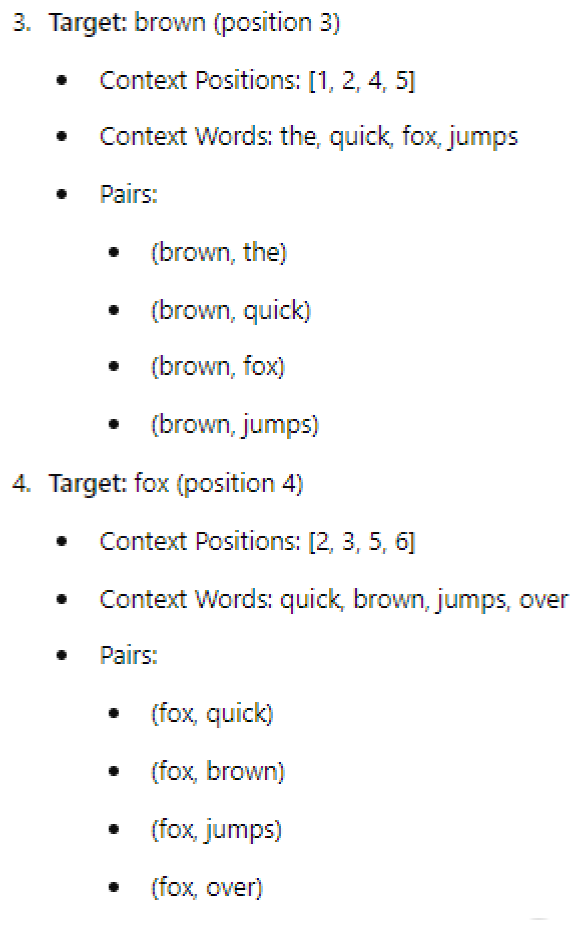
Step 2: Generating Training Data

Total Training Pairs: 30 pairs

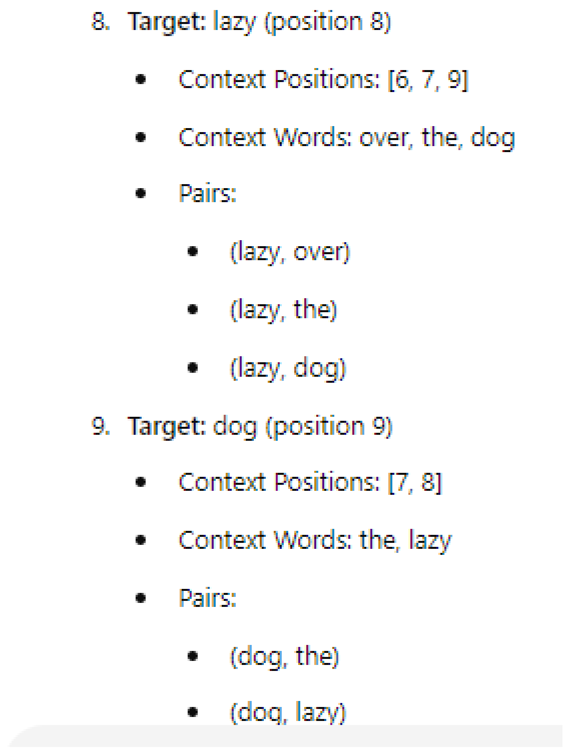
- Generating Pairs:



1. **Target:** the (position 1)
   - Context Positions: [2, 3]
   - Context Words: quick, brown
   - Pairs:
     - (the, quick)
     - (the, brown)
2. **Target:** quick (position 2)
   - Context Positions: [1, 3, 4]
   - Context Words: the, brown, fox
   - Pairs:
     - (quick, the)
     - (quick, brown)
     - (quick, fox)



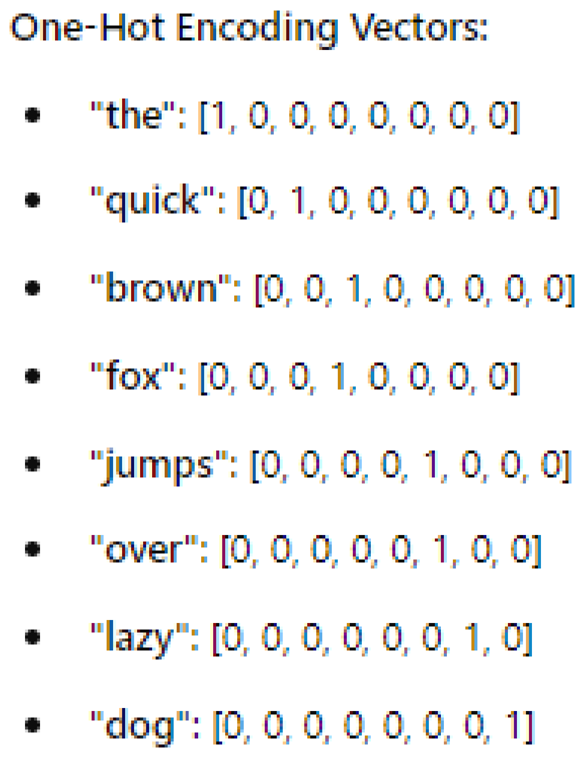
3. **Target:** brown (position 3)
   - Context Positions: [1, 2, 4, 5]
   - Context Words: the, quick, fox, jumps
   - Pairs:
     - (brown, the)
     - (brown, quick)
     - (brown, fox)
     - (brown, jumps)
4. **Target:** fox (position 4)
   - Context Positions: [2, 3, 5, 6]
   - Context Words: quick, brown, jumps, over
   - Pairs:
     - (fox, quick)
     - (fox, brown)
     - (fox, jumps)
     - (fox, over)



8. **Target:** lazy (position 8)
   - Context Positions: [6, 7, 9]
   - Context Words: over, the, dog
   - Pairs:
     - (lazy, over)
     - (lazy, the)
     - (lazy, dog)
9. **Target:** dog (position 9)
   - Context Positions: [7, 8]
   - Context Words: the, lazy
   - Pairs:
     - (dog, the)
     - (dog, lazy)

Step 2: One-Hot Encoding

- Each word in the vocabulary is represented as a one-hot vector of size V=8



One-Hot Encoding Vectors:

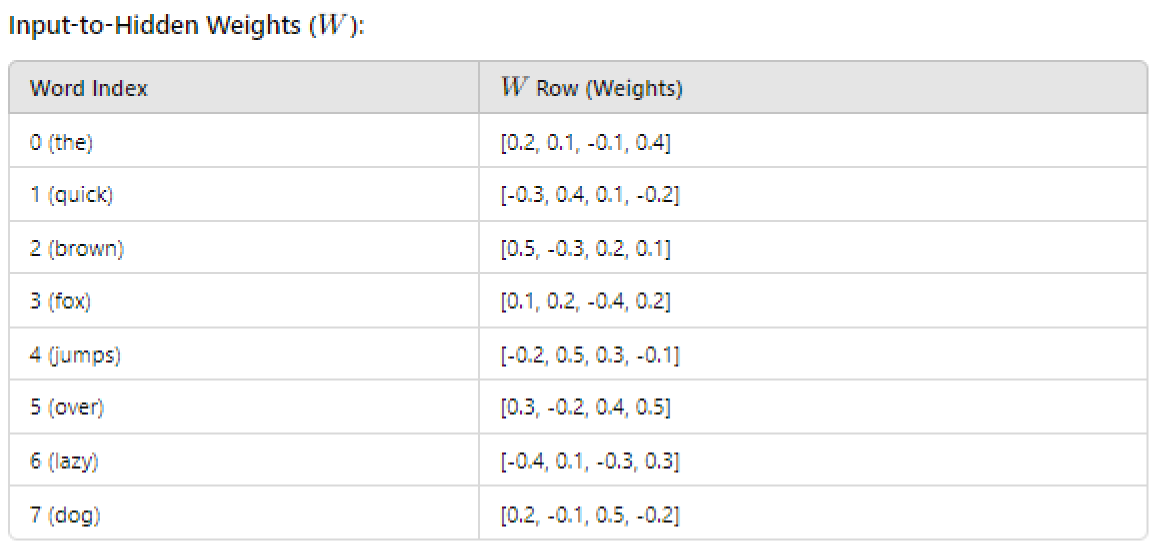
- "the": [1, 0, 0, 0, 0, 0, 0, 0]
- "quick": [0, 1, 0, 0, 0, 0, 0, 0]
- "brown": [0, 0, 1, 0, 0, 0, 0, 0]
- "fox": [0, 0, 0, 1, 0, 0, 0, 0]
- "jumps": [0, 0, 0, 0, 1, 0, 0, 0]
- "over": [0, 0, 0, 0, 0, 1, 0, 0]
- "lazy": [0, 0, 0, 0, 0, 0, 1, 0]
- "dog": [0, 0, 0, 0, 0, 0, 0, 1]

Step 3: Model Architecture

- The skip-gram model uses a simple neural network with:
  - Input Layer: Size V=8 (vocabulary size)
  - Hidden Layer: Size N=4 (embedding dimension)
  - Output Layer: Size V=8 (vocabulary size)
  - Weight Matrices:
    - Input-to-Hidden Weights (W): Matrix of size $V\times N$ (8 x 4)
    - Hidden-to-Output Weights (W'): Matrix of size $N\times V$ (4 x 8)

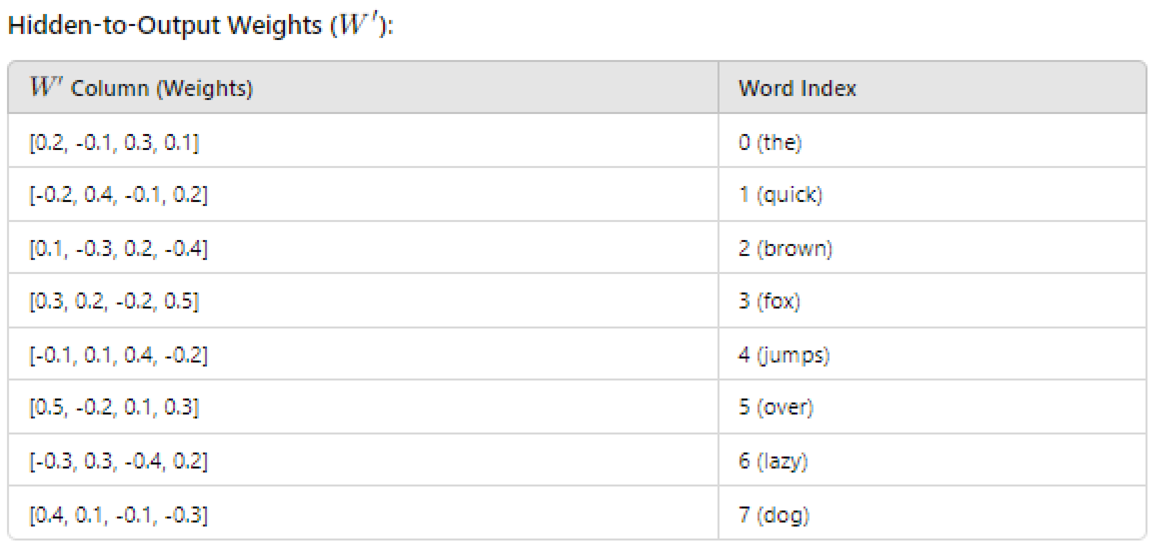
Step 4: Initializing Weights

- We initialize weights with small random numbers for demonstration purposes.



Input-to-Hidden Weights (W):

Word Index	W Row (Weights)
0 (the)	[0.2, 0.1, -0.1, 0.4]
1 (quick)	[-0.3, 0.4, 0.1, -0.2]
2 (brown)	[0.5, -0.3, 0.2, 0.1]
3 (fox)	[0.1, 0.2, -0.4, 0.2]
4 (jumps)	[-0.2, 0.5, 0.3, -0.1]
5 (over)	[0.3, -0.2, 0.4, 0.5]
6 (lazy)	[-0.4, 0.1, -0.3, 0.3]
7 (dog)	[0.2, -0.1, 0.5, -0.2]

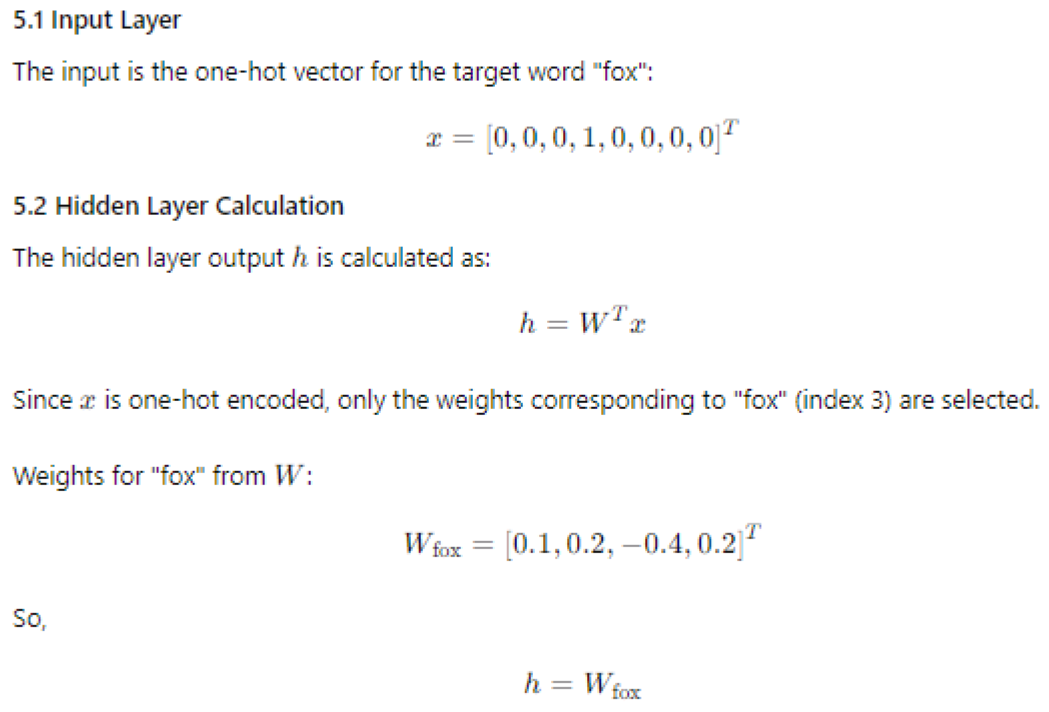


Hidden-to-Output Weights (W'):

W' Column (Weights)	Word Index
[0.2, -0.1, 0.3, 0.1]	0 (the)
[-0.2, 0.4, -0.1, 0.2]	1 (quick)
[0.1, -0.3, 0.2, -0.4]	2 (brown)
[0.3, 0.2, -0.2, 0.5]	3 (fox)
[-0.1, 0.1, 0.4, -0.2]	4 (jumps)
[0.5, -0.2, 0.1, 0.3]	5 (over)
[-0.3, 0.3, -0.4, 0.2]	6 (lazy)
[0.4, 0.1, -0.1, -0.3]	7 (dog)

Step 5: Forward Pass for a Training Example

- We'll perform the forward pass for one training pair:
  - Training Pair: (Target word: "fox", Context word: "jumps")
  - Indices:
    - Target word "fox": index 3
    - Context word "jumps": index 4



5.1 Input Layer

The input is the one-hot vector for the target word "fox":

$$x = [0, 0, 0, 1, 0, 0, 0, 0]^T$$

5.2 Hidden Layer Calculation

The hidden layer output h is calculated as:

$$h = W^T x$$

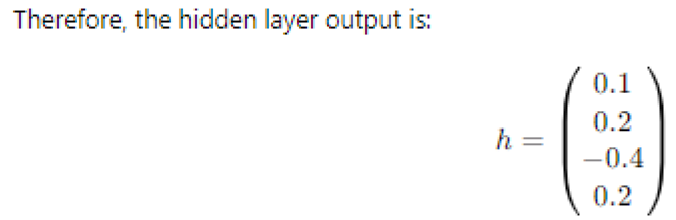
Since x is one-hot encoded, only the weights corresponding to "fox" (index 3) are selected.

Weights for "fox" from W:

$$W_{\text{fox}} = [0.1, 0.2, -0.4, 0.2]^T$$

So,

$$h = W_{\text{fox}}$$

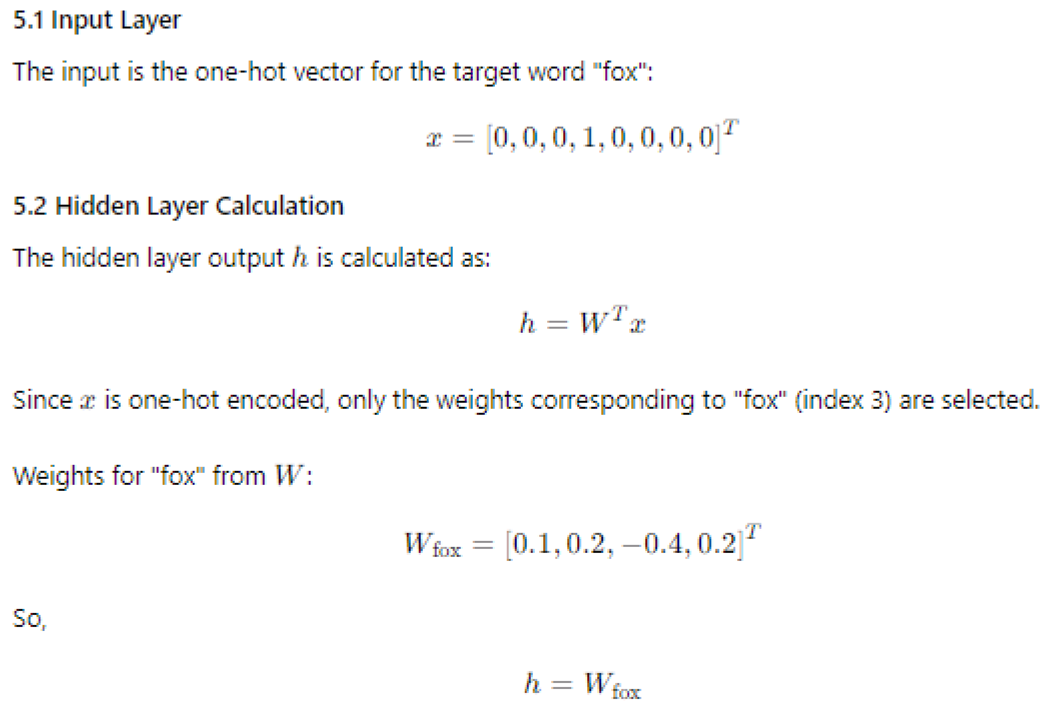


Therefore, the hidden layer output is:

$$h = \begin{pmatrix} 0.1 \\ 0.2 \\ -0.4 \\ 0.2 \end{pmatrix}$$

Step 5: Forward Pass for a Training Example

- We'll perform the forward pass for one training pair:
  - Training Pair: (Target word: "fox", Context word: "jumps")
  - Indices:
    - Target word "fox": index 3
    - Context word "jumps": index 4



5.1 Input Layer

The input is the one-hot vector for the target word "fox":

$$x = [0, 0, 0, 1, 0, 0, 0, 0]^T$$

5.2 Hidden Layer Calculation

The hidden layer output h is calculated as:

$$h = W^T x$$

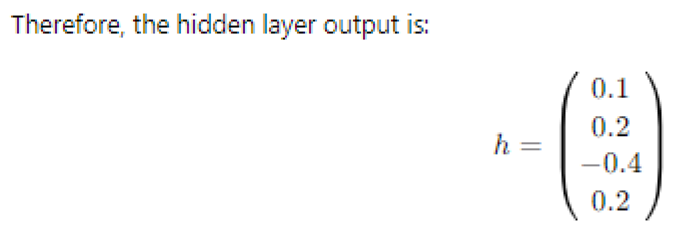
Since x is one-hot encoded, only the weights corresponding to "fox" (index 3) are selected.

Weights for "fox" from W:

$$W_{\text{fox}} = [0.1, 0.2, -0.4, 0.2]^T$$

So,

$$h = W_{\text{fox}}$$



Therefore, the hidden layer output is:

$$h = \begin{pmatrix} 0.1 \\ 0.2 \\ -0.4 \\ 0.2 \end{pmatrix}$$

Step 5: Forward Pass for a Training Example

5.3 Output Layer Calculation

The output layer computes the scores u for each word in the vocabulary:

$$u = W'h$$

1. u_0 ("the"):

$$u_0 = h^T W'_{:,0} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.2, -0.1, 0.3, 0.1]$$

$$u_0 = (0.1)(0.2) + (0.2)(-0.1) + (-0.4)(0.3) + (0.2)(0.1) = 0.02 - 0.02 - 0.12 + 0.02 = -0.10$$

2. u_1 ("quick"):

$$u_1 = h^T W'_{:,1} = [0.1, 0.2, -0.4, 0.2]^T \cdot [-0.2, 0.4, -0.1, 0.2]$$

$$u_1 = (0.1)(-0.2) + (0.2)(0.4) + (-0.4)(-0.1) + (0.2)(0.2) = -0.02 + 0.08 + 0.04 + 0.04 = 0.14$$

3. u_2 ("brown"):

$$u_2 = h^T W'_{:,2} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.1, -0.3, 0.2, -0.4]$$

$$u_2 = (0.1)(0.1) + (0.2)(-0.3) + (-0.4)(0.2) + (0.2)(-0.4) = 0.01 - 0.06 - 0.08 - 0.08 = -0.21$$

Step 5: Forward Pass for a Training Example

4. u_3 ("fox"):

$$u_3 = h^T W'_{:,3} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.3, 0.2, -0.2, 0.5]$$

$$u_3 = (0.1)(0.3) + (0.2)(0.2) + (-0.4)(-0.2) + (0.2)(0.5) = 0.03 + 0.04 + 0.08 + 0.10 = 0.25$$

5. u_4 ("jumps"):

$$u_4 = h^T W'_{:,4} = [0.1, 0.2, -0.4, 0.2]^T \cdot [-0.1, 0.1, 0.4, -0.2]$$

$$u_4 = (0.1)(-0.1) + (0.2)(0.1) + (-0.4)(0.4) + (0.2)(-0.2) = -0.01 + 0.02 - 0.16 - 0.04 = -0.19$$

6. u_5 ("over"):

$$u_5 = h^T W'_{:,5} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.5, -0.2, 0.1, 0.3]$$

$$u_5 = (0.1)(0.5) + (0.2)(-0.2) + (-0.4)(0.1) + (0.2)(0.3) = 0.05 - 0.04 - 0.04 + 0.06 = 0.03$$

7. u_6 ("lazy"):

$$u_6 = h^T W'_{:,6} = [0.1, 0.2, -0.4, 0.2]^T \cdot [-0.3, 0.3, -0.4, 0.2]$$

$$u_6 = (0.1)(-0.3) + (0.2)(0.3) + (-0.4)(-0.4) + (0.2)(0.2) = -0.03 + 0.06 + 0.16 + 0.04 = 0.13$$

8. u_7 ("dog"):

$$u_7 = h^T W'_{:,7} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.4, 0.1, -0.1, -0.3]$$

$$u_7 = (0.1)(0.4) + (0.2)(0.1) + (-0.4)(-0.1) + (0.2)(-0.3) = 0.04 + 0.02 + 0.04 - 0.06 = 0.04$$

Step 5: Forward Pass for a Training Example

4. u_3 ("fox"):

$$u_3 = h^T W'_{:,3} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.3, 0.2, -0.2, 0.5]$$

$$u_3 = (0.1)(0.3) + (0.2)(0.2) + (-0.4)(-0.2) + (0.2)(0.5) = 0.03 + 0.04 + 0.08 + 0.10 = 0.25$$

5. u_4 ("jumps"):

$$u_4 = h^T W'_{:,4} = [0.1, 0.2, -0.4, 0.2]^T \cdot [-0.1, 0.1, 0.4, -0.2]$$

$$u_4 = (0.1)(-0.1) + (0.2)(0.1) + (-0.4)(0.4) + (0.2)(-0.2) = -0.01 + 0.02 - 0.16 - 0.04 = -0.19$$

6. u_5 ("over"):

$$u_5 = h^T W'_{:,5} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.5, -0.2, 0.1, 0.3]$$

$$u_5 = (0.1)(0.5) + (0.2)(-0.2) + (-0.4)(0.1) + (0.2)(0.3) = 0.05 - 0.04 - 0.04 + 0.06 = 0.03$$

7. u_6 ("lazy"):

$$u_6 = h^T W'_{:,6} = [0.1, 0.2, -0.4, 0.2]^T \cdot [-0.3, 0.3, -0.4, 0.2]$$

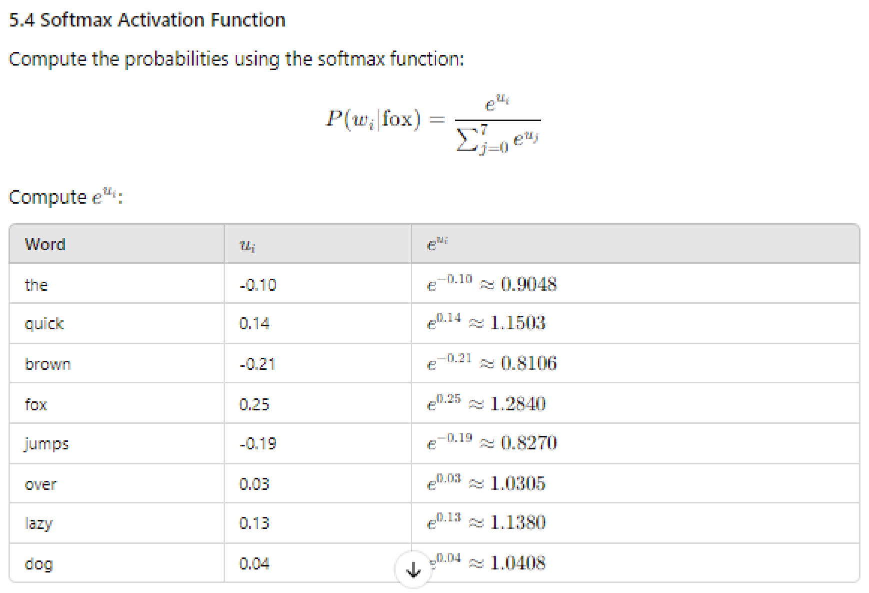
$$u_6 = (0.1)(-0.3) + (0.2)(0.3) + (-0.4)(-0.4) + (0.2)(0.2) = -0.03 + 0.06 + 0.16 + 0.04 = 0.13$$

8. u_7 ("dog"):

$$u_7 = h^T W'_{:,7} = [0.1, 0.2, -0.4, 0.2]^T \cdot [0.4, 0.1, -0.1, -0.3]$$

$$u_7 = (0.1)(0.4) + (0.2)(0.1) + (-0.4)(-0.1) + (0.2)(-0.3) = 0.04 + 0.02 + 0.04 - 0.06 = 0.04$$

Step 5: Forward Pass for a Training Example



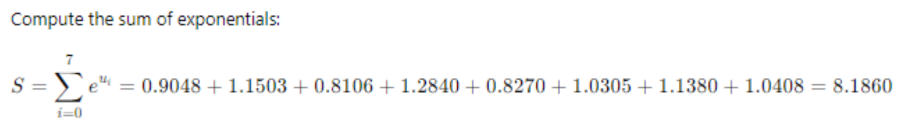
5.4 Softmax Activation Function

Compute the probabilities using the softmax function:

$$P(w_i|\text{fox}) = \frac{e^{u_i}}{\sum_{j=0}^{7} e^{u_j}}$$

Compute e^{u_i}:

Word	u_i	e^{u_i}
the	-0.10	$e^{-0.10} \approx 0.9048$
quick	0.14	$e^{0.14} \approx 1.1503$
brown	-0.21	$e^{-0.21} \approx 0.8106$
fox	0.25	$e^{0.25} \approx 1.2840$
jumps	-0.19	$e^{-0.19} \approx 0.8270$
over	0.03	$e^{0.03} \approx 1.0305$
lazy	0.13	$e^{0.13} \approx 1.1380$
dog	0.04	$e^{0.04} \approx 1.0408$



Compute the sum of exponentials:

$$S = \sum_{i=0}^{7} e^{u_i} = 0.9048 + 1.1503 + 0.8106 + 1.2840 + 0.8270 + 1.0305 + 1.1380 + 1.0408 = 8.1860$$

Step 5: Forward Pass for a Training Example

Compute the probabilities:

- P(the):

$$P(\text{the}|\text{fox}) = \frac{0.9048}{8.1860} \approx 0.1105$$

- P(quick):

$$P(\text{quick}|\text{fox}) = \frac{1.1503}{8.1860} \approx 0.1406$$

- P(brown):

$$P(\text{brown}|\text{fox}) = \frac{0.8106}{8.1860} \approx 0.0990$$

- P(fox):

$$P(\text{fox}|\text{fox}) = \frac{1.2840}{8.1860} \approx 0.1569$$

- P(jumps):

$$P(\text{jumps}|\text{fox}) = \frac{0.8270}{8.1860} \approx 0.1010$$

- P(over):

$$P(\text{over}|\text{fox}) = \frac{1.0305}{8.1860} \approx 0.1259$$

- P(lazy):

$$P(\text{lazy}|\text{fox}) = \frac{1.1380}{8.1860} \approx 0.1390$$

- P(dog):

$$P(\text{dog}|\text{fox}) = \frac{1.0408}{8.1860} \approx 0.1271$$

5.5 Interpreting the Output

The model predicts the probability of each word in the vocabulary being the context word for the target word "fox".

Our actual context word is "**jumps**", which has a probability of approximately **0.1010**.

Step 6: Calculating the Loss

We use the cross-entropy loss function for this multi-class classification problem.

For the actual context word "jumps" (index 4), the loss L is:

$$L = -\log P(\text{jumps}|\text{fox})$$

$$L = -\log(0.1010) \approx 2.292$$

Step 7: Backpropagation and Weight Updates

7.1 Gradient of the Loss with Respect to u

Compute the gradient δ for the output layer:

$$\delta_i = P(w_i|\text{fox}) - y_{\text{true},i}$$

- y_{true}: One-hot vector for the context word "jumps" (index 4)

$$y_{\text{true}} = [0, 0, 0, 0, 1, 0, 0, 0]^T$$

Compute δ_i:

i	δ_i
0	$0.1105 - 0 = 0.1105$
1	$0.1406 - 0 = 0.1406$
2	$0.0990 - 0 = 0.0990$
3	$0.1569 - 0 = 0.1569$
4	$0.1010 - 1 = -0.8990$
5	$0.1259 - 0 = 0.1259$
6	$0.1390 - 0 = 0.1390$
7	$0.1271 - 0 = 0.1271$

Step 7: Backpropagation and Weight Updates

7.2 Updating Hidden-to-Output Weights (W')

Compute the gradient with respect to W':

$$\frac{\partial L}{\partial W'_{:,i}} = h \cdot \delta_i$$

For each word i, update $W'_{:,i}$:

Example for $i = 4$ ("jumps"):

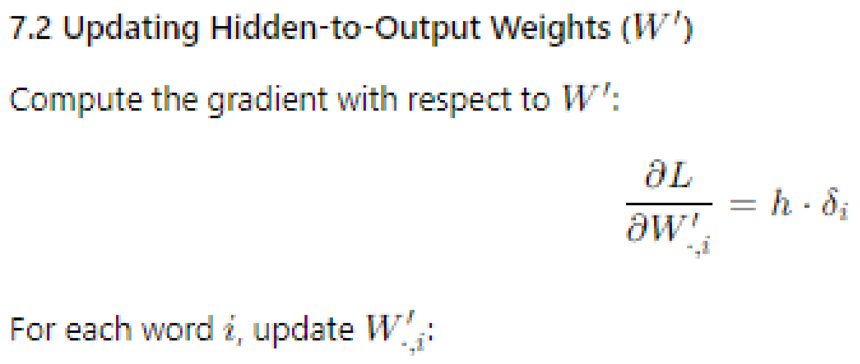
$$\delta_4 = -0.8990$$

Compute the gradient:

$$\frac{\partial L}{\partial W'_{:,4}} = h \cdot (-0.8990) = [0.1, 0.2, -0.4, 0.2]^T \cdot (-0.8990)$$

$$\frac{\partial L}{\partial W'_{:,4}} = [-0.0899, -0.1798, 0.3596, -0.1798]^T$$

Step 7: Backpropagation and Weight Updates

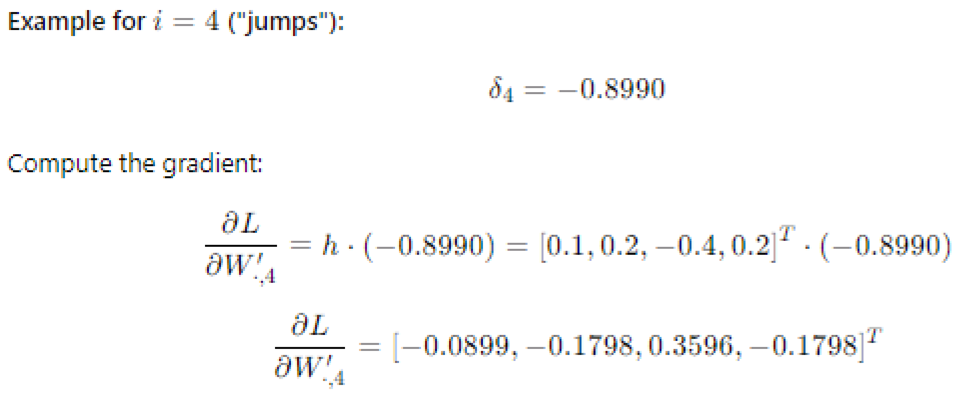


7.2 Updating Hidden-to-Output Weights (W')

Compute the gradient with respect to W':

$$\frac{\partial L}{\partial W'_{\cdot,i}} = h \cdot \delta_i$$

For each word i, update $W'_{\cdot,i}$:



Example for $i = 4$ ("jumps"):

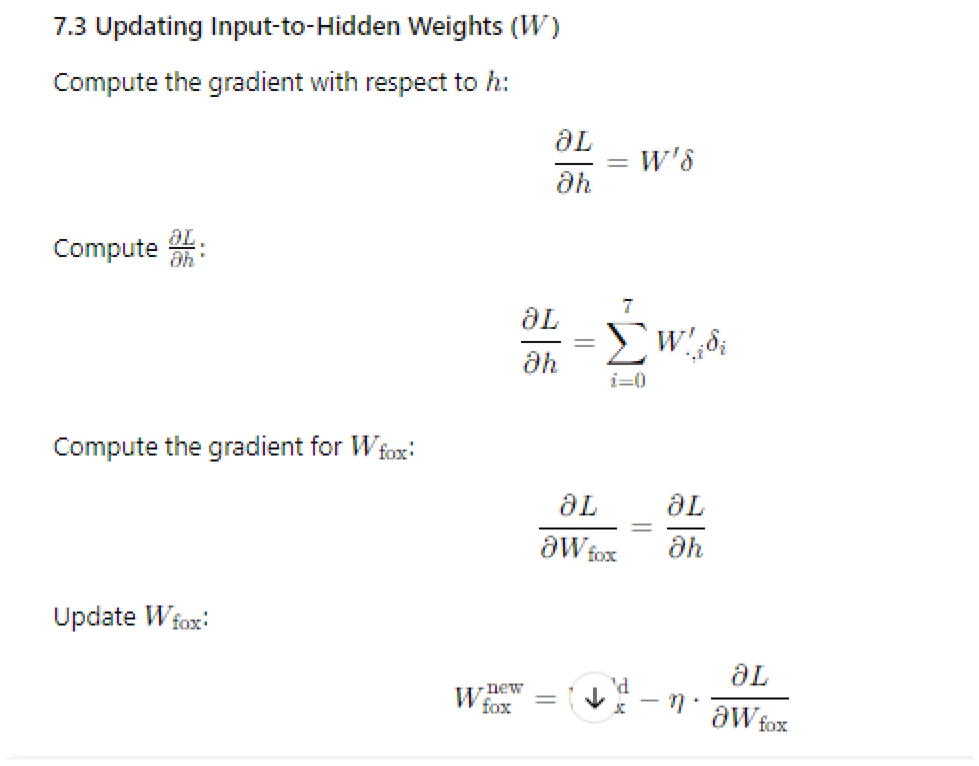
$$\delta_4 = -0.8990$$

Compute the gradient:

$$\frac{\partial L}{\partial W'_{:,4}} = h \cdot (-0.8990) = [0.1, 0.2, -0.4, 0.2]^T \cdot (-0.8990)$$

$$\frac{\partial L}{\partial W'_{:,4}} = [-0.0899, -0.1798, 0.3596, -0.1798]^T$$

Step 7: Backpropagation and Weight Updates



7.3 Updating Input-to-Hidden Weights (W)

Compute the gradient with respect to h:

$$\frac{\partial L}{\partial h} = W'\delta$$

Compute $\frac{\partial L}{\partial h}$:

$$\frac{\partial L}{\partial h} = \sum_{i=0}^{7} W'_{:,i}\delta_i$$

Compute the gradient for W_{fox}:

$$\frac{\partial L}{\partial W_{\text{fox}}} = \frac{\partial L}{\partial h}$$

Update W_{fox}:

$$W_{\text{fox}}^{\text{new}} = \text{[illegible]} - \eta \cdot \frac{\partial L}{\partial W_{\text{fox}}}$$

Step 8: Repeat for All Training Pairs

We repeat the forward pass, loss calculation, and backpropagation steps for each of the 30 training pairs generated in Step 1.2.

Step 9: Conclusion

After multiple epochs of training over all the pairs, the weights W (input-to-hidden) will converge to values that represent meaningful word embeddings.

- The rows of W correspond to the word embeddings for each word in the vocabulary.
- Words that appear in similar contexts will have similar embeddings.

Final Notes

- **Negative Sampling:** In practice, to make training computationally feasible, especially with large vocabularies, techniques like negative sampling are used.
- **Batch Processing:** Instead of updating weights after each pair (stochastic gradient descent), we might use mini-batches or full batches for updates.
- **Multiple Epochs:** Training typically involves multiple passes over the data to ensure convergence.
- **Evaluation:** The quality of the embeddings can be evaluated using intrinsic methods (e.g., word similarity tasks) or extrinsic methods (e.g., performance on downstream NLP tasks).