Overview of the Skip-gram Model

- **Objective**: The Skip-gram model predicts context words given a target (input) word. This is the opposite of the Continuous Bag of Words (CBOW) model, which predicts a target word from its context.

Key Components

1. **Input Word**:
    - Represents a single word for which we want to predict surrounding (context) words.
    - For example, given the input word "sat," we want to find related words in the context, such as "he," "chair," etc.
2. **Context Words**:
    - These are the words surrounding the input word. In a typical implementation, context words can be found on both sides of the input word.
    - For example, in the sentence "he sat on the chair," if "sat" is the input word, context words could be "he," "on," and "the."
3. **Word Vectors**:
    - Each word is represented as a vector in a high-dimensional space, using a weight matrix W_{word} for input words and another matrix W_{context} for context words.
    - The dimension of these vectors is typically denoted as k.
4. **Hidden Layer**:
    - The hidden layer representation is calculated as $h = W_{\text{word}} \cdot x$, where x is the input word vector.

Loss Function

- The loss function for the Skip-gram model involves calculating the probability of context words given the input word.
- The formula is often represented as: $$ L(\theta) = -\sum_{i=1}^{d} \log(y_{w_i}) $$ where y_{w_i} is the predicted probability of context word w_i, and d is the number of context words.

Predictions

- For a single input word, the model will produce multiple context words.
- The output layer typically uses the softmax function to convert raw scores (logits) into probabilities.

Challenges

- **Computational Expense**: The softmax function can be computationally expensive, especially with a large vocabulary size.
- **Solutions**:
    1. **Negative Sampling**: Instead of updating all context words, randomly sample a few negative words to update the model, thus reducing computation.

2. **Contrastive Estimation**: This method helps improve the efficiency of the training process by contrasting positive examples (actual context words) against negative ones.
3. **Hierarchical Softmax**: This method uses a tree structure to reduce the computational complexity of calculating softmax, allowing for faster training times.

Summary

- The Skip-gram model focuses on predicting context words from a given input word, making it a powerful tool for learning word embeddings in NLP. It leverages a hidden layer to represent the input word and uses a loss function based on cross-entropy to train the model efficiently, despite challenges related to computational cost, which can be addressed through various techniques.

Overview of Word-Context Pairs

In natural language processing (NLP), word-context pairs are used to train models like the Skip-gram model. The idea is to learn the relationships between words and their surrounding contexts.

Definitions

1. **Correct Word-Context Pairs (D_{wc})**:
    - These are pairs of words where the context word has appeared with the target word in the training corpus.
    - Example: If we have the sentence "He sat on the chair," the correct pairs can be derived as:
        - (sat, on)
        - (sat, a)
        - $(sat, chair)$
        - (on, a)
        - $(on, chair)$
        - $(a, chair)$
        - (on, sat)
        - $(chair, sat)$
        - $(chair, on)$
        - $(chair, a)$

    Thus, the set of correct pairs can be denoted as: $$ D_{wc} = {(sat, on), (sat, a), (sat, chair), (on, a), (on, chair), (a, chair), (on, sat), (chair, sat), (chair, on), (chair, a)} $$

2. **Incorrect Word-Context Pairs (D_{wr})**:
    - These are pairs of words where the context word has *not* appeared with the target word in the corpus.
    - To create these pairs, we randomly sample words that have never co-occurred with the target word. For example:
        - $(sat, oxygen)$
        - $(sat, magic)$
        - $(chair, sad)$
        - $(chair, walking)$

The incorrect pairs can be denoted as: $$ D_{wr} = {(sat, oxygen), (sat, magic), (chair, sad), (chair, walking)} $$

Construction of Sets

- **Set of Correct Pairs (D_{wc})**: This is constructed from the training corpus by identifying all instances where words co-occur within a specified context window.
- **Set of Incorrect Pairs (D_{wr})**: This is constructed by randomly selecting a context word that has never been observed with the target word w. This sampling helps create negative examples for training, which are essential for models like Skip-gram to learn useful representations.

Representation Vectors

- **Word Representation (v_w)**: Each word w is represented as a vector in a continuous vector space. This representation is learned during the training process.
- **Context Word Representation (u_c)**: Similarly, each context word c is also represented as a vector in the same vector space.

Summary of the Process

1. Collect all correct pairs D_{wc} from the corpus based on actual co-occurrences.
2. Randomly sample incorrect pairs D_{wr} by choosing words that have never co-occurred with the target word.
3. Use both sets of pairs to train the Skip-gram model:
    - The model aims to maximize the probability of correct pairs while minimizing the probability of incorrect pairs.

By training on both correct and incorrect pairs, the Skip-gram model can effectively learn meaningful word embeddings that capture semantic relationships in the language.

Objective

The goal is to **maximize the probability** of correct word-context pairs while minimizing the probability of incorrect pairs. This is accomplished through the use of probabilistic models and the sigmoid function.

Probability Modeling

1. **Probability of Correct Pair**:
    - For a given word-context pair (w, c) from the set of correct pairs D_{wc}: $$ P(z = 1 | w, c) = \sigma(u_c^T v_w) $$
    - Here, u_c is the context word representation, v_w is the target word representation, and σ is the sigmoid function defined as: $$ \sigma(x) = \frac{1}{1 + e^{-x}} $$
2. **Probability of Incorrect Pair**:
    - For a pair (w, r) from the set of incorrect pairs D_{wr}: $$ P(z = 0 | w, r) = \sigma(-u_r^T v_w) $$

- This indicates we want to model the probability that a randomly sampled context word r is not a valid context for the target word w.

Combining Probabilities

To derive a comprehensive objective function for training the model, we combine the probabilities from the correct and incorrect pairs:

1. **Maximizing the Product of Probabilities**:
   - The overall objective can be expressed as: $$ \text{maximize} \prod_{(w,c) \in D_{wc}} P(z = 1 | w, c) \prod_{(w,r) \in D_{wr}} P(z = 0 | w, r) $$
2. **Logarithmic Transformation**:
   - To facilitate optimization, we take the logarithm of the product: $$ = \text{maximize} \sum_{(w,c) \in D_{wc}} \log P(z = 1 | w, c) + \sum_{(w,r) \in D_{wr}} \log P(z = 0 | w, r) $$
   - This turns the product into a sum, which is easier to work with mathematically.

Final Objective Function

Substituting the probability models into the objective function gives:

1. **Substituting Probabilities**: $$ = \text{maximize} \sum_{(w,c) \in D_{wc}} \log \sigma(u_c^T v_w) + \sum_{(w,r) \in D_{wr}} \log \sigma(-u_r^T v_w) $$
2. **Using the Sigmoid Function**: $$ = \text{maximize} \sum_{(w,c) \in D_{wc}} \log \left(\frac{1}{1 + e^{-u_c^T v_w}}\right) + \sum_{(w,r) \in D_{wr}} \log \left(\frac{1}{1 + e^{u_r^T v_w}}\right) $$

Key Takeaways

- **Objective**: The final objective function aims to maximize the likelihood of the correct word-context pairs while minimizing the likelihood of incorrect pairs.
- **Dot Product**: The dot product $u_c^T v_w$ and $-u_r^T v_w$ serves to quantify the similarity between the context and word embeddings, which is crucial for learning effective word representations.
- **Use of the Sigmoid Function**: The sigmoid function is key to mapping the outputs to probabilities, ensuring that the model outputs are in the range [0, 1].

Conclusion

This mathematical framework helps to train the Skip-gram model efficiently, leveraging both positive (correct) and negative (incorrect) examples to learn meaningful embeddings for words. By maximizing the specified objective function, the model learns to distinguish valid contexts for words from those that do not co-occur, leading to better performance in various NLP tasks.

Notation Breakdown

1. **$P(z = 1 | w, c)$**:
   - **Meaning**: The probability that the word-context pair (w, c) is correct (i.e., $z = 1$).

- **Interpretation**: This tells us how likely it is that the context word c actually appears with the word w.

2. **$P(z = 0 | w, r)$**:
   - **Meaning**: The probability that the word-context pair (w, r) is incorrect (i.e., $z = 0$).
   - **Interpretation**: This indicates how likely it is that the context word r does not appear with the word w.

3. **D_{wc}**:
   - **Meaning**: The set of all correct word-context pairs.
   - **Interpretation**: This is a collection of pairs where the context words are valid for their corresponding target words.

4. **D_{wr}**:
   - **Meaning**: The set of all incorrect word-context pairs.
   - **Interpretation**: This includes pairs where the context words are not valid for their corresponding target words.

5. **u_c**:
   - **Meaning**: The vector representation (embedding) of the context word c.
   - **Interpretation**: This is a numerical representation that captures the semantic meaning of the context word.

6. **v_w**:
   - **Meaning**: The vector representation (embedding) of the target word (w).
   - **Interpretation**: Similar to (u_c), this represents the target word's meaning in numerical form.

7. **$\sigma(x)$**:
   - **Meaning**: The sigmoid function, defined as $\sigma(x) = \frac{1}{1 + e^{-x}}$.
   - **Interpretation**: This function maps any real number to a value between 0 and 1, effectively converting scores (like dot products) into probabilities.

8. **$u_c^T v_w$**:
   - **Meaning**: The dot product of the context word vector u_c and the target word vector v_w.
   - **Interpretation**: This measures the similarity between the context word and the target word. A higher value indicates greater similarity.

9. **$-u_r^T v_w$**:
   - **Meaning**: The negative dot product of the incorrect context word vector u_r and the target word vector v_w.
   - **Interpretation**: This reflects how dissimilar the incorrect context word r is from the target word w. A lower (more negative) value indicates less likelihood of being a valid context.

10. **$\prod$**:

- **Meaning**: Represents the product of multiple terms.
- **Interpretation**: In this context, it combines probabilities for all correct or incorrect pairs.

11. **$\sum$**:

- **Meaning**: Represents the sum of multiple terms.
- **Interpretation**: It aggregates the log probabilities for all pairs in D_{wc} or D_{wr}.

Overall Objective

- **Objective Function**: $$ \text{maximize} \sum_{(w,c) \in D_{wc}} \log P(z = 1 | w, c) + \sum_{(w,r) \in D_{wr}} \log P(z = 0 | w, r) $$
  - **Interpretation**: The goal is to maximize the total log probability of correct pairs while minimizing the log probability of incorrect pairs. This helps the model learn which contexts are valid for which words.

Combining Probabilities

1. **Starting Equation**: $$ \text{maximize} \prod_{(w,c) \in D_{wc}} P(z = 1 | w, c) \prod_{(w,r) \in D_{wr}} P(z = 0 | w, r) $$

   - **Meaning**: We want to maximize the product of the probabilities that a word-context pair (w, c) is correct and that a word-random context pair (w, r) is incorrect.

2. **Reformulation**: $$ = \text{maximize} \prod_{(w,c) \in D_{wc}} P(z = 1 | w, c) \prod_{(w,r) \in D_{wr}} (1 - P(z = 1 | w, r)) $$

   - **Meaning**: The second term changes from $P(z = 0 | w, r)$ to $1 - P(z = 1 | w, r)$ because $P(z = 0 | w, r) = 1 - P(z = 1 | w, r)$.

Logarithmic Transformation

3. **Taking the Logarithm**: $$ = \text{maximize} \sum_{(w,c) \in D_{wc}} \log P(z = 1 | w, c) + \sum_{(w,r) \in D_{wr}} \log (1 - P(z = 1 | w, r)) $$
   - **Meaning**: The product of probabilities is converted to a sum of log probabilities, which is mathematically more manageable.

Substituting Probability Models

4. **Substituting the Sigmoid Function**: $$ = \text{maximize} \sum_{(w,c) \in D_{wc}} \log \left(\frac{1}{1 + e^{-v_c^T v_w}} \right) + \sum_{(w,r) \in D_{wr}} \log \left(\frac{1}{1 + e^{v_r^T v_w}} \right) $$

   - **Meaning**: Here, $v_c^T v_w$ is the dot product of the context and target word embeddings. The sigmoid function σ is applied to model the probabilities.

5. **Final Objective Function**: $$ = \text{maximize} \sum_{(w,c) \in D_{wc}} \log \sigma(v_c^T v_w) + \sum_{(w,r) \in D_{wr}} \log \sigma(-v_r^T v_w) $$

   - **Meaning**: This step aggregates the log probabilities of correct pairs and the log probabilities of incorrect pairs using the sigmoid function.

Definitions of Terms

- **Sigmoid Function**: $$ \sigma(x) = \frac{1}{1 + e^{-x}} $$
    - **Meaning**: The sigmoid function transforms any real-valued number into the range [0, 1], allowing it to be interpreted as a probability.

Interpretation of the Objective Function

- The objective function is structured to:
    - **Maximize the log probability** of the correct word-context pairs (w, c) being classified correctly (i.e., $z = 1$).
    - **Maximize the log probability** of the incorrect word-context pairs (w, r) being classified as incorrect (i.e., $z = 0$).

Summary

1. **Maximization Goal**: The overall aim is to adjust the word and context embeddings to improve the probabilities of correct pairs and reduce the probabilities of incorrect pairs.
2. **Role of Dot Products**: The dot products $v_c^T v_w$ and $-v_r^T v_w$ play a crucial role in determining how similar or dissimilar the word and context pairs are, influencing the sigmoid function's output.
3. **Logarithmic Form**: Taking the logarithm simplifies the optimization process, making it easier to work with sums rather than products.

Negative Sampling Overview

1. **Purpose of Negative Sampling**:

    - Negative sampling is used to improve the efficiency of training word embeddings by reducing the computational load associated with softmax. Instead of predicting probabilities for all words in the vocabulary, the model focuses on a few sampled words.

2. **Sampling k Negative Pairs**:

    - For every positive word-context pair (w, c) (where w is the target word and c is the correct context word), the model samples **k negative pairs** (w, r). Here, r represents a randomly selected context word that is **not** a valid context for w.

3. **Size of the Dataset**:

    - Given that there are $|D_{wc}|$ positive pairs, the total number of pairs in the dataset D becomes: $$ |D| = k \cdot |D_{wc}| $$
    - This means the size of the dataset now includes the original positive pairs plus the additional negative pairs, enhancing the learning process.

Probability Distribution for Sampling

4. **Modified Unigram Distribution**:
    - The random context word r is drawn from a **modified unigram distribution**. The probability of selecting a context word r is determined based on its frequency in the corpus: $$ P(r) =

\frac{\text{count}(r)^{3/4}}{N} $$

- **Where**:
  - $\text{count}(r)$ is the number of times the word r appears in the corpus.
  - N is the total number of words in the corpus (i.e., the sum of all words in all sentences).

Key Concepts Explained

- **Unigram Distribution**: This is a simple probability distribution where the probability of each word is proportional to its frequency in the corpus. However, in this case, the counts are raised to the power of $3/4$ to reduce the likelihood of selecting very frequent words as negatives. This adjustment helps to balance the sampling process, making it less biased toward very common words.
- **Total Number of Words N**: The total number of words in the corpus serves as a normalizing factor. It ensures that the probabilities sum to 1 when considering all words in the vocabulary.

Summary

1. **Negative Sampling**: A technique to make training more efficient by using a smaller number of negative samples, leading to faster convergence of the model.
2. **Sampling Distribution**: The modified unigram distribution ensures that less frequent words have a higher chance of being selected as negative samples compared to very frequent words, helping to create more informative negative pairs.
3. **Impact on Training**: By employing negative sampling, the Skip-gram model can effectively learn word embeddings with fewer computational resources while improving the overall quality of the embeddings produced.

Overview of Contrastive Estimation

1. **Definition**:
   - Contrastive estimation is a method used to train models by contrasting positive examples (correct word-context pairs) against negative examples (incorrect pairs). This approach helps to reinforce the model's understanding of valid word relationships while discouraging invalid ones.
2. **Contextual Setup**:
   - In your example, you have a positive sentence: **"He sat on a chair."**
   - An associated negative example might be: **"He sat abracadabra a chair."**
   - The goal is to ensure that the model can differentiate between valid context pairs and nonsensical ones.

Softmax Function and its Challenges

3. **Softmax Function**:
   - The softmax function is used to convert raw scores (logits) into probabilities. However, it can be computationally expensive, especially when dealing with large vocabularies.
4. **Solutions to the Softmax Issue**:

- **Solution 1**: Use **negative sampling** to reduce the number of comparisons needed.
- **Solution 2**: Employ **contrastive estimation** to maximize the difference between scores for positive and negative pairs.
- **Solution 3**: Implement **hierarchical softmax**, which reduces complexity by structuring the output layer.

Objective of Contrastive Estimation

5. **Scoring Mechanism**:
   - Let s represent the score of the positive context (e.g., for "sat" in "He sat on a chair").
   - Let sc represent the score of the negative context (e.g., for "abracadabra" in "He sat abracadabra a chair").
   - The objective is to maximize the difference between s and sc: $$ \text{maximize } (s - sc) $$
   - Additionally, you want this difference to be at least some margin m: $$ \text{maximize } (s - (sc + m)) $$
     - This ensures that the positive score is not only greater than the negative score but also exceeds it by a margin m.

Mathematical Formulation

6. **Final Objective Function**:
   - The objective can be expressed as: $$ \text{maximize } \max(0, s - (sc + m)) $$
   - Here, the function returns the positive difference if it exceeds $sc + m$; otherwise, it returns zero (i.e., if the positive score does not surpass the negative score by the required margin).

Implications

- **Training Objective**: By maximizing this expression, the model learns to give significantly higher scores to valid word-context pairs compared to invalid ones, effectively enhancing its ability to discriminate between correct and incorrect relationships.

- **Effectiveness of Contrastive Estimation**: This method allows the model to improve its predictions without needing to compute the softmax over the entire vocabulary, thus making training more efficient.

Summary

1. **Contrastive Estimation**: A training approach that contrasts positive and negative examples to reinforce learning.
2. **Softmax Alternatives**: Techniques like negative sampling and contrastive estimation help mitigate the computational burden of softmax.
3. **Objective Function**: Focuses on ensuring positive scores exceed negative scores by a defined margin, promoting robust embeddings.

Hierarchical Softmax Overview

1. **Purpose**:

   - Hierarchical softmax is an efficient alternative to the traditional softmax function used in training neural networks, especially in NLP tasks like word embeddings. It is designed to handle large

vocabularies more efficiently.

2. **Binary Tree Structure**:
    - The hierarchical softmax constructs a **binary tree** where:
        - Each **leaf node** corresponds to a unique word in the vocabulary.
        - The number of leaf nodes is equal to the number of words, denoted as V.
3. **Path Representation**:
    - For any word w, there is a unique path from the **root node** to the leaf node representing that word.
    - Let $l(w_1), l(w_2), \ldots, l(w_p)$ be the nodes along this path.
4. **Binary Vector Encoding**:
    - A **binary vector** $\pi(w)$ encodes the path from the root to the leaf node for word w:
        - $\pi(w)_k = 1$: Indicates a left branch taken at node $l(w_k)$.
        - $\pi(w)_k = 0$: Indicates a right branch taken at node $l(w_k)$.

Model Parameters

5. **Internal Node Vectors**:
    - Each **internal node** in the tree is associated with a vector u_i. These vectors are learned during training and contribute to the semantic representation of the words.
6. **Parameters**:
    - The model has parameters that include:
        - W_{context}: A matrix representing context information for the words.
        - $u_1, u_2, \ldots, u_V$: Vectors for each internal node in the binary tree.

How Hierarchical Softmax Works

7. **Probability Calculation**:
    - To compute the probability of a word w given a context c using hierarchical softmax, the model traverses the tree:
        - For each node in the path to the leaf node corresponding to w, a binary decision is made (left or right).
        - This involves calculating the sigmoid function for each internal node: $$ P(w \mid c) = \prod_{k=1}^{p} \sigma(u_{l(w_k)}^T v_c) $$
        - Here, v_c is the context vector, and $u_{l(w_k)}$ is the vector associated with the internal node at position k in the path to the leaf node representing w.
8. **Efficiency**:
    - The advantage of hierarchical softmax is that instead of computing the softmax over all V words, it only requires a fixed number of calculations proportional to the depth of the tree (usually $\log_2 V$). This makes it significantly faster for large vocabularies.

Advantages of Hierarchical Softmax

9. **Scalability**:
    - Suitable for large vocabularies, as it scales logarithmically rather than linearly with the number of words.
10. **Memory Efficiency**:
    - Reduces the memory footprint needed to store and compute the probabilities for word predictions, making it easier to work with large datasets.
11. **Improved Training Speed**:
    - Speeds up training times due to reduced computational complexity, which is especially beneficial in models with large vocabularies.

Summary of Key Points

- **Hierarchical Softmax**: An efficient softmax alternative using a binary tree structure.
- **Binary Tree**: Each word is a leaf node, with a unique path from the root.
- **Binary Vector $\pi(w)$**: Encodes the path taken to reach the word w.
- **Internal Node Vectors u_i**: Parameters contributing to semantic representation.
- **Efficiency**: Only requires logarithmic computations with respect to vocabulary size.

Conclusion

Hierarchical softmax provides an elegant solution to the challenges posed by traditional softmax in natural language processing, allowing for efficient learning and representation of words in large vocabularies. It leverages tree structures and binary encoding to minimize computational costs while maintaining effective learning dynamics.

Key Concepts

1. **Probability of a Word Given Context**:
    - For a given word w and context c, we are interested in calculating the probability $p(w|v_c)$, where v_c is the vector representation of the context.
2. **Modeling the Probability**:
    - The probability of a word given the context is modeled as a product of probabilities along a unique path in the binary tree: $$ p(w|v_c) = \prod_k \pi(w_k|v_c) $$
    - Here, k indexes the nodes along the path from the root node to the leaf node corresponding to the word w.

Example of Probability Calculation

3. **Example for a Specific Word**:
    - For the word **"on"** given the context **"sat"**: $$ P(on|v_{sat}) = P(\pi(on)*1*=*1*|*v*{sat}) \times P(\pi(on)*2*=*0*|*v*{sat}) \times P(\pi(on)*3*=*0*|*v*{sat}) $$

- This equation demonstrates that the overall probability of predicting the word **"on"** depends on the probabilities of traversing the binary tree correctly through its left and right branches.

Probability Calculation Details

4. **Modeling Branching Decisions**:
   - Each decision to branch left or right at a node i can be represented using a sigmoid function:
     - For branching left: $$ P(\pi(on)_i = 1) = \frac{1}{1 + e^{-v_c^T u_i}} $$
     - For branching right: $$ P(\pi(on)_i = 0) = 1 - P(\pi(on)_i = 1) = \frac{e^{-v_c^T u_i}}{1 + e^{-v_c^T u_i}} = \frac{1}{1 + e^{v_c^T u_i}} $$
   - Here, u_i is the vector representation of the internal node i, and v_c is the vector representation of the context word.

Semantic Relationships

5. **Node Representation and Similarity**:
   - The model ensures that:
     - The representation of a context word v_c will have **high similarity** with the representation of the node u_i if the path to the word w branches left at node i.
     - Conversely, it will have **low similarity** if the path branches right at node i.
   - This relationship helps to enforce semantic meanings, as context words that appear with the same target words will have similar representations.

Summary of Key Points

- **Path Representation**: The probability of predicting a word can be derived from the unique path taken in the binary tree, highlighting the hierarchical structure.
- **Sigmoid Function**: The sigmoid function is used to model decisions at each node (left or right), which is critical for estimating the probability of a word given its context.
- **Semantic Similarity**: The model inherently captures semantic relationships through the similarities of vector representations for words and context, enabling more meaningful predictions.

Conclusion

Hierarchical softmax provides a computationally efficient way to predict word probabilities based on context by leveraging a binary tree structure and capturing semantic relationships through learned embeddings. The transition from words to probabilities using a path-based approach simplifies the learning process while ensuring meaningful representation.

Probability Equation

1. **Probability of a Word Given Context**:
   - The equation you provided is: $$ P(w|v_c) = \prod_{k=1}^{|\pi(w)|} P(\pi(w)_k|v_c) $$
   - In this equation:
     - $P(w|v_c)$ represents the probability of the word w given the context vector v_c.
     - $\pi(w)$ is the path from the root of the binary tree to the leaf node corresponding to the word w.

- $|\pi(w)|$ is the length of that path, or the number of nodes traversed to reach the leaf node representing w.
- Each $P(\pi(w)_k|v_c)$ computes the probability of taking the k^{th} step in the path (either left or right) based on the context vector.

Efficiency of Hierarchical Softmax

2. **Efficiency Compared to Traditional Softmax**:
    - Using hierarchical softmax significantly reduces computational complexity:
        - Instead of requiring $|V|$ computations (where $|V|$ is the vocabulary size) as in traditional softmax, hierarchical softmax only requires $|\pi(w)|$ computations.
        - Since $|\pi(w)|$ (the depth of the binary tree) is typically much smaller than $|V|$, this makes hierarchical softmax much faster and more efficient, especially for large vocabularies.

Constructing the Binary Tree

3. **Binary Tree Construction**:

    - **Random Arrangement**:
        - A practical aspect of hierarchical softmax is that the binary tree does not need to follow any specific structure regarding word semantics.
        - In practice, even a **random arrangement** of words on the leaf nodes of the binary tree performs well. This means that the tree can be constructed without any specific consideration of the relationships between words; as long as each word is a leaf node, the model can effectively learn word representations.

4. **Why Random Works**:

    - The reason a random arrangement can be effective is that the model learns the necessary semantic relationships during training based on the context in which words appear, rather than relying on the initial arrangement of the tree.
    - As the training progresses, the embeddings (representations) of the words will adjust based on their co-occurrences with context words, making the specific arrangement less critical.

Summary of Key Points

- **Computational Efficiency**: Hierarchical softmax reduces the number of computations needed to calculate the probability of a word given context from the size of the vocabulary to the depth of the binary tree.
- **Random Binary Tree**: A random arrangement of words in the binary tree is sufficient for effective training, as the model learns relationships through context rather than relying on the tree structure.
- **Learning Process**: The training process adjusts word embeddings based on their contextual relationships, allowing the model to effectively capture semantics.

1. Overview of GloVe Representations

- **Count-Based Methods**:

    - These methods, such as Singular Value Decomposition (SVD), rely on global co-occurrence counts from a corpus to compute word representations.
    - They use the entire corpus to gather statistics about how often words appear together, which captures the relationship between words.

- **Predictive Methods**:

    - These methods learn word representations using co-occurrence information in a more dynamic way.
    - Instead of relying solely on global statistics, they predict word context based on surrounding words in a given window.

- **Combining Both Approaches**:

    - The idea is to leverage the strengths of both count-based and predictive methods to create more accurate word representations. This approach aims to ensure that word vectors reflect true co-occurrence statistics while still being learned through the prediction of contexts.

2. Corpus Example

- **Example Sentences**:

    - "Human machine interface for computer applications"
    - "User opinion of computer system response time"
    - "User interface management system"
    - "System engineering for improved response time"

- **Co-occurrence Matrix (X)**:

    - This matrix captures the frequency of co-occurrences between words in the corpus.
    - Each entry X_{ij} represents how often word i appears in the context of word j.

3. Co-occurrence Count Notation

- **Co-occurrence Probabilities**:
    - The probability of word j occurring in the context of word i is given by: $$ P(j|i) = \frac{X_{ij}}{X_i} $$
    - Here, X_{ij} is the co-occurrence count, and X_i is the total count of occurrences of word i.

4. Learning Word Vectors

- **Objective of Learning**:
    - The goal is to learn word vectors v_i and v_j such that their dot product is equal to the logarithm of their co-occurrence probability: $$ v_i^T v_j \approx \log P(j|i) = \log X_{ij} - \log X_i $$
    - This means that the dot product of the vectors should reflect the co-occurrence statistics.

5. Optimization Problem Formulation

- **Combined Objective**:

- To enforce the relationship between the word vectors and the co-occurrence counts, we derive: $$ v_i^T v_j + b_i + b_j = \log X_{ij} $$
- Here, b_i and b_j are biases associated with words i and j.

- **Optimization Function**:
  - The optimization problem can be framed as minimizing the squared difference between the predicted value (from the model) and the actual log co-occurrence: $$ \min_{v_i, v_j, b_i, b_j} \left(v_i^T v_j + b_i + b_j - \log X_{ij} \right)^2 $$

6. Weighting Function

- **Challenge with Equal Weighting**:
  - The initial formulation treats all co-occurrences equally, which can be a drawback since rare and frequent words may not contribute equally to learning.

- **Weighted Optimization**:
  - To address this, a weighting function $f(X_{ij})$ is introduced, which adjusts the contributions of different co-occurrences: $$ \min_{v_i, v_j, b_i, b_j} f(X_{ij}) \left(v_i^T v_j + b_i + b_j - \log X_{ij} \right)^2 $$

- **Desired Properties of Weighting Function**:
  - The weighting function $f(X_{ij})$ should ensure that neither rare nor frequent words are over-weighted.
  - A common form is: $$ f(x) = \left(\frac{x}{x_{\text{max}}} \right)^\alpha \quad \text{if } x < x_{\text{max}} $$
  - Here, x_{max} is a threshold that can be tuned for specific datasets.

7. Summary

- GloVe combines the strengths of count-based and predictive methods to generate word representations that reflect true word relationships.
- The use of a co-occurrence matrix allows for a clear statistical foundation for learning.
- The optimization process aims to minimize the difference between predicted and actual log probabilities while incorporating a weighting function to manage the contribution of each co-occurrence.

Table Structure

The table can be read as follows:

	human	**machine**	**system**	**for**	**...**	**user**
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29

	human	machine	system	for	...	user
for	2.14	2.14	0.96	1.87	...	-0.13
...	...	...	...	...	...	...
user	0.43	0.43	1.29	-0.13	...	1.71

Explanation of Each Element

- **Rows and Columns**:
    - The rows and columns of the matrix represent different words in the vocabulary. For example, "human", "machine", "system", "for", and "user" are all words from the corpus.
- **Matrix Entries**:
    - Each entry in the matrix, such as the one at the intersection of "human" and "system" (0.23), represents the **co-occurrence count** or weight between the two words. This could denote how frequently these words appear together in the context of the surrounding text.
    - **Positive Values**: The positive values indicate how frequently the word pair occurs together. For instance, "human" appears in the context of "for" with a frequency of 2.14, suggesting that whenever "for" appears, "human" also appears often in the nearby context.
    - **Negative Values**: The entry -0.13 for the co-occurrence between "for" and "user" could suggest an unusual or rare negative association, or it could indicate a reduced count or weight in the context, possibly hinting at infrequent co-occurrences or adjustments made during preprocessing.

Purpose of the Co-occurrence Matrix

1. **Semantic Relationships**:
    - This matrix serves as a representation of semantic relationships between words based on their co-occurrence patterns in the corpus. Words that frequently occur together tend to be semantically related.
2. **Word Vector Learning**:
    - In models like GloVe (Global Vectors for Word Representation), this co-occurrence matrix is used to train word vectors. The goal is to learn a set of word embeddings (vector representations) such that the dot product of the vectors reflects the log probability of the co-occurrence counts.
3. **Contextual Information**:
    - The matrix captures contextual information. For example, words that share common contexts (like "human" and "machine") can be grouped or represented similarly in a high-dimensional vector space.
4. **Dimensionality Reduction**:

- High-dimensional co-occurrence matrices can be reduced to lower-dimensional vector representations (embeddings), making them easier to work with for various natural language processing tasks such as classification, clustering, and semantic analysis.

Summary

The co-occurrence matrix is a crucial element in natural language processing, enabling the extraction of meaningful relationships between words based on their usage in context. By analyzing these co-occurrences, models can learn to represent words in a way that captures their meanings and relationships in a mathematically tractable format.

---

1. Semantic Relatedness

Definition: Semantic relatedness measures how closely related two words are in meaning.

- **Human Judgments**:
  - Humans are asked to judge the relatedness between pairs of words. For example, given the pair **(cat, dog)**, a human might assign a score of $$ S_{\text{human}}(\text{cat, dog}) = 0.8 $$, indicating a high degree of relatedness.
- **Model Computation**:
  - Compute the cosine similarity between the word vectors corresponding to the two words: $$ S_{\text{model}}(\text{cat, dog}) = \frac{v_{\text{cat}}^T v_{\text{dog}}}{|v_{\text{cat}}| |v_{\text{dog}}|} $$
  - If the model's cosine similarity for the same pair is $$ S_{\text{model}}(\text{cat, dog}) = 0.7 $$, it suggests a good, but not perfect, alignment with human judgment.
- **Correlation Analysis**:
  - Given a large dataset of word pairs, compute the correlation between the model's scores ($$ S_{\text{model}} $$) and human judgments ($$ S_{\text{human}} $$).
  - Compare different models:
    - **Model 1** is considered better than **Model 2** if: $$ \text{correlation}(S_{\text{model}1}, S{\text{human}}) > \text{correlation}(S_{\text{model}2}, S{\text{human}}) $$

2. Synonym Detection

Definition: The task of identifying a synonym for a given term from a list of candidates.

- **Example**:
  - **Term**: *levied*
  - **Candidates**: *unposed, believed, requested, correlated*
- **Computation**:
  - For each candidate, compute the cosine similarity with the term: $$ \text{Synonym} = \text{argmax}{c \in C} \left(\frac{v{\text{term}}^T v_{c}}{|v_{\text{term}}| |v_{c}|}\right) $$
  - Select the candidate with the largest cosine similarity.

- **Evaluation**:
    - Compute the accuracy of different models based on how many times they correctly identify the synonym.

3. Analogy

Definition: The task of solving analogies, typically framed as "A is to B as C is to D."

- **Semantic Analogy**:
    - Example: Find the nearest neighbor for **sister** using the relation: $$ v_{\text{brother}} + v_{\text{grandson}} \approx v_{\text{sister}} + v_{\text{?}} $$
    - This implies searching for a vector $$ v_{?} $$ that maintains the semantic relationship.
- **Syntactic Analogy**:
    - Example: Work with syntactic relationships: $$ v_{\text{works}} - v_{\text{work}} + v_{\text{speak}} \approx v_{\text{?}} $$
    - This reflects a different aspect of word relationships based on grammatical forms.
- **Evaluating Algorithms**:
    - **Boroni et al. (2014)** found that predictive models (like Word2Vec) consistently outperform count-based models (like GloVe) in all evaluation tasks.
    - **Levy et al. (2015)** conducted a thorough analysis and found that:
        - Singular Value Decomposition (SVD) performed better than prediction-based models on similarity tasks.
        - Prediction-based models excelled on analogy tasks.

The relationship between **Singular Value Decomposition (SVD)** and **Word2Vec** lies in how both techniques approach the task of generating word embeddings from co-occurrence information in a corpus.

1. Matrix Factorization

SVD:

- SVD is a technique used to factorize a matrix into three components: $$ M \approx U \Sigma V^T $$
    - **M**: Original matrix (e.g., a co-occurrence matrix).
    - **U**: Left singular vectors (representations of words).
    - **Σ**: Diagonal matrix containing singular values.
    - **V^T**: Right singular vectors (representations of context).
- In the context of word embeddings, SVD operates on a co-occurrence matrix that captures how often words appear together across the corpus.

Word2Vec:

- Word2Vec, particularly in its Continuous Bag of Words (CBOW) and Skip-Gram models, also performs a form of matrix factorization.

- It does this indirectly by learning two separate weight matrices:
    - **W_context**: Represents context words.
    - **W_word**: Represents target words.
- These matrices can be thought of as learned embeddings for words and their contexts.

2. Relationship Between the Models

- Levy et al. (2015) showed that Word2Vec can be seen as performing an implicit matrix factorization of a certain matrix **M** that relates to the **Pointwise Mutual Information (PMI)** matrix: $$ M = W_{\text{context}} W_{\text{word}} $$
- **PMI** is a measure of association between two words and is computed as: $$ \text{PMI}(w_i, c_j) = \log \left(\frac{P(w_i, c_j)}{P(w_i) P(c_j)} \right) $$
    - Where $P(w_i, c_j)$ is the joint probability of words w_i and context c_j, and $P(w_i)$ and $P(c_j)$ are the individual probabilities of w_i and c_j.
- The matrix **M** can also be expressed in terms of a modified co-occurrence matrix where: $$ M_{ij} = \text{PMI}(w_i, c_j) \cdot \log(k) $$
    - k is the number of negative samples used in the Word2Vec training process, and it adjusts the influence of negative samples on learning.

3. Implications of the Relationship

- Both SVD and Word2Vec aim to represent word meanings based on their usage in context. The primary difference is in their approaches:
    - **SVD** operates directly on a full co-occurrence matrix, which can be computationally expensive for large vocabularies.
    - **Word2Vec** learns embeddings through stochastic gradient descent, optimizing for prediction tasks (such as predicting context words given a target word) rather than directly modeling co-occurrences.
- The insight that Word2Vec performs an implicit matrix factorization helps understand why both methods can yield similar embeddings under certain conditions, despite their different operational mechanisms.

Summary

In conclusion, while SVD and Word2Vec employ different methodologies, both are fundamentally about uncovering relationships between words based on co-occurrence data. Word2Vec can be viewed as an efficient, scalable method for performing a type of matrix factorization similar to that of SVD, allowing it to produce high-quality word embeddings for various natural language processing tasks.