Skip-gram Model Explanation

The Skip-gram Model

Goal: The primary objective of the skip-gram model is to maximize the probability of context words given a target word. In other words, for a given word, the model tries to predict the words that appear in its context.

Mathematical Formulation

Given a target word w_t and its context words $w_{t-k}, w_{t-(k-1)}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k-1}, w_{t+k}$, the goal is to maximize the conditional probability:

$$\prod_{-k \leq j \leq k, j \neq 0} p(w_{t+j}|w_t)$$

Example Sentence

Consider the sentence: "The quick brown fox jumps over the lazy dog."

Let's choose the target word $w_t =$ "fox" and a context window size of 2. The context words within a window size of 2 around "fox" are "quick", "brown", "jumps", and "over".

Training Process

1. Vocabulary and One-hot Encoding

First, we create a vocabulary of unique words in the corpus and assign an index to each word. For simplicity, let's assume our vocabulary is:

$$\{\text{"the"}, \text{"quick"}, \text{"brown"}, \text{"fox"}, \text{"jumps"}, \text{"over"}, \text{"lazy"}, \text{"dog"}\}$$

Each word can be represented as a one-hot vector. For example, "fox" is represented as:

$$\text{"fox"} = [0, 0, 0, 1, 0, 0, 0, 0]$$

2. Word Embeddings

We initialize word embeddings randomly. Let's assume we use 3-dimensional embeddings. For instance, the initial embeddings might be:

$$\text{"fox"} = [0.2, -0.1, 0.3] \text{"quick"} = [0.4, 0.3, 0.1]$$

3. Forward Pass

The input to the model is the one-hot vector of the target word. The output is the predicted probability distribution over the vocabulary for each context word.

For the target word "fox", the input one-hot vector is:

$$[0, 0, 0, 1, 0, 0, 0, 0]$$

This is multiplied by the embedding matrix to get the word embedding for "fox":

$$\text{Embedding("fox")} = [0.2, -0.1, 0.3]$$

4. Predicting Context Words

We use the embedding of "fox" to predict its context words. For simplicity, assume the softmax layer directly outputs the probabilities for each word in the vocabulary.

Let's say the model predicts the following probabilities for the context words:

$$\begin{aligned} \text{"quick"} &: 0.15 \\ \text{"brown"} &: 0.30 \\ \text{"jumps"} &: 0.20 \\ \text{"over"} &: 0.10 \\ \text{other words} &: 0.25 \end{aligned}$$

5. Loss Calculation

We use the cross-entropy loss to measure the difference between the predicted probabilities and the actual context words. For the actual context words "quick", "brown", "jumps", and "over", the loss can be calculated as:

$$\text{Loss} = -(\log(0.15) + \log(0.30) + \log(0.20) + \log(0.10))$$

6. Backpropagation and Parameter Update

The loss is then backpropagated to update the word embeddings and other parameters of the model. This process is repeated for each target word in the corpus until the model converges.

Numerical Example

Let's work through a numerical example with the following simplified data:

- Vocabulary: ["the", "quick", "brown", "fox", "jumps", "over", "lazy", "dog"]
- Embedding size: 2
- Initial embeddings:

$$\begin{aligned}
\text{"the"} &: [0.1, 0.2] \\
\text{"quick"} &: [0.4, 0.3] \\
\text{"brown"} &: [0.2, 0.5] \\
\text{"fox"} &: [0.6, 0.1] \\
\text{"jumps"} &: [0.1, 0.4] \\
\text{"over"} &: [0.3, 0.2] \\
\text{"lazy"} &: [0.5, 0.6] \\
\text{"dog"} &: [0.2, 0.4]
\end{aligned}$$

Target word: "fox" -¿ Embedding: [0.6, 0.1]
Context words: "quick", "brown", "jumps", "over"
Assume the predicted probabilities (softmax output) for context words are:

$$\begin{aligned}
\text{"quick"} &: 0.25 \\
\text{"brown"} &: 0.30 \\
\text{"jumps"} &: 0.20 \\
\text{"over"} &: 0.10 \\
\text{other words} &: 0.15
\end{aligned}$$

Actual context words (one-hot encoded):

$$\begin{aligned}
\text{"quick"} &: [0, 1, 0, 0, 0, 0, 0, 0] \\
\text{"brown"} &: [0, 0, 1, 0, 0, 0, 0, 0] \\
\text{"jumps"} &: [0, 0, 0, 0, 1, 0, 0, 0] \\
\text{"over"} &: [0, 0, 0, 0, 0, 1, 0, 0]
\end{aligned}$$

Cross-entropy loss:

$$\text{Loss} = -(\log(0.25) + \log(0.30) + \log(0.20) + \log(0.10))$$

Simplifying:

$$\text{Loss} \approx -(-1.386 + -1.204 + -1.609 + -2.303) \approx 6.502$$

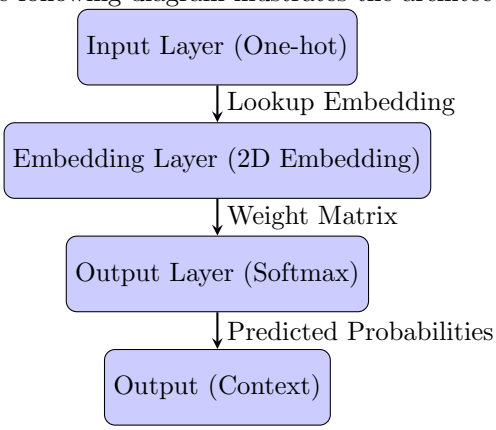
Conclusion

The skip-gram model uses the target word to predict its context words within a specified window. By training on a large corpus, the model learns to generate meaningful word embeddings that capture semantic relationships between words. The embeddings are updated iteratively through forward and backward passes until the model converges to minimize the loss.

Note: Embeddings are the hidden layer in the skip-gram model.

Skip-gram Model Architecture

The following diagram illustrates the architecture of the skip-gram model:



1 Explanation

- **Input Layer**: The input to the model is a one-hot vector representing the target word. In this example, the target word is "fox".
- **Hidden Layer (Embeddings)**: The one-hot vector is multiplied by the embedding matrix to obtain the dense vector representation (embedding) of the target word. For "fox", the embedding is (0.6, 0.1).
- **Output Layer**: The embedding vector is used to predict the probabilities of each word in the vocabulary being a context word. The model outputs the probabilities for the context words "quick", "brown", "jumps", and "over".

2 The Math Behind It

2.1 Conditional Probability

The main goal of the skip-gram model is to predict the probability of a word (let's call it c) appearing in the context of another word (let's call it w). This is represented mathematically as:

$$p(c|w;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}}$$

Here, v_c and v_w are the vector representations of the words c and w. The dot product $v_c \cdot v_w$ measures how similar the two words are based on their vectors. C is the set of all words that can be in the context of w.

2.2 Simplified Notation

Let's say:

$P(c|w)$ is the probability that c appears in the context of w.

The numerator is how likely c is given w. The denominator sums up the probabilities of all possible contexts.

2.3 Maximizing the Probability

We want to find the best vectors that maximize this probability. This means we want our model to do a great job at predicting contexts:

$$\text{Maximize} \sum_{(w,c) \in D} \log p(c|w)$$

Here, D is the dataset of word-context pairs.

2.4 Breaking It Down

The sum tells us to look at every pair of words and contexts in our dataset. The log helps us switch from multiplying probabilities (which can get very small) to adding them (which is easier to handle mathematically).

2.5 Using Logarithm

When we take the logarithm of the probability, we simplify the calculations:

$$\log p(c|w) = v_c \cdot v_w - \log \left(\sum_{c' \in C} e^{v_{c'} \cdot v_w} \right)$$

The first term $v_c \cdot v_w$ measures how similar c and w are. The second term, $\log \left(\sum_{c' \in C} e^{v_{c'} \cdot v_w} \right)$, is a little tricky because it sums over all possible context words, which can be very large and expensive to compute.

2.6 Why is it Expensive?

Calculating the probability for all possible context words can be very slow, especially when there are many words in the vocabulary (think of every word you know!). To make this faster, we can use something called hierarchical softmax. This is a method to simplify the calculations, but we won't go into detail about that right now.

2.7 The Key Assumption

The idea behind this model is that if we can maximize this probability, we will get good word embeddings (the vector representations) for each word. The assumption is:

Similar words will have similar vectors. For instance, the words "king" and "queen" might have similar vector representations because they often appear in similar contexts.

3 Negative Sampling

Negative sampling is an efficient method to derive word embeddings, introduced by Mikolov et al. While it's based on the skip-gram model, it optimizes a different objective.

3.1 Understanding the Objective

We start with a pair of words: (w, c), where w is a word and c is its context. Our goal is to determine whether this pair comes from the training data or not. We can represent this as: - $p(D = 1|w, c)$: the probability that the pair (w, c) came from our training data. - $p(D = 0|w, c) = 1 - p(D = 1|w, c)$: the probability that the pair did not come from the training data.

3.2 Maximizing Probabilities

Our aim is to find parameters that maximize the probability that all observations indeed came from the data:

$$\arg\max_\theta \prod_{(w,c)\in D} p(D = 1|w, c; \theta)$$

We can take the logarithm of this expression:

$$= \arg\max_\theta \sum_{(w,c)\in D} \log p(D = 1|w, c; \theta)$$

3.3 Using Softmax

We can define the probability $p(D = 1|w, c; \theta)$ using the softmax function:

$$p(D = 1|w, c; \theta) = \frac{1}{1 + e^{-v_c \cdot v_w}}$$

3.4 Formulating the Objective

Our objective becomes:

$$\arg\max_\theta \sum_{(w,c) \in D} \log \frac{1}{1 + e^{-v_c \cdot v_w}}$$

This has a trivial solution if we set θ such that $p(D = 1|w, c; \theta) = 1$ for every pair (w, c).

3.5 Preventing Uniformity

To avoid all vectors being the same, we need a mechanism that allows the model to see pairs (w, c) where the probability $p(D = 1|w, c; \theta)$ should be low. This is done by generating a set D' of random pairs (w, c) that we assume are incorrect.

3.6 Combining Objectives

Our optimization objective now includes both the correct pairs and the negative samples:

$$\arg\max_\theta \prod_{(w,c) \in D} p(D = 1|c, w; \theta) \prod_{(w,c) \in D'} p(D = 0|c, w; \theta)$$

This leads us to:

$$= \arg\max_\theta \sum_{(w,c) \in D} \log p(D = 1|c, w; \theta) + \sum_{(w,c) \in D'} \log(1 - p(D = 1|w, c; \theta))$$

3.7 Using the Sigmoid Function

If we let $\sigma(x) = \frac{1}{1+e^{-x}}$, we can rewrite our objective as:

$$= \arg\max_\theta \sum_{(w,c) \in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c) \in D'} \log \sigma(-v_c \cdot v_w)$$

3.8 Difference from Original Work

The main difference from the original work by Mikolov et al. is that we present the objective for the entire dataset $D \cup D'$, while they focused on one example (w, c) and k negative samples.

3.9 Sampling Negative Examples

Mikolov et al. constructed D' to be k times larger than D. For each $(w, c) \in D$, they generated k negative samples, $(w, c_1), \ldots, (w, c_k)$, where each c_j was drawn according to a specific probability distribution.

4 Remarks

- Unlike the skip-gram model, this formulation does not model $p(c|w)$ but rather a quantity related to the joint distribution of w and c.
- Fixing either the word representations or the context representations reduces the model to logistic regression, making it convex. However, learning both jointly leads to a non-convex model.