Word2Vec : Skip-Gram

From Vanilla Softmax to Negative Sampling (SGNS)

NLP Students

Contents

1 Introduction: The Word Embedding Revolution

In Natural Language Processing (NLP), computers require numerical representations of words. Traditional methods, such as One-Hot Encoding, treat words as atomic symbols (e.g., "cat"=[1, 0, 0, ...], "dog"=[0, 1, 0, ...]). This approach fails completely to capture semantic relationships; the similarity (dot product) between "cat" and "dog" is zero, the same as the similarity between "cat" and "philosophy".

Word Embeddings solve this by representing words as dense, low-dimensional vectors of real numbers. The positioning of these vectors in a high-dimensional space captures meaning, such that similar words are physically close.

This concept is rooted in the **Distributional Hypothesis** (J.R. Firth, 1957):

> *"You shall know a word by the company it keeps."*

Words appearing in similar contexts tend to have similar meanings.

The Skip-Gram model, part of the Word2Vec family introduced by Mikolov et al. (2013), is a highly influential technique for learning these embeddings by analyzing the contexts in which words appear across a large corpus.

The Intuition: Semantics as Geometry

We are essentially drawing a map of language. Words are coordinates (vectors) in this space. If "coffee" and "tea" are often used in similar situations (e.g., near "drink", "hot", "cup"), the algorithm places them close together on the map. This geometric proximity represents semantic similarity. Furthermore, relationships are captured by directions, enabling analogies like:

$$v_{King} - v_{Man} + v_{Woman} \approx v_{Queen}$$

2 The Core Skip-Gram Model

2.1 The Objective: Predicting the Context

The Skip-Gram model learns embeddings by training a neural network on a "fake" task. The task is: Given a central **target word** (w), predict the surrounding **context words** (c).

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

If our target word is "fox" and our window size is 2, the training pairs generated are:

- (fox, quick), (fox, brown), (fox, jumps), (fox, over)

The goal is to adjust the word vectors such that the model assigns a high probability to the words that actually appear near "fox". We don't care about the prediction task itself; we care about the internal representations the network learns in the process.

2.2 The Architecture: A Shallow Neural Network

The model is a shallow neural network with one hidden layer.

Architecture Focus: The Shallow Network Structure

Let V be the vocabulary size (e.g., 100,000) and D be the embedding dimension (e.g., 300).

1. **Input Layer:** A One-Hot Encoded vector representing the target word w. Size $(1 \times V)$.

2. **Input Weight Matrix (W):** The primary embedding matrix. Size ($V \times D$).
3. **Hidden Layer (Projection):** The result of the multiplication. Size ($1 \times D$).
4. **Output Weight Matrix (W'):** The context embedding matrix. Size ($D \times V$).
5. **Output Layer:** Produces scores for every word in the vocabulary. Size ($1 \times V$).

Important Note: Two Sets of Vectors

The model actually learns two distinct vectors for each word during training:

- $\mathbf{v_w}$: The vector when the word is the **input**/target (stored in W).
- $\mathbf{u_c}$: The vector when the word is the **output**/context (stored in W').

Typically, the input vectors (v_w) are used as the final embeddings, as they tend to capture semantics more effectively.

2.3 The "Lookup Trick": How Weights Become Vectors

A crucial insight is how the hidden layer is calculated. When a One-Hot vector is multiplied by the weight matrix W, it effectively selects the single row corresponding to the input word.

Let's visualize this with $V = 3$ (Vocabulary: [King, Queen, Man]) and $D = 2$. Input is "King" (Index 0).

$$\underbrace{[1, 0, 0]}_{\text{Input "King"}} \times \underbrace{\begin{bmatrix} \mathbf{w_{00}} & \mathbf{w_{01}} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix}}_{\text{Weight Matrix } W} = \underbrace{[\mathbf{w_{00}}, \mathbf{w_{01}}]}_{\text{Hidden Layer Output}}$$

The Hidden Layer Output **is** the word vector v_{King}. Therefore, training the weights of this network is synonymous with learning the embeddings.

3 Vanilla Skip-Gram: The Mathematics of Softmax

How do we quantify the probability of a context word c given a target word w?

3.1 Similarity as Dot Product

The similarity score between the target word w and a potential context word c is calculated using the dot product of their respective vectors (input vector for w, output vector for c):

$$\text{Score}(w, c) = v_w \cdot u_c$$

A higher dot product indicates higher similarity and thus a higher likelihood of appearing together.

3.2 The Softmax Function

To convert these raw scores into a valid probability distribution (where all probabilities sum to 1), we use the **Softmax** function.

The Math: The Softmax Probability

The probability of predicting context word c given target word w is:

$$P(c|w) = \frac{\exp(v_w \cdot u_c)}{\sum_{k=1}^{V} \exp(v_w \cdot u_k)} \tag{1}$$

- **Numerator:** Exponentiates the score of the specific pair (w, c).
- **Denominator:** The normalization term. It sums the exponentiated scores of the target word w paired with *every single word* k in the vocabulary V.

3.3 The Loss Function: Cross-Entropy

The goal of training is to maximize the probability of the actual observed context words. This is typically formulated as minimizing the negative log-likelihood (which is equivalent to the Cross-Entropy Loss in this scenario).

For a single training pair (w, c_{pos}), where c_{pos} is the true context word:

The Math: Cross-Entropy Loss

$$\begin{aligned} J(\theta) &= -\log P(c_{pos}|w) \\ &= -\log \left(\frac{\exp(v_w \cdot u_{c_{pos}})}{\sum_{k=1}^{V} \exp(v_w \cdot u_k)} \right) \\ &= -(v_w \cdot u_{c_{pos}}) + \log \left(\sum_{k=1}^{V} \exp(v_w \cdot u_k) \right) \end{aligned}$$

We use Gradient Descent to update the vectors (v_w and all u_k) to minimize this loss.

4 The Computational Crisis: The Softmax Bottleneck

The Vanilla Skip-Gram model is mathematically sound but computationally impractical for real-world applications. The culprit is the denominator of the Softmax function.

4.1 The Tyranny of the Denominator

To calculate the loss $J(\theta)$ for just **one** training pair, we must compute the normalization term:

$$\sum_{k=1}^{V} \exp(v_w \cdot u_k)$$

If the vocabulary size $V = 100,000$, this requires 100,000 dot product calculations and exponentiations for every single training example in the corpus (which might contain billions of words).

4.2 The Gradient Nightmare

The problem is exacerbated during training (backpropagation). When we calculate the gradient of the loss function to update the weights, the derivative of the normalization term also requires summing over the entire vocabulary. This means that for every single training step, we are calculating gradients and updating the vectors for nearly all words in the vocabulary (the entire

output matrix W'). This computational complexity makes Vanilla Skip-Gram prohibitively slow.

The Intuition: The Security Guard Analogy (The Bottleneck)

Imagine a security guard (the model) checking tickets at a massive stadium (the vocabulary).

- **The Softmax Approach (The Exhaustive Check):** To validate *one* person's ticket (the correct context word c_{pos}), the guard must first check the ID of **every single person in the stadium** (the denominator, V) to ensure the probabilities are correctly normalized relative to everyone else.

This is incredibly inefficient. We need a faster approximation.

5 The Solution: Skip-Gram with Negative Sampling (SGNS)

Negative Sampling (SGNS) provides an elegant optimization by redefining the learning objective entirely. It is an approximation technique related to Noise Contrastive Estimation (NCE), approximating the full Softmax normalization using a small sample of noise.

5.1 The Shift: From Multi-Class to Binary Classification

We stop asking the V-way classification question:

- *Old Question (Softmax):* "Given w, which word out of the 100,000 possibilities is the context word?"

And start asking a binary classification question:

- *New Question (SGNS):* "Is this specific pair (w, c) real or fake?"

5.2 The New Training Strategy

We generate a new dataset for this binary task. For each actual training pair (the **Positive Sample**), we generate K artificial noise pairs (the **Negative Samples**). K is typically between 5 and 20.

Example: Target="King", Context="Rules". K=2.

- **Positive Pair ($D = 1$):** ("King", "Rules"). Goal: Output 1 (Real).
- **Negative Pair 1 ($D = 0$):** ("King", "Table"). Goal: Output 0 (Fake).
- **Negative Pair 2 ($D = 0$):** ("King", "Water"). Goal: Output 0 (Fake).

The Intuition: The Security Guard Analogy (Optimized)

- **The Negative Sampling Approach (The Spot Check):** The guard checks the ticket of the person entering (Positive Sample). Then, instead of checking everyone in the stadium, the guard glances at K (e.g., 5) random people nearby (Negative Samples) and confirms they *don't* have the ticket.

This reduces the computation from V (100,000) calculations to just $K + 1$ (e.g., 6) calculations per training step. Crucially, we only need to update the $K + 1$ context

vectors involved, not the entire vocabulary.

6 SGNS: Mathematical Formulation

Now we adapt the mathematics for this new binary objective.

6.1 The Sigmoid Function

Since we are now dealing with binary classification (Real vs. Fake), we replace Softmax with the **Sigmoid** function $\sigma(x)$. It squashes any real number (like the dot product score) into the range [0, 1].

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

We use this to model the probability that a pair (w, c) is "Real" ($D = 1$).

The Math: Binary Probabilities

- Probability the pair is Real: $P(D = 1|w, c) = \sigma(v_w \cdot u_c)$
- Probability the pair is Fake: $P(D = 0|w, c) = 1 - \sigma(v_w \cdot u_c)$

A useful mathematical identity of the Sigmoid function is its symmetry: $1-\sigma(x) = \sigma(-x)$. Therefore:

- Probability the pair is Fake: $P(D = 0|w, c) = \sigma(-v_w \cdot u_c)$

6.2 The New Objective Function

We want to simultaneously maximize the likelihood of the positive sample being identified as real AND the likelihood of the K negative samples being identified as fake. We maximize the joint probability, which in log space becomes a summation.

The Math: The SGNS Objective Function (Log-Likelihood)

For a target word w, its true context c_{pos}, and K negative samples u_k:

$$J(\theta) = \underbrace{\log \sigma(v_w \cdot u_{c_{pos}})}_{\text{Maximize Positive}} + \underbrace{\sum_{k=1}^{K} \log \sigma(-v_w \cdot u_k)}_{\text{Maximize Negative (as Fake)}}$$

This is the function we aim to maximize using Gradient Ascent.

7 Gradient Descent: Derivation and Dynamics

To maximize $J(\theta)$, we need to calculate the gradients with respect to the vectors involved (e.g., v_w).

We need a key derivative identity for the log-sigmoid function:

$$\frac{\partial}{\partial x} \log \sigma(x) = 1 - \sigma(x)$$

7.1 Case 1: The Positive Pair Gradient

We calculate the derivative of the first term of $J(\theta)$ with respect to v_w. We apply the chain rule (derivative of the outside $\log \sigma(\cdot)$ times the derivative of the inside $v_w \cdot u_{c_{pos}}$):

$$\begin{aligned}\frac{\partial J}{\partial v_w} &= (1 - \sigma(v_w \cdot u_{c_{pos}})) \cdot \frac{\partial}{\partial v_w}(v_w \cdot u_{c_{pos}}) \\ &= \underbrace{(1 - \sigma(v_w \cdot u_{c_{pos}}))}_{\text{Error Term}} \cdot u_{c_{pos}}\end{aligned}$$

Interpretation: The error term is (1 − Prediction). If the model predicted 1 (perfect), the error is 0, and the gradient is 0. If the model predicted 0.5, the error is 0.5, and we update v_w in the direction of $u_{c_{pos}}$.

7.2 Case 2: The Negative Pair Gradient

We calculate the derivative of one term in the summation:

$$\begin{aligned}\frac{\partial J}{\partial v_w} &= (1 - \sigma(-v_w \cdot u_{c_{neg}})) \cdot \frac{\partial}{\partial v_w}(-v_w \cdot u_{c_{neg}}) \\ &= (1 - \sigma(-v_w \cdot u_{c_{neg}})) \cdot (-u_{c_{neg}})\end{aligned}$$

Using the symmetry identity $1 - \sigma(-x) = \sigma(x)$:

$$\begin{aligned}&= \sigma(v_w \cdot u_{c_{neg}}) \cdot (-u_{c_{neg}}) \\ &= - \underbrace{\sigma(v_w \cdot u_{c_{neg}})}_{\text{Probability of Mistake}} \cdot u_{c_{neg}}\end{aligned}$$

Interpretation: The error term here is the model's prediction that the negative sample is real. If the model predicted 0 (perfectly identified as fake), the gradient is 0. If the model mistakenly predicted 0.5 (thought it might be real), we update v_w in the direction opposite to $u_{c_{neg}}$.

7.3 The Update Rules: Push and Pull Dynamics

We use Stochastic Gradient Ascent to update the vectors: $v_{new} = v_{old} + \alpha \cdot \nabla J$, where α is the learning rate. The elegance of the SGNS derivation leads to a beautiful geometric interpretation of how the vectors learn.

The Intuition: Geometric Interpretation: Attraction and Repulsion

The training process creates forces that shape the semantic space.

1. The Positive Pull (Attraction):

$$v_w \leftarrow v_w + \alpha \cdot (1 - \sigma(v_w \cdot u_{c_{pos}})) \cdot u_{c_{pos}}$$

We are adding a fraction of the context vector $u_{c_{pos}}$ to the target vector v_w. This action physically pulls v_w closer to $u_{c_{pos}}$. Words that appear together attract each other.

2. The Negative Push (Repulsion):

$$v_w \leftarrow v_w - \alpha \cdot \sigma(v_w \cdot u_{c_{neg}}) \cdot u_{c_{neg}}$$

We are subtracting a fraction of the negative vector $u_{c_{neg}}$ from the target vector v_w. This action pushes v_w away from $u_{c_{neg}}$. Words that do not appear together (in this sampled instance) repel each other.

Through billions of iterations of these pushes and pulls, the vectors settle into an equilibrium where the geometric arrangement reflects the underlying semantics of the language.

8 Advanced Mechanics and Implementation Details

Several practical optimizations are crucial for high-quality embeddings.

8.1 How to Choose Negative Samples? (The Noise Distribution)

Negative words must be drawn from a "Noise Distribution" $P_n(w)$. Using the raw unigram distribution (frequency in the corpus) leads to very frequent words (like "the", "is") dominating the samples, which is suboptimal. Mikolov et al. found that raising the unigram distribution $U(w)$ to the power of $3/4$ works best empirically:

$$P_n(w) = \frac{[U(w)]^{3/4}}{\sum_k [U(k)]^{3/4}}$$

This heuristic acts as a smoothing factor, slightly increasing the sampling probability of rare words and decreasing it for frequent words.

8.2 Subsampling Frequent Words

Words like "the" and "is" appear frequently but carry less semantic information. For example, ("France", "the") is less informative than ("France", "Paris").

To counter this, Skip-Gram implements **Subsampling**. Frequent words are randomly discarded from the training data before generating pairs, with a probability $P(w_i)$:

$$P(\text{discard } w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Where t is a threshold (e.g., 10^{-5}) and $f(w_i)$ is the frequency of the word. This speeds up training and improves the quality of embeddings for rarer words.

8.3 Handling Window Size > 1

In practice, we use a window (e.g., size 2), yielding multiple context words (e.g., c_1, c_2, c_3, c_4) for a center word w.

In Skip-Gram, we do **not** average the context vectors (that is the CBOW approach). Instead, we treat each pair independently and sum their objectives (or equivalently, sum their gradients).

$$J_{total} = J(w, c_1) + J(w, c_2) + J(w, c_3) + J(w, c_4)$$

The vector v_w is simultaneously pulled in multiple directions, eventually settling near the geometric center (centroid) of its neighbors.

9 Numerical Example: End-to-End SGNS

Let's walk through a single step of training using SGNS with concrete numbers. It is crucial to remember that both the center vector (v) and the context vectors (u) are updated in each step.

Numerical Walkthrough: Training the Vector for "King"

Scenario:

- Target Word (w): "King"

- Positive Context (c_{pos}): "Rules"
- Negative Sample (c_{neg}): "Table" (Assuming K=1 for simplicity)
- Learning Rate (α): 0.1
- Embedding Dimension (D): 2

1. Initial Vectors (Initialized Randomly):

- $v_{king} = [0.10, 0.30]$
- $u_{rules} = [0.20, 0.90]$
- $u_{table} = [0.70, 0.10]$

2. Forward Pass (Dot Products & Sigmoid): We calculate the scores and the probabilities (predictions).
Positive Pair ("King", "Rules"):

$$\text{Score}_{pos} = v_{king} \cdot u_{rules} = 0.1(0.2) + 0.3(0.9) = 0.29$$

$$P_{pos} = \sigma(0.29) \approx \mathbf{0.57}$$

(Goal is 1.0).
Negative Pair ("King", "Table"):

$$\text{Score}_{neg} = v_{king} \cdot u_{table} = 0.1(0.7) + 0.3(0.1) = 0.10$$

$$P_{neg} = \sigma(0.10) \approx \mathbf{0.52}$$

(Goal is 0.0. The model is performing poorly, thinking this fake pair is likely real).
3. Calculate Error Scalars: We calculate the scaling factor for the gradients based on the derivations in Section 7.

- Pos Error Scalar (Error Term): $E_{pos} = (1 - P_{pos}) = (1 - 0.57) = \mathbf{0.43}$
- Neg Error Scalar (Prob. of Mistake): $E_{neg} = P_{neg} = \mathbf{0.52}$

4. Calculate Gradients for v_{king}: We calculate the contribution of the positive pull and the negative push.
Gradient from "Rules" (The Pull): $E_{pos} \cdot u_{rules}$

$$\nabla_{pos} = 0.43 \times [0.20, 0.90] = [0.086, 0.387]$$

Gradient from "Table" (The Push): $-E_{neg} \cdot u_{table}$

$$\nabla_{neg} = -0.52 \times [0.70, 0.10] = [-0.364, -0.052]$$

Net Gradient $\nabla J_{v_{king}}$*:* (Pull + Push)

$$\nabla J = [0.086 - 0.364, \quad 0.387 - 0.052] = [-0.278, 0.335]$$

5. Update v_{king} (Gradient Ascent):

$$v_{king}^{new} = v_{king}^{old} + \alpha \cdot \nabla J_{v_{king}}$$

$$v_{king}^{new} = [0.10, 0.30] + 0.1 \times [-0.278, 0.335]$$

$$v_{king}^{new} = [\mathbf{0.0722}, \mathbf{0.3335}]$$

6. Update Context Vectors (Crucial Step!) We must also update the output vectors u. The gradients are symmetric to the ones calculated in Step 4. We use the same error scalars but multiply by the input vector v_{king} (using the original v_{king} for this step).

Update u_{rules} (Attraction):

$$u_{rules}^{new} = u_{rules}^{old} + \alpha \cdot (E_{pos} \times v_{king})$$

$$u_{rules}^{new} = [0.20, 0.90] + 0.1 \times (0.43 \times [0.10, 0.30])$$

$$u_{rules}^{new} = [0.20, 0.90] + 0.1 \times [0.043, 0.129] = [\mathbf{0.2043}, \mathbf{0.9129}]$$

Update u_{table} (Repulsion):

$$u_{table}^{new} = u_{table}^{old} - \alpha \cdot (E_{neg} \times v_{king})$$

$$u_{table}^{new} = [0.70, 0.10] - 0.1 \times (0.52 \times [0.10, 0.30])$$

$$u_{table}^{new} = [0.70, 0.10] - 0.1 \times [0.052, 0.156] = [\mathbf{0.6948}, \mathbf{0.0844}]$$

Result Analysis: v_{king} adjusted its position: the Y-coordinate increased (towards u_{rules} at Y=0.9) and the X-coordinate decreased (away from u_{table} at X=0.7). The context vectors also adjusted slightly to reflect this interaction. The system is learning.

10 Conclusion: The Emergence of Meaning

The journey of the Skip-Gram model highlights a crucial pattern in machine learning: starting with a mathematically pure but computationally expensive objective (Softmax) and finding efficient approximations (Negative Sampling) to make it feasible at scale.

By defining a simple, efficient "fake task"—distinguishing real word pairs from randomly generated noise—we can train a shallow neural network whose internal weights organize themselves into a high-dimensional semantic space. This process transforms the abstract concept of linguistic meaning into concrete geometric relationships (distances and angles), providing the foundational numerical representations that power modern Natural Language Processing.