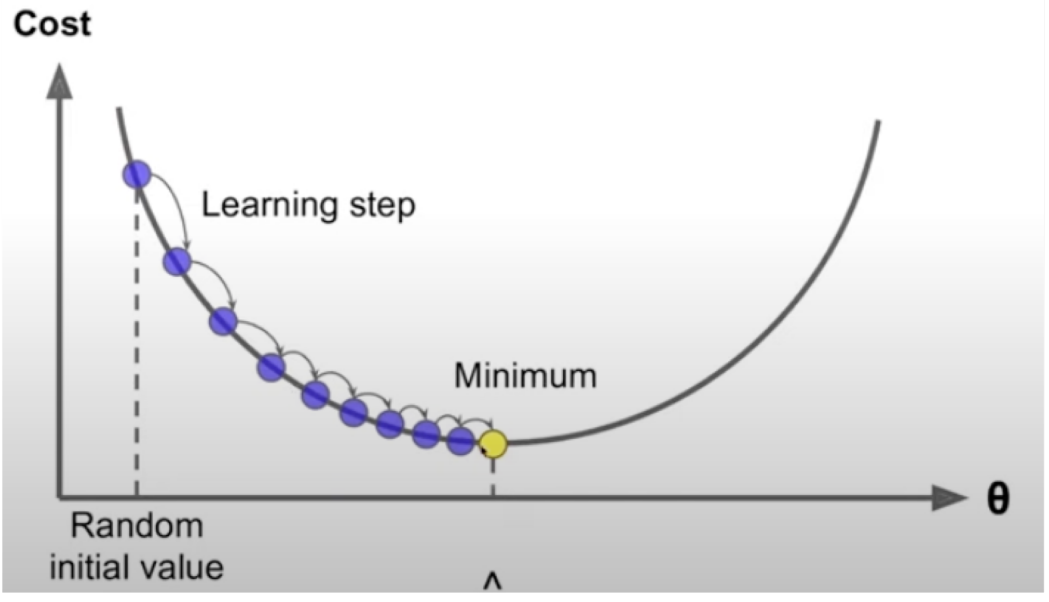
Bag of words

* They don't pay attention to word order or position This is a model makes the some predictions at each position

⇒ word2vec maximizes objective func by putting similar words nearby in space.

Optimization : Gradient Descent

* We want to minimize $J(\theta)$ (cost func) to learn good vectors. Gradient descent is an algo. that minimize $J(\theta)$ by changing θ.
* Idea is from current value of θ calculate gradient of $J(\theta)$, then take a small step in the direction of negative gradient and repeat.



* we need to find optimum learning step.

* Update equation : for a single parameter

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

α ↳ learning rate / step size

NOTE: This one is simple GD algo. No one use this now. It is very expensive to compute

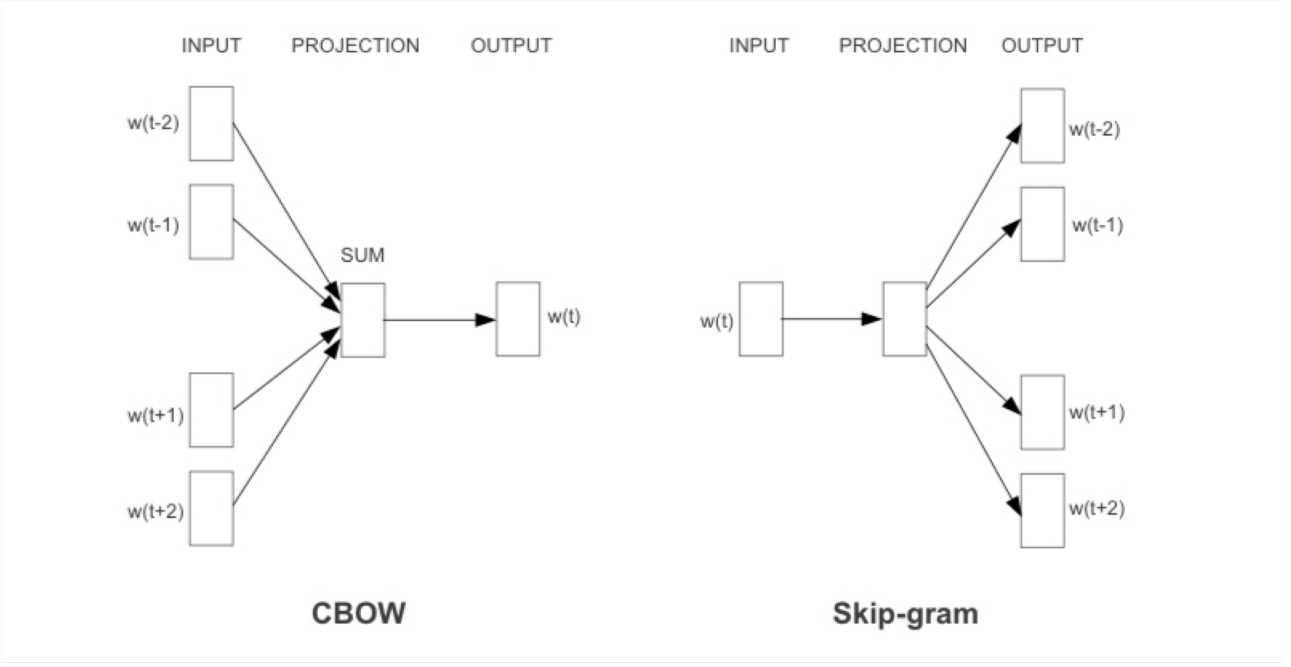
Stochastic Gradient Descent

* Rather than estimating gradient based on entire corpus we simply take one center word or a small batch like 32 center words and we work out an estimate of the gradient descent based on them
* SGD is fast also it perform better solutions on complex networks

Word2Vec Alghoritm Family

1. Skip Gram Model : Predict the context ("outside") words (position independent) given center word. Increasing the range (maximum distance of words) improve quality of resulting word vectors, but it is also increases the computational complexity

2. Continuous Bag of Words: Predict the center word from context words



The SkipGram model with Negative Sampling

$$P(O|C) = \frac{\exp(v_O^T v_C)}{\sum_{w \in V} \exp(v_w^T v_C)}$$

⇒ The idea of negative sampling is instead of using softmax we are going to train binary logistic regression models for both the true pair of center word and the context word versus noise pairs where we keep the true center word and just randomly sample words from the vocab.

$$J_t(\theta) = \overbrace{\log \sigma(u_o^T v_c)}^{1} + \overbrace{\sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma(-u_j^T v_c) \right]}^{2}$$

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

* We want to maximize the probability of two words co-occuring in first log (1) and minimize probability of noise words (2). We add minus side inside of sigmoid function because of that.
* It is going to work with k random words.

Co-occurrence vectors

* Negative sides of simple count co-occurrence vectors
  1. Vectors increase in size with vocab.
  2. Very high dimensional, requires a lot of storage
  3. Subsequent classification models have sparsity issues (models are less robust)

Because of these issues we can get better results working with low-dimensional vectors

* Idea is store "most" of the important info in a fixed small number of dimensions → dense vector
* Usually 25 - 1000 dimensionality ⇒ We need to reduce that.

Dimensionality Reduction : SVD

* It can be use for reducing the co-occurrence matrix dimensionality. It is expensive to compute for large matrices

* Running SVD on raw counts does not work well. So; Scaling the count in the cells help a lot

PROBLEM: "The, he, has" like function words are to frequent.

SOLUTION: Log the frequencies
Ignore these words

Q) How can we captures ratio of co-occurance probabilities as linear meaning components in a word vector space?

Log-bilinear model : $w_i \cdot w_j = \log P(i|j)$

⇒ Dot product between two word vectors attempts to approximate the log of the probability of co-occurrence

with vector differences : $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

⇒ Difference between two word vectors similarity to another word corresponds to the log of the probability ratio

Glove

* Fast training
* Scalable to huge corpora
* Good performance even with small corpus and small vectors.
* For its training phase instead of continuously iterating over local windows of sequenced data we use the co-occurrence matrix as a lookup table for words

which have appeared in the context of other words, as well as prevent computation for words who have no co-occurrence