Stanford CS224N NLP with Deep Learning

Winter 2021

Lecture 2 – Neural Classifiers

Word2Vec Reviewed

As we have seen, Word2Vec starts with random word vectors, iterates through each word in the corpus, and learns vectors that maximises probability of the context words occurring.

The parameters of this model are only the word vectors of all the words.

This is called a *bag of words*, in which no attention is paid to the position of the word. All words in the window are treated equally.

Word2Vec maximises the objective function by putting similar words' vectors near to each other in space.

Gradient Descent

To learn good vectors, we have a cost function $J(\theta)$ which we want to minimise. Gradient descent enables us to do this. The idea is to take a small step in the direction in which the gradient of $J(\theta)$ decreases the fastest.

Mathematically,

$$\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_\theta J(\theta),$$

where α, the step size, is called the *learning rate.* Note that this is in matrix notation; for a single component, the formula is

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

However, since J is a function of all the words in the corpus, this takes an impractically long time. This is solved by a concept called *stochastic gradient descent.* Here, we make updates based on a single sample window (or a small batch of sample windows) in the corpus.
Note, however, that when we do this for word vectors, the number of actual parameters being updated would be very small compared to the size of θ (as windows are very small compared to the corpus). To solve this, we need efficient sparse matrix operations.

Variants on Word2Vec

There are two basic model variants on Word2Vec.
The Skip-Gram (SG) model predicts the context words given the centre word, independent of position; it is what we have studied so far.
The Continuous Bag of Words (CBOW) model predicts the centre word given a bag of context words.

Negative Sampling

Additional efficiency in training is given by negative sampling (NS) in place of the softmax function, which helps us avoid computing the normalisation term $\sum_{w \in V} \exp(u_w \cdot v_c)$.
The idea is to train binary logistic regression models for the true pair versus several noise pairs (the centre word paired with a random word). The objective function to be maximised in this case is

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta),$$

where

$$J_t(\theta) = \log \sigma(u_0 \cdot v_c) + \sum_{i=1}^{k} E_{j \sim P(w)} [\log \sigma(-u_j \cdot v_c)]$$

and t varies over all the windows in the corpus.
In more familiar notation,

$$J_{\text{neg-sample}}(u_0, v_c, U) = -\log \sigma(u_o^T v_c) - \sum_{k \in \{K \text{sampled indices}\}} \log \sigma(-u_k^T v_c),$$

which means that we maximise the probability that a real outside word appears, and minimise the probability that the k random samples occur, around the centre word.
The sampling is done with

$$P(w) = \frac{U(w)^{\frac{3}{4}}}{Z},$$

where $U(w)$ is the unigram distribution. This makes the less frequent words come up more often (the $\frac{3}{4}$ has the effect of dampening the disparity in probabilities).

Latent Semantic Analysis

A more simple, naïve way of defining a word vector is using a co-occurrence matrix. We make a symmetric table counting how many times each pair of words occurs in each other's context (within a window length).
Then we can consider the vector of a word as the corresponding row or column of this matrix.

A problem with this is that the size of the vectors is the same as the vocabulary size, and this can become very high. This takes up a lot of storage and leads to sparsity issues, which makes models less robust.
Thus, we try to store important information in a fixed, small number of dimensions. The dimensionality can be reduced by the singular value decomposition method.
Using this method, X is decomposed into $U\Sigma V^T$, where Σ is diagonal. However, the singular values in it are ordered from largest to smallest, and we can successively delete the smallest singular values to get back a "compressed" version of X.

However, running an SVD on raw counts doesn't work well, since some words are extremely frequent (like function words). Some possible fixes are:

- take the logarithm of the frequencies
- take $\min(X, t)$, with $t = 100$ or so
- ignore function words entirely

We could also use windows that count differently based on distance (ramped windows).

Encoding Meaning Components

We wish to have a way to define vectors so that it at least approximately works out that, say,

$$v_{\text{teacher}} - v_{\text{teach}} = v_{\text{student}} - v_{\text{learn}}.$$

A method that enabled (GloVe) this used the insight that ratios of co-occurrence probabilities could encode components.

How do we capture this as linear meaning components? We could use a log-bilinear model, where

$$w_i \cdot w_j = \log P(i \mid j).$$

The objective function in this model is

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i \cdot \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij}) \right)^2,$$

where f is a function that tops out at high frequencies similar to $\min(x, t)$.

Evaluation of Word Vectors

As in the general case of evaluation in NLP, there are two ways to evaluate word vectors – intrinsic and extrinsic.

Intrinsic evaluation involves evaluation on a specific subtask, and is fast to compute. It helps to understand that system, but it may not be helpful unless

the correlation to the real task is established.
Examples of this include testing analogies and similarities.

Extrinsic evaluation, on the other hand, is carried out on a real task, and can take a long time to compute. It is unclear if the subsystem is the problem or its interaction with other subsystems.
One example of this is named entity recognition.

Word Sense Ambiguity

What effect does multiplicity of senses have on word vectors? Why do we have only one word vector for words with have more than one sense?

In fact, in standard word embeddings like Word2Vec, different senses of a word reside in a linear superposition; for example,

$$v_{\text{bank}} = \alpha_1 v_{\text{bank}_1} + \alpha_2 v_{\text{bank}_2},$$

where

$$\alpha_i = \frac{f_i}{\sum_i f_i},$$

with f_i being the frequency of sense i.