Language Models

Goals:

1. Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

2. Related Task: Probability of an upcoming word:

$$P(w_5|w_1, w_2, w_3, w_4)$$

A model that computes either of the above two tasks is called a language model or LM.

How to compute Joint Probability(P(W))

Chain Rule of Probability

Recall the definition of conditional probabilities

$$P(B|A) = P(A, B)/P(A)$$

Rewriting:

$$P(A, B) = P(A)P(B|A)$$

For More variables:

$$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$$

The Chain Rule in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1, \dots, x_{n-1})$$

So, Probability of words in a sentence

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i|w_1 w_2 \dots w_{i-1})$$

Note:

$$w_1 w_2 \dots w_{i-1} \text{ often referred as a prefix.}$$

For eg .

$$P(\text{The quick brown fox}) = P(the) * P(quick|the) * P(brown|\text{the quick}) * P(fox|\text{the quick brown})$$

Now to estimate the probability of word "jump" given the prefix("the quick brown fox")

$$P(jump|\text{the quick brown fox}) = \frac{Count(\text{the quick brown fox jumps})}{\text{Count(the quick brown fox)}}$$

Problems we will encounter:

1. Too many possible sentences.
2. We will never see enough data for estimating these.

We can simplify the above calculation using Markov Assumption by modelling it as an n-gram model:

1. Unigram Model(where n=1) where 0th order Markov Assumptions is made which means probability of a word is independent of all other previous words.

$$P(jump|\text{the quick brown fox}) = P(jump)$$

2. Bigram Model(when n=2) where 1st order Markov Assumptions is made which means probability of a word depends on previous 1 word.

$$P(jump|\text{the quick brown fox}) = P(jump|fox)$$

2. Trigram Model (when n=3) where 2nd order Markov Assumption is made which means probability of a word depends on previous 2 words.

$$P(jump|\text{the quick brown fox}) = P(jump|brownfox)$$

Note:

An n-grams model makes order n-1 Markov assumption. This assumption implies: given the previous n-1 words, probability of (nth) word is independent of words prior to (n-1) words.

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i / w_{i-k} .. w_{i-1}) \text{ where k is kth order Markov Assumption.}$$

In other words, we approximate each component in the product.

$$P(w_i|w_1, w_2 \dots w_{i-1}) \approx \prod_i P(w_i|w_{i-k} .. w_{i-1})$$

Estimating Bigram Probabilities

$$\text{We use Maximum Likelihood Estimate to calculate } P(w_i|w_{i-1})$$

In this case, Maximum likelihood estimate is just the relative frequency based on the empirical counts on a training set.

$$P(w_i|w_{i-1}) = \frac{count(w_i, w_{i-1})}{count(w_{i-1})}$$

Problems with N-gram Language Model:

1. Bigram Count Table could be big and sparse.Model size is O(exp(n)).
2. Long distance dependencies can not be modelled.
3. Use log to handle underflow problem i.e Sentence probability can be very tiny for longer inputs with infrequent words as they will have very smaller probability of occurrence.
4. In case of out of vocabulary in train data[i.e No count in Bigram Count Table] ,Probability of occurrence of the word is 0,making the whole Probability of the sentence to be 0, in such case we use different smoothing techniques. For eg. One such technique is to steal probability mass from frequent words and assign them equally to 0 occurring words.

5. In case of out of vocabulary words in test data, try adding a reserved word like UNK in the training set to handle them.
6. All words are represented through one-hot vector as a result all words are equally (dis)similar! and lacks representation space in which words, phrases, sentences etc. that are semantically similar also have similar representations.

Evaluation of Language Models

1. A good language model should assign a high probability to held out text.

Metric to Evaluate Language Models

Perplexty

1. It is the inverse probability of the test set ,normalised by the number of words.
2. Minimising perplexity is the same as maximising probability.
3. Intuition to Perplexity - Perplexity is weighted equivalent branching factor. On average,how many things can occur next.

$$PP(w) = P(w_1w_2w_3w_4..w_n)^{-1/N} \text{ where N - number of words}$$

or

$$\sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_1w_2w_3...w_n)}}$$

Applying Chain Rule:

$$\sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_1/w_{i-1}...w_1)}}$$

For bigram :

$$\sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_1/w_{i-1})}} \text{ where N - number of words}$$

In practice:

Perplexity is the exponentiated **token-level negative log-likelihood**.

$$PP(W) = \exp(-\frac{1}{N}\sum_{i}^{N} logp(w_i|w_{<i})$$

where

$$-\frac{1}{N}\sum_{i}^{N} logp(w_i|w_{<i}) \text{ --- token-level negative log-likelihood}$$

Neural Language Model

1. Represent prefix words with low-dimensional vectors called embeddings.
2. Use of Softmax to generate probability distribution over the entire vocabulary.

3. Use of a composition function which takes the input "sequence of word embeddings corresponding to the tokens of a given prefix " and generates and output of "single vector".Different ways we can model this composition function:
    1. Element-wise functions e.g., just sum up all of the word embeddings!
    2. Concatenation
    3. Feed-forward neural networks
    4. Convolutional neural networks
    5. Recurrent neural networks
    6. Transformers

Concatenation based Approach

Fixed Window Neural Language Model

1. Choose a fixed window size of word prefix.
2. Concatenate word embeddings to form the input representation of the prefix words.
3. Bunch of Linear Layers to apply non linear transformations.
4. Output layer with Softmax to generate output distribution.

Improvements over n-gram LM:

1. No sparsity problem
2. Model size is O(n) not O(exp(n))

Existing Problems

1. Fixed window is too small
2. Enlarging window enlarges Weight Matrix[W]
3. Window can never be large enough!
4. We don’t share weights across the window.