**Natural Language Processing (NLP)**

**Session 3**

1. **Sequence modelling**:
   1. *Language models*: displays the likelihood of different sequences (of words).  
        
      Predicts the probability function P(current word wi given the history/context w1, w2, w3, … wi-1)
      1. For each context it gives a discrete probability distribution over all words in the vocabulary for wi.
      2. It assigns a probability value for a given word observed at position wi given the context observed at w1 … wi-1.
   2. *N-gram models*:
      1. The probability of the next word being a given value, (e.g. ‘loves’) independent of the previous words is the unigram probability. Calculated as:  
           
         P(wi = loves) = Frequency of the word (unigram) / the summed frequencies of all the words in the vocabi.e. the length of the training data  
           
          This is a bag-of-words model and doesn’t consider the word order.
      2. Using the probability of the next word given the previous one i.e. the conditional probability p(wi given wi-1) (e.g. for ‘john loves’) is the bigram probability. In event terms:  
           
         P(wi = loves given wi-1 = john) = Frequency of n-gram (bigram) / Frequency of context (previous word from position i)
      3. General N-gram: After training a Maximum Likelihood Estimation (MLE) n-gram model from counting function C from a corpus:  
           
         P(wi-n+1 given wi-1) = Frequency of n-gram / Frequency of (previous n-1 words from word in position i)
      4. **Chain rule**:   
           
         We can rearrange the formula for conditional probability to get the so-called *product rule*:

P(A intersect B) = P(A given B) times P(B)

We can extend this for three variables:

P(A intersect B intersect C) = P(A given B intersect C) times P(B intersect C) = P(A given B intersect C) times P(B given C) times P(C)

And, in general, to *n* variables:

P(A1, A2, ..., An) = P(A1 given A2, ..., An) times P(A2 given A3, ..., An) times P(An-1 given An) times P(An)

Referred to this as the *chain rule*.

* + 1. **Markov assumption**:
       1. Probability of next word only depends on a fixed number of words back.
       2. Instead of checking the full word history, approximate by using the “n-gram model of length k” (where k = n-1). E.g., a bigram model only depends on the previous word, a trigram model depends on the previous two words only, 4-gram model depends on the previous 3 words only, etc.
    2. *General method*:
       1. When processing sequences extract the relevant n-grams (word sequences) according to the value of *n*.
       2. In training count the frequency of the n-grams occurring in the training data and store the counts.
       3. In testing use those counts to get probabilities of sequences of unseen data.
       4. Deriving the probabilities can be done with a variety of methods, called n-gram language models.
  1. **Evaluation**:
     1. Gather a corpus.
     2. Divide it into 3 standard sections: Training data, held-out data, and test data.
     3. Gather all the counts/estimations from the training data
     4. Iteratively develop by assigning probability to the heldout (not the test!) data.
     5. Experiment with value of n and other parameters like discounts (more later).
     6. Get the Perplexity score on the Test data (measure of how confused the model is by the unseen corpus).
     7. As it’s a sequence modelling task, unlike bag-of-words based classification, we shouldn’t just remove unknown or ‘out of vocabulary’ (OOV)/unknown words, leads to ungrammatical sequences.
     8. Several approaches to OOV words:
        1. Define the vocab by stipulating a minimum document frequency for words in the training data (e.g. 2). Any words appearing less than that, replace with an unknown word token **</unk>** in the training data.
        2. Define the vocab by setting some held out data aside- any words appearing in that which are not in the main training data are defined as OOV- replace their occurrences with </unk> in the training data.
     9. On test data, always replace all OOV words with </unk>.
  2. **Perplexity**:
     1. The best language model is one that best predicts an unseen test data *W*, i.e., the one that gives the highest probability for those sentences.
     2. Perplexity is the inverse probability of the test set, normalised by the number of words: Minimising perplexity is the same as maximising probability.
     3. *Cross-entropy*: Cross-entropy is another metric used for evaluating the confusion of the language model on a test corpus.  
          
        Calculated as the negative sum of the log probabilities divided by the length of the corpus.  
          
        Perplexity can be simply calculated from cross entropy by whatever log base you’re using to the power of the cross-entropy.  
          
        So, minimising cross-entropy is also the same as maximising probability.
  3. **Overfitting**:
     1. Need to train robust models that generalise.
     2. Three main approaches:
        1. Smoothing
           1. Hold back some probability mass for unseen events
        2. Backoff & Interpolation
           1. Estimate n-gram probability from (n-1)-gram probability
        3. Class-based models
           1. Group words together, estimate class n-gram probability
     3. *Laplace smoothing/Add-1 smoothing*:
        1. Pretend we saw each word one more time than we did.
        2. Add one to all the counts  
             
           1. Maximum Likelihood Estimation (MLE):   
           P(word wi given wi-1) = count(wi-1 intersect wi) / count(wi-1)  
             
           2. Add-k estimate:

P(word wi given wi-1) = count(wi-1 intersect wi) + 1 / count(wi-1) + vocabulary size *V*.  
  
Add-1 estimation is a blunt instrument- too much mass given to unseen events, some seen events reduced too much.

* + 1. *Generalized additive smoothing / Add-k smoothing*:
       1. Pretend we saw each word a value k, (0,1], more than we did.  
            
          1. Maximum Likelihood Estimation (MLE):   
          Same as Add-1 smoothing  
            
          2. Add-k estimate:

P(word wi given wi-1) = count(wi-1 intersect wi) + *k* / count(wi-1) + vocabulary size *V*.  
  
Search for k that gives largest probability to the held-out data, using an optimisation/gradient descent method to find these efficiently.

* + 1. *Backoff and interpolation*:
       1. Use a held-out corpus to get the right lambda
       2. Choose lambda to maximize the probability of held-out data:
          1. Fix the N-gram probabilities (on the training data)
          2. Then search for lambda that give largest probability to held-out set
          3. Again, you could use an optimisation/gradient descent method to find these efficiently.
    2. *Kneser-Ney Smoothing*:
       1. For the lower orders in the back-off, instead of probability of occurrence (a function of token frequency), use the probability of a word being a continuation in terms of how many types of the word appears in as a continuation.  
            
          e.g., For a tri-gram model, you can use a standard tri-gram model with a fixed absolute discount D for the main model.

1. **Summary**:
   1. Language models offer a way to assign a probability to a sentence or other sequence of words, and to predict a word from preceding words.
   2. n-gram models are Markov models that estimate words from a fixed window of previous words. n-gram probabilities can be estimated by counting in a corpus and normalizing (the maximum likelihood estimate).
   3. n-gram language models are evaluated extrinsically in some tasks, or intrinsically using perplexity.
   4. The perplexity of a test set according to a language model is the geometric mean of the inverse test set probability computed by the model.
   5. Smoothing algorithms provide a more sophisticated way to estimate the probability of n-grams. Commonly used smoothing algorithms for n-grams rely on lower order n-gram counts through backoff or interpolation.
   6. Both backoff and interpolation use discounting to create probability distributions over different orders.