Summary of Neural Probabilistic Language Model

Before the introduction of neural language model by Benjio and his colleagues on 2003, language modeling though an interesting topic to explore, had had many limitations and above all was computationally expensive. To begin with, language modelling is a technique that involves the probabilistic prediction of next word, given the sequence of words. If we let the sequence of words represent by 'n', where n is the number of words in a sequence, we define the process by the size of n i.e. unigram, bigram, trigram and so on [1]. This method i.e. the probability of calculating the next word given the particular sequence, though laid the foundation work, suffered from data sparsity, curse of dimensionality and high computational cost resulting from disregard to the syntactic and semantic similarity between words. For example, the cost to calculate the probability of a word given the sequence of n = 10 and vocabulary size V of 100000 is 100000^10 -1 [2]. Moreover, when we are calculating the probability of a word given a sequence of size n, and the word if is not present in the vocabulary, the model returns 0, which does not do justice. The same situation arises when we have a sequence that is not present in the vocabulary. Although we can use **smoothing** i.e. adding the small value delta, and **back-off process** – reducing the size of sequence i.e. n - to overcome these issues, it is nonetheless a time-consuming as well as computationally expensive model.

Therefore, *Benjio et al.* in 2003 introduced the concept of Neural Language Model focusing on taking into consideration: the context father than one or two words, and the similarity between words. The authors proposed a method to calculate the distributed representation of words, and simultaneously compute the probability function for word sequences. They came up with a noble approach to have the feature vector represent the different aspects of the word i.e. give every word a point in the vector space, which improved the computation alone over the one-hot vector model exponentially. Once they represent the words in the vector form, they concatenate those vectors and project it to the weighted matrix W with a dimension H * M, where H is the number of hidden states, and M is the total number of features in the concatenated matrix. They apply the activation function to the resulting matrix, which then is projected to another weighted matrix, where softmax is implemented to obtain the desired output.

Taking two input words, if we are to predict another word in a sequence, the model goes through the following steps [3]:

$$x' = e(x1) \oplus e(x2)$$

$$h = tanh(W1x' + b)$$

$$y = softmax(W2h)$$

Here $\oplus$ represents concatenation, whereas x1 and x2 are the two input words. W1 and W2 are the weighted matrixes, b is the bias, and h represents the matrix from the activation function *tanh*.

The main objective of this model is to predict the word while minimizing the error as any other algorithms. The authors use Negative Log Likelihood to calculate the loss function, and use back propagation algorithm to update the parameters of the weighted matrix, unless the loss function converges at some point. Since the model uses the backpropagation to learn the parameters that best fits the model, or that best gives us the desired output, we can see the neural network play a vital role. In addition, the implementation of neural network in the model overcomes the issues that we encountered in previous models with the likes of data sparsity, and poor representation of unknown words in the vocabulary.

References:

[1] https://en.wikipedia.org/wiki/N-gram

[2] Bengio, Yoshua, et al. "A neural probabilistic language model." *The journal of machine learning research* 3 (2003): 1137-1155.

[3] https://abhinavcreed13.github.io/blog/bengio-trigram-nplm-using-pytorch/