TYH: Good evening, I am Tang Yu Hin.

YKL: I am Yue Ka Leung.

TYH: Our project title is “A peek into word embedding using word2vec”. First, let me briefly explain what word embeddings and word2vec are. Basically, word embeddings are a way to represent words as vectors and word2vec is a popular technique or framework to achieve this, in which the vector of a word depends on its surrounding words and hence captures its semantic meaning. In the vector space, each dimension may correlate to an aspect of the word, for example, in the simple example here, we use two bases, royalty and gender, to generalize the four words man, woman, king, queen. Then, the value of the word “queen” can be computed through vector addition and subtraction. Queen equals to king minus man plus woman. This is a famous analogy that some of you might have heard of too. Of course, things are more complicated in real life applications.

YKL: There are mainly two types of models in Word2Vec, CBOW and Skip-Gram, and we will focus on Skip-Gram as it is easier to understand and implement. It is simply the other way around for the CBOW model anyway. The Skip-Gram model accepts input of a center word and then estimates the conditional probability of outside context words appearing nearby given the center word by using the SoftMax function which gives a probability distribution as it has a range of [0, 1]. The output of the function is the required probability used for training. In the training, all words are first assigned unique randomized vectors with small values, usually in the range of [0, 0.5 or 0.8]. Then, the vectors are multiplied to a weight matrix. We apply the model and predict the outside context words of center words from the estimated probability. We update the weight matrix with gradient descent algorithm if there is difference between the predicted and actual words. This updating process is repeated many times – we are simply adjusting the word vectors continuously to maximize the estimated probability and hence minimize the loss function, so the predictions match the actual training data, and the resulting vectors capture the semantic meanings of words in sentences. After training, we can use the trained model to compute cosine similarities of words – which is also the similarity of semantic meanings of words as we recall the geometrical meaning of dot product of vectors. Interestingly, dot products are involved in both the calculations of cosine similarity and probability. However, they are not the same at all as we normalize the vectors to neglect the effect of occurrence count of words when we calculate cosine similarity.

TYH: We have tried to implement our version of Skip-gram model using Python. By feeding in a roughly 400-word essay, we tried predicting the similarity between certain target words. The results are accurate to a certain extent. We also tried using a pre-trained model, and it yields quite high accuracy. As for the difference between our implementation and pre-trained model, this could be due to a lot of factors, most importantly the size of the dataset, and the number of epochs we used. The code is posted on GitHub, and this would be the end of our presentation. Thank you.