TYH: Good evening, our project title is “A peek into word embedding using word2vec”, which is a way of representing words into vector so that analysis can be performed on words to calculate the probability of relation between different words.

TYH: First, let me briefly introduce what are word embeddings and word2vec. Basically, word embeddings are a way to represent words as vectors and word2vec is a popular technique to obtain vector representations of words. Each dimension may correlate to an aspect of the word, for example, in the simple example below, 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. Moreover, we can also see in the graph “king” and “queen” vectors are close to each other, having a high cosine similarity, meaning that they might be correlated.

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. So, the Skip-gram model accepts input of a context word and tries to calculate the conditional probability of outside context words given a context word, as visualized in the diagrams here. It first assigns each word with a vector, usually assigning 1 to one dimension and all other dimensions are set to 0. The word vector is then multiplied to a weight matrix, resulting in a vector that represents the word. We would then feed the word vector of the context word to the output weight of outside context word, which is like “dot product”, or cosine similarity that computes how similar the two vectors are to each other. The closer to 1, the more similar the words are. The SoftMax function is used here to generate a probability distribution by bounding the output value to between 0 to 1, so its output is the required probability. This model tries to understand the semantic meaning of words in a sentence. This process is repeated many times as the model goes through the dataset, and the weight matrix is constantly adjusted to make the probability more accurate.

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 data set, and the number of epochs we used. The code is posted on GitHub, and this would be the end of our presentation. Thank you.