Hello my name is Jihwan Bae, and in today’s session I will be presenting you on the topic of Translating Embeddings for Modeling Multi-relational Data. (Next) Here is the table of contents. Firstly, I will go over some of the necessary preliminaries and background knowledge for understanding TransE. Then, I will explain the model with accordance to the paper. Consequently, I will present my own implementation. And finally end with conclusion.

(Next) To start off, let’s review the some of the essential concepts by dissecting the title. (Next) First keyword is translating. Translating is a Type of transformation that takes each point in a figure and slides it the same distance in the same direction. (Next) So, in this case the initial point shifted by 3 and 2 in x and y direction respectively. (Next) Nextly, we will examine the embedding. (Next) Suppose that we have these 4 words. We could represent individual word as an one hot vector. But, then the computational cost increases exponentially with increasing word. (Next) Instead, we could we embed each word onto 2 dimensional space composed of Royalty and Femineity axis. This way, we could gain representation without increasing the number of dimension. So, for instance, King plus Woman will give Queen. This is quite intuitive as king with feminine feature is queen. (Next) Finally, let’s examine the multi-relational data. (Next) On the left is the classical graph structure. Graphs are composed of nodes and edges, typically with only one type of node and edge. In contrast, Multi-relational data refers to directed graphs whose nodes correspond to entities and edges of the form head, label, and tail each of which indicates that there exists a relationship of label between the entities head and tail. On the graph on the right, head is Tom Holland and the tail is English and the relation between head and tail is nationality as tom Holland is an English actor. Likewise, Willem Dafoe is American.

(Next)Having these concepts in mind, let’s move on to the explanation the actual model. TransE focuses on modeling multi-relational data from Knowledge bases with the goal of providing an efficient tool to complete them by automatically adding new facts, without requiring extra knowledge. A knowledge graph is a collection of nodes and edges where each node represents a real-world entity, and each edge represents a relation between entities.

(Next) This is the whole algorithm, but for now let’s focus on the loss function. I will go over the algorithm in detail when I present my own implementation of the paper. (Next) Given a training set S of triplets (h, l, t) our model learns vector embeddings of the entities and the relationships. We want the result of head plus tail to approximately equal to tail when (h, l, t) triplet holds. In other words, t should be a nearest neighbor of h + l), while h + l should be far away from t for the set of corrupted triplets. The corrupted triplets are composed of training triplets with either the head or tail replaced by a random entity but not both at the same time). Let’s derive the loss function step by step. Firstly, we want to minimize the dissimilarity measure for the correct triplet relationship. Next, we also want to maximize the dissimilarity measures for the corrupted triplet, which is equivalent to minimizing the negative distance of the corrupted triplet. Finally, we add the margin hyperparameter in order to prevent the situation of the distance between the corrupted set and the correct set being exactly equal to each other. Let me explain in detail what this means in the next slide.

(Next) This is the situation that may occur when we do not introduce the margin term gamma. The distance between the correct head and tail is exactly equal to the distance between corrupted head and the tail. But this is not what we want. We want the distance between the correct ones to be closer than the corrupted pairs. So we implement the gamma term to prevent such case. The optimization is carried out by stochastic gradient descent (in minibatch mode), over the possible h, l and t.

(Next) We will now move on to examine some of the related works. Structured embedding is a very closely related work. The basic idea is the same, but the difference is that instead of adding the relation vector l, SE embeds the head and tail entities in a way that their embeddings are close to each other in some subspace that depends on the relationship for the correct set. Two different projection matrices for the head and for the tail is used in order to account for the possible asymmetry of relationship.

SE is actually more expressive than the transE as linear operators in dimension k + 1 reproduce affine transformations in a subspace of dimension k. In other words, while transE works on k dimension, structured embedding works one dimension higher in k+1 dimension.

(Next) This can be explained by affine transformation. Let’s assume that k is equal to 2. And we want to perform translation of x y pair to x+1, y.

(Next) However this is impossible. If x y is (0,0), regardless of what the transformation matrix is, we cannot translate the x, y as the result will always be equal to zero, as shown on the right.

(Next) Instead, what we could do is increase one dimension. We increased one dimension by adding z-axis and achieved 2-d translation from (x,y) to (x+1, y). The 3x3 matrix is actually performing shear in the 3d, but if we fix the z dimension to constant, in this case Z equal to 1, that ‘slice’ of the layer in performing translation in the 2d. (Next) To recap, SE, with L2 as the identity matrix and L1 taken so as to reproduce a translation is then equivalent to TransE. So you could think of Structured embedding as a similar model to transE with greater expressiveness.

(Next) Another related approach is the Neural Tensor Model. A special case of this model corresponds to learning scores s(h, l, t) as shown on the screen. Score is lower for the corrupted triplets. Considering our norm constraints h norm and t norm being equal to 1, because the embedding vectors are normalized each iteration, and the ranking criterion that l norm does not play any role in comparing corrupted triplets, transE corresponds to the score formula of the neural tensor model. When the capital l is the identity matrix, and l = l1 = −l2. As in the case of SE, we can see that TransE is a simpler model with much fewer parameters.

(Next) Experiment was carried out with related models. For the data set, wordnet and freebase were used. Wordnet knowledge base is designed to produce an intuitively usable dictionary and thesaurus, and support automatic text analysis. Freebase is a huge and growing KB of general facts; For the testing, redundant data due to reversing the relationship were removed. We wanted to test for both small and large data set. For the smaller one FB15k was used and for the the larger one, FB1M which 25k relationships and more than 17 millions training triplets, was used.

For the evaluation, the ranking procedure was used. For each test triplet, the head is removed and replaced by each of the entities of the dictionary in turn. Dissimilarities of those corrupted triplets are first computed by the models and then sorted by ascending order; the rank of the correct entity is finally stored. This whole procedure is repeated while removing the tail instead of the head. We report the mean of those predicted ranks and the hits@10, which is the proportion of correct entities ranked in the top 10.

These metrics are indicative but can be flawed when some corrupted triplets end up being valid ones. To avoid such a misleading behavior, the triplets that appear either in the training, validation or test set were removed. The original (possibly flawed) is termed raw, while we refer to the newer as filtered (or filt.).

In the baselines, The first method is Unstructured, a version of TransE which considers the data as mono-relational and sets all translations to 0. We also compare with RESCAL, the collective matrix factorization model and the energy-based models SE, SME(linear)/SME(bilinear) and LFM [6]. RESCAL is trained via an alternating least-square method, whereas the others are trained by stochastic gradient descent, like TransE.

As expected, the filtered setting provides lower mean ranks and higher hits@10, which is a clearer evaluation of the performance of the methods in link prediction. However, generally the trends between raw and filtered are the same.

Despite the lower expressiveness of TransE, TransE outperforms all counterparts on all metrics, usually with a wide margin, and reaches some promising absolute performance scores such as 89% of hits@10 on WN and 34% on FB1M. It is likely due to the fact that transE model is a more direct way to represent the true properties of the relationship, and also due to the fact that optimization is difficult in embedding models. Simpler model simplify the training and prevent underfitting, which seems to compensate for a lower expressiveness in this case.

(Next) Results were classified depending on several categories of the relationships in table 4. Categories are 1-TO-1, 1-TO-MANY, MANY-TO-1, MANY-TO-MANY. appears that, as one would expect, it is easier to predict entities on the “side 1” of triplets (i.e., predicting head in 1-TO-MANY and tail in MANY-TO-1),. SME(bilinear) proves to be very accurate in such cases because they are those with the most training examples.

Table 5 below gives examples of link prediction results of TransE on the FB15k test set. This illustrates the capabilities of transE to predict the tail. Even if the good answer is not always top-ranked, the predictions reflect common-sense.

(Next) Using FB15k, test was carried out to test how well methods could generalize to new facts by checking how fast they were learning new relationships. Results for Unstructured, SE, SME(linear), SME(bilinear) and TransE are presented in the Figure on the left. The performance of Unstructured is the best when no example of the unknown relationship is provided, because it does not use this information to predict. But, of course, this performance does not improve while providing labeled examples. TransE is the fastest method to learn: with only 10 examples of a new relationship, the hits@10 is already 18% and it improves monotonically with the number of provided samples. It seems to be due to the simplicity of the TransE model that makes it able to generalize well, without having to modify any of the already trained embeddings.

(Next) I will now present my own implementation of the code. First step was to create negative datasets for training. In order to do so dataset was examined. I was able to find out that this data set is very simple as the unique head is simply arithmetic sequence of numbers with step size one.

Then, I created rel\_matrix, which is an adjacent matrix filled with relationship l if relationship exists for the head and tail pair. If the relationship does not exist, then it is filled with zero.

(Next) Our goal here is to create corrupted set, so we want to eliminate correct sets and leave only the corrupted sets. Based on this idea, I assigned negative one for the correct set relations and positive one for the corrupted set relations. As seen from previous slide, unique head set is simply arithmetic sequence. So, I created unique head set named series by using arrange function.

Then, I multiplied series with relation matrix of the particular head. This makes the corrupted set positive, while making the correct ones negative. Using this feature, I was able to generate the negative headset. Applying the same logic on the tail, I created the negative tail set as well.

(Next) For the model, I followed the algorithm as the paper states. In the init function, I created the emedding for the layer, which embeds the labels to embedding dimension. The weight of the embedding was initialized through Xavier Initialization to facilitate quick convergence. The embedding for the entity was also created and same initialization methodology was used.

(Next) In the forward function, embedding of the entities and relationships were performed using the embedding functions defined in the init section. Consequently, the loss was calculated. Matching the dimension was the trickiest part for me. I used unsqueeze function to match the dimension of the corrupted sets with relation set and correct sets. For instance, in the case of head\_prime corrupted set, I unsqueezed relation matrix and tail matrix and added a dimension at index 1 in order to to match the dimension with the head\_prime matrix. Then I calculated norm along the embed\_dim axis. Same logic was applied for the tail corrupted set. This was done to to fully utilize the gpu’s parallel computing, thus I refrained from using the for-loop, instead carried out the calculation is matrix format.

If you look at the loss function, you can notice torch dot max. This was done to implement the function of returning only the positives. Loss function is exactly the same as the suggested algorithm from the paper, only difference is that 2 x #neg\_sample is multiplied for dissimilarity measure for correct set. This is because for each positive set, there are 2 corrupted set(h\_p, t\_p) each with neg # of samples,,,, just in case you are wondering why I multiplied this term.

(Next) The dataset was loaded through the usage of dataset class. (Next) This is the code for the execution. For every epoch, loss was calculated and I coded to display its result using matplotlib. Furthermore, optuna package was used for the hyperparameter tuning. (Next) I set the learning rate, embedding dimension, and gamma value to be variables. And the best result was given when learning rate is 9.3 time ten to the power of 5, gamma equal to 8 and embedding dimenson equal to 18. Initially I wanted to do grid-wise search, yet the multi-dimensional analysis took too long, so alternatively I tested using random combinations of each variables. I’ve done 6 tirals. Obviously, higher gamma value led to higher starting point, but if the learning rate is adequate, the convergence seemed to be achieved in similar pace for all testing variataions. And embedding dimensions seems to not affect the result as much, but this may be biased as the range of embedding dimension is not significantly different from each other for all.

(Next) To sum up, TransE was a new approach to learn embeddings of KBs, focusing on the minimal parametrization of the model to primarily represent hierarchical relationships. It showed that it works very well compared to competing methods on two different knowledge bases. TransE is highly scalable model, as shown through the application of a very large-scale chunk of data Even in complex and heterogeneous multi-relational domains simple yet appropriate modeling assumptions can lead to better trade-offs between accuracy and scalability. The greater expressivity of these models comes at the expense of substantial increases in model complexity which results in modeling assumptions that are hard to interpret, and in higher computational costs.