Evaluasi Kinerja Penggabungan *Knowledge Graph Embedded-Based Question Answering* dan TransP pada Data Freebase

Fransisco Remon Liemena, Henry Novianus Palit, Alvin Nathaniel Tjondrowiguno
Program Studi Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) - 8417658
E-mail: fransiscoliemena@gmail.com, hnpalit@petra.ac.id, alvin.nathaniel@petra.ac.id

ABSTRAK

Penggunaan *graph* sebagai bentuk penyimpanan dan analisa data semakin hari semakin meningkat. Salah satu penerapan data dalam bentuk *graph* adalah *knowledge graph*. Telah banyak penelitian yang dilakukan untuk perolehan informasi pada *knowledge graph*, salah satunya adalah *natural language question answering*. Namun, penelitian-penelitian tersebut menggunakan *query* langsung untuk mencari jawabannya. *Knowledge Graph Embedding-based Question Answering* (KEQA) merupakan penelitian terbaru yang memanfaatkan *deep learning* dan menggunakan *embedding* untuk mencari jawaban pertanyaan. KEQA terbukti mampu mengungguli metode-metode sebelumnya, namun KEQA hanya menggunakan metode *embedding* yang simpel.

Knowledge graph embedding merupakan salah satu metode dalam merepresentasikan sebuah *knowledge graph*, dimana tiap *entity* dan *relation* dalam *knowledge graph* direpresentasikan dalam bentuk vektor (*embedding*) dengan menggunakan *deep learning*. Banyak penelitian yang telah dilakukan sebelumnya, namun penelitian-penelitian tersebut tidak cukup menggali kedalaman dari *knowledge graph*. TransP merupakan penelitian yang memanfaatkan hubungan tidak langsung untuk merepresentasikan *knowledge graph*. TransP terbukti mengungguli model-model *embedding* lainnya. Dari ini, KEQA akan dibangun menggunakan TransP dengan harapan akurasi KEQA akan meningkat.

Berdasarkan hasil penelitian, TransP mendapat nilai Mean Rank sebesar 5.390,25 dan nilai HIT10 sebesar 28,5%. Kemudian, nilai akurasi maksimum yang dapat diperoleh oleh KEQA adalah 88,89% dengan *embedding* dan 88,15% tanpa *embedding*. Dari hasil penelitian juga ditemukan bahwa nilai parameter untuk *scoring* lebih berpengaruh pada KEQA dengan *embedding*. Dengan ini, disimpulkan bahwa TransP dapat meningkatkan akurasi KEQA.

Kata Kunci: *Knowledge graph embedding*, *Natural language question answering*, *deep learning*, KEQA, dan TransP.

ABSTRACT

In the past few years, data storage and analysis using graph keep increasing. One of the implementation of this is knowledge graph. There are many methods proposed on information extraction from knowledge graph, one of them is natural language question answering. However, all of the researches around question answering use direct query to find the answer. Knowledge Graph Embedding-based Question Answering (KEQA) is the latest method that implements deep learning and embedding to answer questions. Experiments demonstrate that KEQA outperforms other question answering methods. Despite having high accuracy, KEQA still uses simple and outdated embedding method.

Knowledge graph embedding is one of the method for knowledge graph representation where the entities and relations are represented in vector (embedding) using deep learning. Many proposed embedding methods do not really consider the depth of a knowledge graph. TransP is a proposed method that consider the indirect relationship to represent a knowledge graph. Experimental results show that TransP outperforms other embedding methods in the task given. Based on this, KEQA will be built using TransP with the expectation that the accuracy of KEQA will increase.

Based on the result of the experiment, TransP achieves Mean Rank of 5.390,25 and HIT10 of 28,5%. After that, KEQA with embedding can achieve up to 88,89% accuracy, and KEQA without embedding can achieve up to 88,89% accuracy. Experiment also shows that scoring parameters value will affect KEQA with embedding. In conclusion, TransP can increase the accuracy of KEQA.

Keywords: *Knowledge graph embedding, Natural language question answering, deep learning, KEQA, and TransP.*

1. PENDAHULUAN

Dalam beberapa tahun ini, penggunaan *graph* sebagai bentuk penyimpanan dan analisa data semakin meningkat [8]. *Knowledge graph* merupakan salah satu hasil penerapan dari teknologi *Knowledge Base* dalam bentuk *graph*. *Knowledge graph* merepresentasikan data dalam bentuk *triple*, yaitu <e1, r, e2> dimana e1 dan e2 merupakan *node/entity*, dan r merupakan relasi/hubungan dari kedua *node/entity* tersebut. Dengan *knowledge graph*, pengguna dimudahkan dalam memahami informasi serta hubungan antar *entity* dalam *graph* karena setiap *entity* terhubung satu sama lain dan membentuk hubungan timbal balik. Hubungan inilah yang nantinya akan menjadi kunci dalam perolehan informasi [11].

Telah banyak penelitian yang dilakukan untuk perolehan informasi pada *knowledge graph*, dan salah satu subtopiknya adalah *natural language question answering*. Menggunakan *natural language* akan membuka cara baru yang lebih *user-friendly* dan intuitif untuk mengeksplor *knowledge base*, serta

dapat menyediakan cara yang mudah dan natural untuk pengguna dalam memperoleh informasi [14]. Pertanyaan yang dijawab biasanya adalah SimpleQuestion. SimpleQuestion merupakan pertanyaan yang hanya mengandung satu *head* entity dan satu predikat, dan mengambil *tail entity* sebagai jawaban dari pertanyaan tersebut. [13].

Terdapat penelitian yang memanfaatkan *neural network* pada tingkat kata dan karakter [7], terdapat juga penelitian yang melakukan formalisasi *entity linking* dan *relation linking* sebagai Generalised Travelling Salesman Problem [3], dan masih banyak lagi. Namun, semua penelitian diatas menggunakan *query* langsung untuk mencari jawaban. *Knowledge Graph Embedding Based Question Answering* (KEQA) *Framework* [5] merupakan hasil penelitian terbaru untuk *natural language question answering*, dimana *framework* ini memanfaatkan vektor-vektor *embedding* hasil *training* pada suatu model *knowledge graph embedding* untuk mencari jawaban dari pertanyaan.

Knowledge graph embedding merupakan salah satu metode dalam merepresentasikan sebuah *knowledge graph* dimana *entity* dan *relation* direpresentasikan dalam bentuk vektor representasi (*embedding)* dengan menggunakan *deep learning*. Penggunaan *deep learning* dalam *embedding* memberikan performa yang cepat dan lebih baik, dan juga dapat secara otomatis mempertimbangkan fitur semantik pada *entity* dan *relation* yang direpresentasikan. [2] membangun model TransE untuk melakukan *embedding* berdasarkan translasi. TransE menjadi model yang simpel dan cepat, serta tidak menggunakan banyak sumber daya, namun tidak memperhatikan kompleksitas dari relasi (1-to-many, many-to-many). [6] membangun model lain, TransR. TransR memiliki performa lebih dibanding TransE dan pendahulunya, namun TransR tidak cukup menggali kedalaman dari *knowledge graph*. [13] melakukan penelitian terkait model *embedding* dengan memanfaatkan LSTM, dan menghasilkan model TransP. Hasil penelitian [13] menunjukkan bahwa TransP mengungguli model-model *embedding* lainnya termasuk TransE dan TransR.

Berdasarkan cara kerja dari KEQA dan *knowledge graph embedding*, hasil dari KEQA *Framework* akan bergantung dari model yang digunakan untuk *knowledge graph embedding*. [5] menggunakan model *embedding* TransE, TransH [12], dan TransR. Maka dari itu, penelitian ini akan menggunakan model *embedding* TransP [13]. Pada penelitian ini, akan dilakukan penggunaan model TransP pada KEQA *Framework* dengan harapan akurasi dari KEQA dapat meningkat.

2. DASAR TEORI

2.1 Knowledge Graph

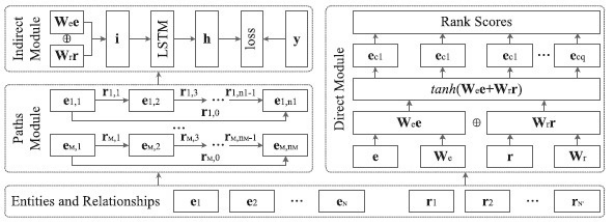
Knowledge graph merupakan konsep yang mendeskripsikan *entity-entity* yang ada dengan relasi-relasi antar *entity* tersebut, dalam bentuk *graph* [10]. *Knowledge graph* juga dapat diartikan sebagai sejenis representasi *knowledge* berbasis *graph*. Data yang ada di dalam knowledge graph direpresentasikan dengan *entity*, dan relasi antar *entity*. *Knowledge graph* sering digunakan dalam beberapa tujuan [9]:

- Representasi dan pertimbangan berdasar knowledge (languages, schema and standard vocabularies),
- Penyimpanan knowledge (graph databases and repositories),
- Knowledge engineering (methodologies, editors and design patterns),
- Knowledge learning secara otomatis termasuk pembelajaran dan populasi skema.

Knowledge graph dapat dideskripsikan dengan G = (E, R, S), dimana E = {e_1, e_2, …, e_n} yang merupakan sekumpulan *entity*, R = {r_1, r_2, …, r_n} yang merupakan sekumpulan relasi, dan S ⊆ ExRxE yang merepresentasikan batasan antar *entity* dan relasi, yaitu *triples set* T. Setiap *triple* pada T dapat digambarkan sebagai T_i = <h_i, r_i, t_i> dimana h adalah *entity* awal, t adalah *entity* akhir, dan r adalah relasi antara h dan t [13]. Tiap *triple* yang ada dapat membangun relasi tidak langsung. Jika terdapat *triple* <h_1, r_1, t_1> dan <t_1, r_2, t_2>, maka secara tidak langsung terdapat hubungan antara h_1 dan t_2.

2.2 TransP

TransP [13] merupakan salah satu model, lebih spesifiknya *learning model*, yang dikembangkan dan digunakan untuk menyelesaikan tugas seputar *knowledge graph* seperti *knowledge graph completion.*



Gambar 1. Struktur dari TransP

Model TransP terbagi menjadi beberapa tahapan pengerjaan, yaitu *Entity Relationship Path Construction, Indirect Module,* dan *Direct Module*. Pada *entity relationship path construction*, untuk tiap triple T = <*h, r, t*> akan dicari tiap *indirect path* pada *graph* dari *h* menuju *t*. *Indirect module* merupakan *module* untuk *training triple* dengan relasi tidak langsung. Pada *indirect module*, digunakan LSTM (*Long Short-term Memory*) sebagai dasar dari *module*. *Direct module* merupakan *module* untuk *training triple* dengan relasi langsung. *Direct module* menggunakan *feed-forward neural network* sebagai dasar dari *module*. Untuk memperoleh representasi *tail* yang benar untuk sebuah *triple*, i*ndirect module* dan *direct module* memanfaatkan matriks kombinasi sebagai operatornya. Ketiga bagian ini akan digunakan saat *training*, untuk memperoleh *embedding* dari tiap-tiap *entity* dan *relation* dari *knowledge graph,* dalam bentuk vektor. Vektor-vektor hasil *training* akan digunakan untuk menjawab kebutuhan pengguna, seperti *entity prediction, relationship prediction*, dan sebagainya.

2.3 KEQA Framework

KEQA *Framework* [5] merupakan *framework* yang dibuat untuk melakukan *question answering*. KEQA *Framework* berfokus untuk menjawab tipe pertanyaan yang lebih umum, seperti *simple question*. *Knowledge graph embedding* akan menghasilkan representasi dari tiap *entity* dan *relation* dari *knowledge graph* dalam bentuk vektor, Vektor-vektor inilah yang nantinya akan digunakan dalam mencari jawaban dari pertanyaan yang diberikan. Secara garis besar, KEQA akan bekerja dalam tiga tahap, (1) berdasarkan pertanyaan yang diberikan dan representasi dari predikat jawabannya (dalam bentuk vektor), KEQA akan melakukan *training* untuk *predicate learning model* yang

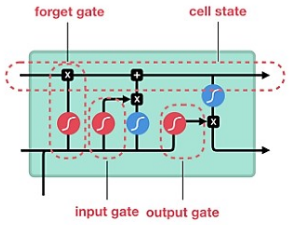
nantinya akan digunakan untuk memperoleh vektor yang diperkirakan merupakan predikat dari pertanyaan tersebut. Selain model untuk predikat, juga terdapat model untuk *head entity* dengan konsep dan cara kerja yang sama. (2) KEQA menggunakan *Head Entity Detection Model* untuk memperkecil jumlah kandidat untuk *head entity*. Tujuan utamanya adalah untuk menemukan beberapa *token* pada pertanyaan sebagai kemungkinan nama *head entity*, kemudian mencari *entity* dengan nama yang mirip atau persis. (3) Dengan kandidat *head entity* yang telah ada, akan dicari semua *triple* dengan *head entity* tersebut. Setelah pencarian *triple* dilakukan, akan dicari *triple* yang paling dekat dari hasil pada proses nomor 1, dan dijadikan jawaban akhir. *Triple* terdekat akan dicari menggunakan rumus pada Persamaan (1).

$$\|p_l - \check{p}_l\|_2 + \beta_1\|e_h - \check{e}_h\|_2 + \beta_2\|f(e_h, p_l) - \check{e}_t\|_2 - \beta_3\, sim\left[n(h), HED_{entity}\right] - \beta_4\, sim[n(l), HED_{non}] \quad (1)$$

(*h, l, t*) merupakan *triple* kandidat. ($\check{\boldsymbol{e}}_h$, $\check{\boldsymbol{p}}_l$) merupakan hasil prediksi dari *entity learning model* dan *predicate learning model*. ($\boldsymbol{e}_h$, $\boldsymbol{p}_l$) merupakan nilai *embedding* dari *h* dan *l*. $\boldsymbol{f(e,p)}$ merupakan fungsi dari model *embedding* yang digunakan untuk mencari representasi *tail* dari sebuah *entity* dan *predicate*. $\boldsymbol{n(.)}$ merupakan fungsi yang mengembalikan nama dari *entity*/*predicate*. $\boldsymbol{HED_{entity}}$ merupakan hasil *entity detection model* yang dianggap sebagai token *entity*. $\boldsymbol{HED_{non}}$ merupakan token yang bukan token *entity*. $\boldsymbol{sim[.]}$ Merupakan fungsi *fuzzy* untuk mencari kesamaan antar *string*.

2.4 Long Short-Term Memory

Long Short-Term Memory (LSTM) [4] merupakan jenis lain dari Recurrent Neural Network (RNN). LSTM memiliki *layer* yang lebih kompleks dibanding RNN. LSTM memperkenalkan mekanisme *gates*, dimana *gates* dapat mengontrol informasi apa yang akan diproses dan tidak diproses. Terdapat juga *cell state* yang menjadi penyalur informasi, dimana informasi-informasi penting dibawa di dalam *cell state* ini.



Gambar 2. Arsitektur unit LSTM

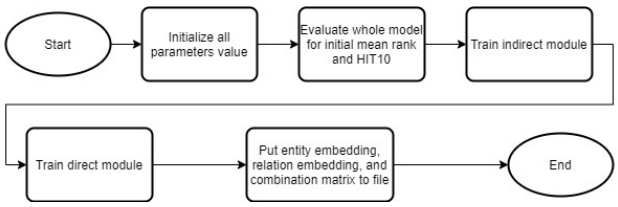
Unit pada LSTM memiliki tiga *gates,* yaitu *forget gate, input gate,* dan *output gate*. *Forget gate* akan menentukan informasi mana yang tidak penting dan akan dihilangkan dari *cell state* dengan fungsi Sigmoid. Kemudian *input gate* akan menentukan informasi mana yang akan disimpan ke *cell state* dengan fungsi Sigmoid. *Input gate* akan bekerja bersama dengan satu *layer* tanh, dimana *layer* ini akan terlebih dahulu menentukan kandidat yang akan disimpan sebelum *input gate* memasukan nilai ke *cell state.* Kemudian *output gate* akan menentukan informasi apa yang harus dikeluarkan. Dengan mekanisme *gates*, informasi yang tidak penting dapat dibuang sehingga informasi yang lebih penting dapat terus dibawa dan digunakan.

3. DESAIN SISTEM

Sistem yang digunakan untuk menjawab pertanyaan dapat dibagi menjadi dua bagian, yaitu *knowledge graph embedding* (TransP) dan *question answering* (KEQA).

3.1 TransP

Knowledge graph terdiri dari sekumpulan entitas dan relasi. Proses *knowledge graph embedding* akan menghasilkan representasi dari entitas-entitas dan relasi-relasi pada *knowledge graph* dalam bentuk vektor dengan dimensi yang telah ditentukan. Pada awalnya, seluruh nilai dari parameter yang diperlukan (ukuran *batch*, *learning rate*, dan lain-lain) akan diinisialisasi, Kemudian akan dilakukan evaluasi awal untuk mendapatkan nilai Mean Rank dan HIT10. Setelah itu, dilakukan *training* pada *indirect module*, dan setelahnya *direct module*. Setelah *training* selesai, variabel-variabel yang dibutuhkan dari TransP akan disimpan. Alur untuk proses *knowledge graph embedding* dapat dilihat pada Gambar 3.

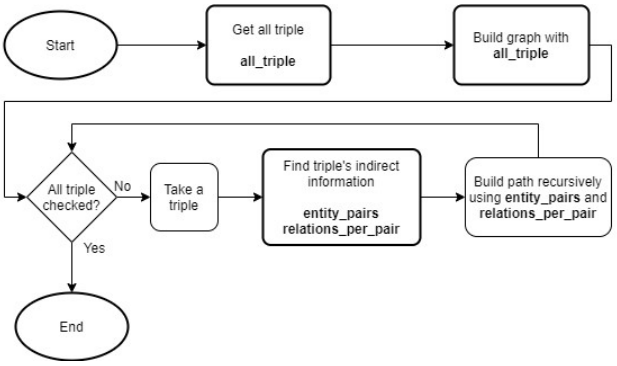


Gambar 3. Keseluruhan proses TransP

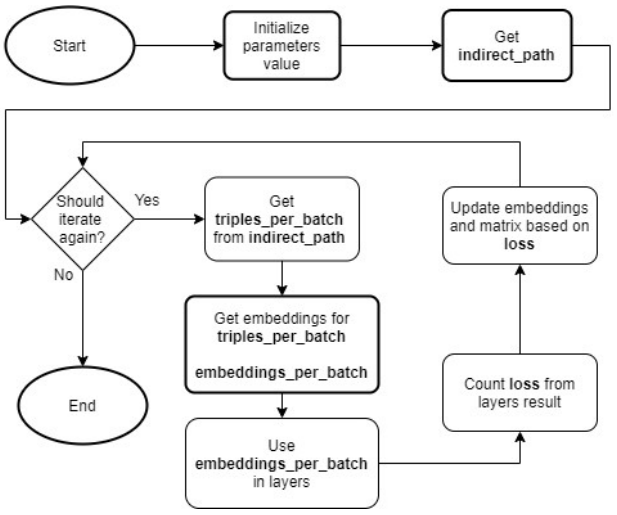
3.1.1 Indirect Module

Indirect module adalah modul dalam TransP yang berperan dalam menganalisa hubungan tidak langsung dari *triple-triple* yang ada, dan men-*train* TransP sesuai dengan hubungan tersebut. Modul ini akan mengambil tiap-tiap *triple* yang ada pada hubungan tidak langsung (*indirect path*) tersebut sebagai inputannya. *Indirect path* harus dibentuk terlebih dahulu sebelum dilakukan *training*. *Indirect path* terdiri dari dua bagian, yaitu *path* untuk *entity* dan *path* untuk *predicate/relation*. Proses pembentukan *indirect path* dilakukan dengan mengambil tiap *triple* pada *knowledge graph*, mencari informasi terkait hubungan tidak langsung *triple* tersebut. Informasi yang diambil adalah pasangan *head* dan *tail* dari *triple-triple* yang berada dalam hubungan tidak langsung tersebut, dan *predicate-predicate* yang menghubungkan *head* dan *tail* tersebut. Setelah informasi yang dibutuhkan diambil, *path* untuk *entity* dan *relation* akan dibentuk. Karena sepasang *entity* bisa terhubung dengan *predicate* yang berbeda-beda, maka dimanfaatkan metode rekursi untuk membangun *path*. Proses pembentukan *indirect path* dapat dilihat pada Gambar 4.

Pada proses *training* untuk *indirect module*, *embedding* dari *head* dan *predicate* dari *triple-triple* akan menjadi input, dan nilai *embedding tail* akan menjadi target. *Indirect module* terdiri dari beberapa *layer*, yaitu *indirect path layer*, *combination layer*, dan LSTM *layer*. *Indirect path layer* akan mengambil tiap-tiap *triple* dari *indirect path* untuk dijadikan input untuk *layer* selanjutnya. *Combination layer* akan melakukan kombinasi dari *head* dan *predicate* pada *triple* dengan mengalikan nilai *embedding entity* dan *predicate* dengan matriks kombinasi yang ada, dan kemudian hasil kombinasi akan dilanjutkan ke LSTM *layer*, untuk dipelajari urutannya. Proses *training* untuk *indirect module* dapat dilihat pada Gambar 5.



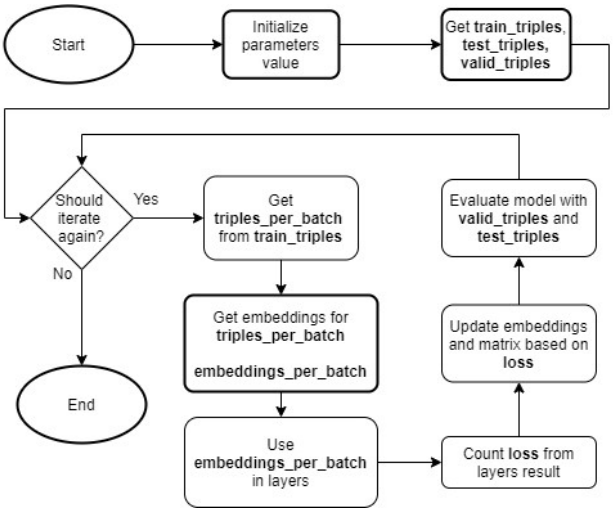
Gambar 4. Proses pembentukan *indirect path*



Gambar 5. Proses *training indirect module*

3.1.2 Direct Module

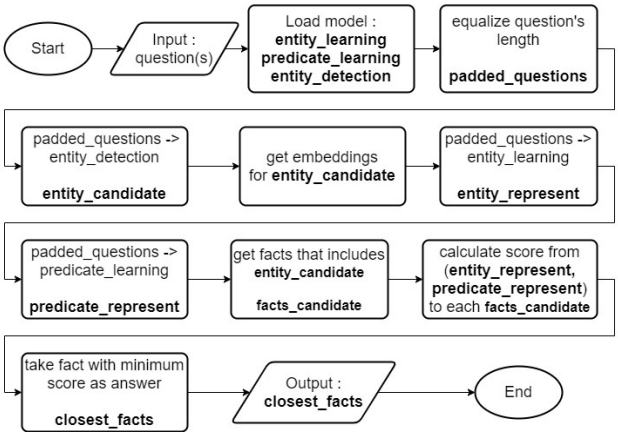
Setelah *indirect module* selesai di-*train*, maka proses *training* akan dilanjutkan ke *direct module.* Modul ini berperan untuk menganalisa hubungan langsung dan men-*train* TransP sesuai dengan hubungan tersebut. Modul ini akan mengambil semua *triple (head* dan *predicate*) pada *knowledge graph* sebagai inputnya, dan *tail* dari *triple* sebagai target. *Direct module* terdiri dari beberapa *layer*, yaitu *combination layer* dan *scoring layer*. *Combination layer* akan mengkombinasikan *head* dan *predicate* dari *triple* input dengan mengalikan nilai *embedding head* dan *predicate* dengan matriks kombinasi yang ada, dan kemudian hasil tersebut akan melewati fungsi tanh. Setelah itu, hasil kombinasi akan dilanjutkan ke *scoring layer*, dimana *layer* ini akan menentukan *entity-entity* yang berpotensi menjadi *tail* dari *triple* input dengan fungsi softmax. Hasil dari *direct module* adalah semua list *entity* yang telah diurutkan berdasarkan kecocokannya dengan *triple* input. Proses *training* dari *direct module* dapat dilihat pada Gambar 6.



Gambar 6. Proses *training direct module*

3.2 KEQA

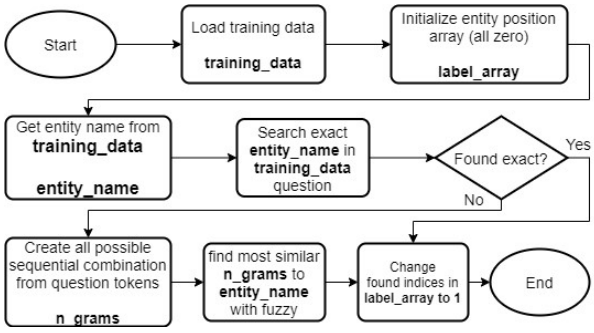
Proses *question answering* tidak langsung memberikan jawaban sebagai hasil akhir, melainkan sistem akan menerima pertanyaan, dan kemudian akan menghasilkan vektor yang merepresentasikan *head entity* dan *predicate* yang dianggap dapat menjawab pertanyaan yang diberikan. Setelah ini, barulah sistem akan mencari *triple* yang memiliki jarak terdekat dengan hasil sebelumnya. Alur untuk proses *question answering* dapat dilihat pada Gambar 7.



Gambar 7. Keseluruhan proses KEQA

3.2.1 Entity Detection Model

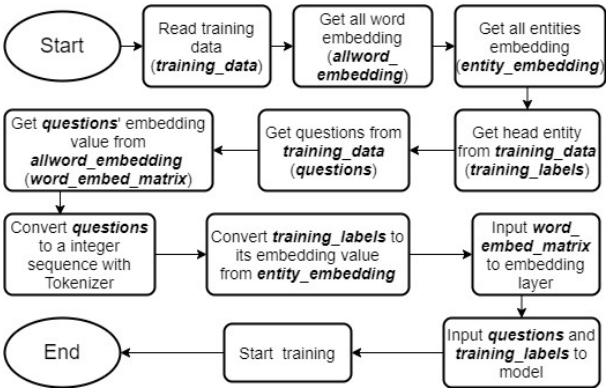
Model ini berguna untuk mengurangi jumlah *entity* yang harus dicek untuk mencari jawaban pertanyaan. Model akan menerima pertanyaan, dan model akan menentukan posisi (indeks) kata-kata dalam pertanyaan tersebut yang dianggap sebagai nama dari *head entity*. Kemudian setelah itu, dapat dicari *entity* sesuai dengan hasil tersebut. Entity Detection Model di-*train* menggunakan data pertanyaan dan posisi (indeks) dari kata dalam pertanyaan tersebut yang merupakan nama *head entity* dari *triple* yang menjawab pertanyaan tersebut. Pada dataset yang tersedia, tidak diberikan secara langsung posisi dari *head entity* pada pertanyaan sehingga untuk mendapat posisi dari *entity* dalam pertanyaan, dilakukan pencarian dengan proses pada Gambar 8.



Gambar 8. Proses pencarian nama *entity* pada pertanyaan

3.2.2 Entity Learning Model

Model ini berguna untuk menghasilkan vektor yang nilainya mendekati nilai *embedding* dari *head entity* dari *triple* yang bisa menjawab pertanyaan. Model ini di-*train* menggunakan data pertanyaan dan nilai *embedding* dari *head entity* dari *triple* yang menjawab pertanyaan tersebut. Proses *training* dari model ini dapat dilihat pada Gambar 9.



Gambar 9. Proses *training entity learning model*

3.2.3 Predicate Learning Model

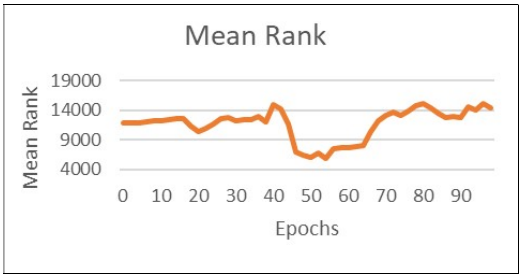
Berkonsep sama dengan *Entity Learning Model*, *Predicate Learning Model* berguna untuk menghasilkan vektor yang nilainya mendekati nilai *embedding* dari *predicate* dari *triple* yang bisa menjawab pertanyaan. Model ini di-*train* menggunakan data pertanyaan dan nilai *embedding* dari *predicate* dari *triple* yang menjawab pertanyaan tersebut.

4. PENGUJIAN SISTEM

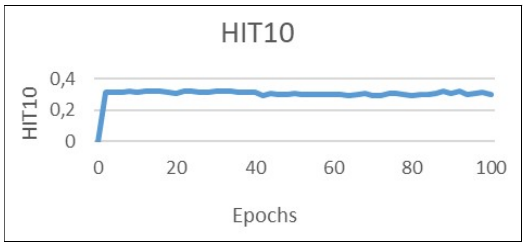
Keseluruhan sistem dibangun menggunakan bahasa Python dengan *framework* Tensorflow dan Keras, dengan dataset FB2M dan SimpleQuestions [1]. Dari data FB2M yang berjumlah 14.174.246 *triple* dengan 1.963.130 *entity* dan 6.701 *predicate*, diambil 50.000 *triple* yang terdiri dari 53.638 *entity* dan 1.130 *predicate*. Kemudian, penggunaan SimpleQuestions pada *entity learning model* dan *predicate learning model* disesuaikan dengan *triple* yang diambil.

4.1 TransP

TransP di-*train* dengan dimensi *embedding* = 100, *learning rate* = 0,01, *max indirect epochs* = 50, dan *max direct epochs* = 100. Hasil TransP dievaluasi menggunakan metode Mean Rank dan HIT10. Mean Rank akan mencari rata-rata nilai *ranking* dari *entity* yang seharusnya merupakan *entity* yang benar, dan HIT10 akan menghitung persentase jawaban TransP dimana *entity* yang benar berada pada *ranking* diatas 10. Model yang akan digunakan kedepannya adalah model dengan hasil terbaik yaitu pada *epoch* 56, dengan Mean Rank 5.390,25 dan HIT10 0,285 (28,5%). Plot untuk perubahan nilai Mean Rank dapat dilihat pada Gambar 10, dan plot untuk perubahan nilai HIT10 dapat dilihat pada Gambar 11.



Gambar 10. Plot perubahan nilai Mean Rank



Gambar 11. Plot perubahan nilai HIT10

4.2 KEQA

Pengujian dilakukan terhadap KEQA dengan *embedding* dari TransP, dan KEQA tanpa *embedding*. *Entity detection model* tidak membutuhkan *embedding* dalam prosesnya sehingga tidak dilakukan perbandingan.

4.2.1 Entity Learning Model

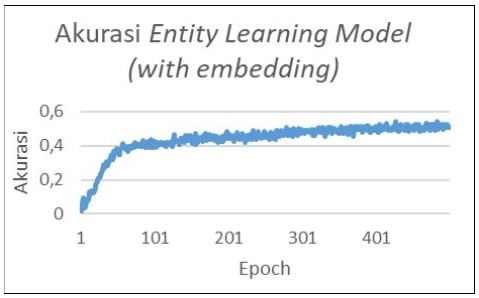
Kedua jenis *entity learning model* (dengan dan tanpa embedding) di-*train* dengan maksimal *epoch* 500. Akurasi akhir *entity learning model* dengan *embedding* (0,5088 atau 50,88%) jauh lebih tinggi daripada tanpa *embedding* (0,2191 atau 21,91%). Plot perubahan nilai akurasi saat *training entity learning model* dengan *embedding* dapat dilihat pada Gambar 12. Plot perubahan nilai akurasi saat *training entity learning model* tanpa *embedding* dapat dilihat pada Gambar 13.

4.2.2 Predicate Learning Model

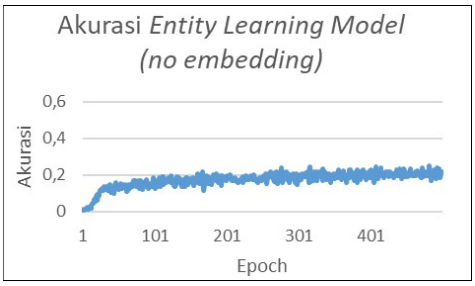
Kedua jenis *predicate learning model* (dengan dan tanpa embedding) di-*train* dengan maksimal *epoch* 500. Akurasi akhir *predicate learning model* dengan *embedding* (0,7138 atau 71,38%) jauh lebih tinggi daripada tanpa *embedding* (0,2014 atau 20,14%). Plot perubahan nilai akurasi saat *training predicate learning model* dengan *embedding* dapat dilihat pada Gambar 14. Plot perubahan nilai akurasi saat *training predicate learning model* tanpa *embedding* dapat dilihat pada Gambar 15.

4.2.3 *Entity Detection Model*

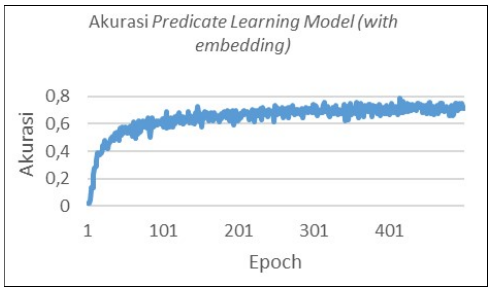
Entity detection model di-*train* dengan maksimal *epoch* 100 dengan hasil akurasi akhir 0,5712 (57,12%).



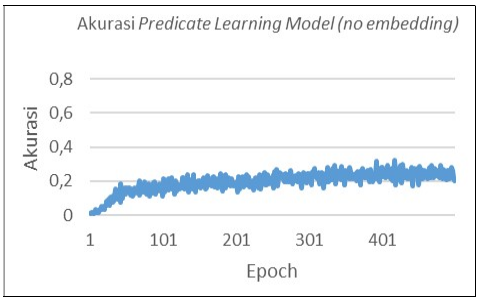
Gambar 12. Plot perubahan akurasi *entity learning model* dengan *embedding*



Gambar 13. Plot perubahan akurasi *entity learning model* tanpa *embedding*



Gambar 14. Plot perubahan akurasi *predicate learning model* dengan *embedding*



Gambar 15. Plot perubahan akurasi *predicate learning model* tanpa *embedding*

4.2.4 *Akurasi KEQA Dengan dan Tanpa Embedding*

Setelah diuji dengan 137 data pertanyaan, akurasi KEQA dengan *embedding* memiliki nilai akurasi tertinggi 0,8889 (88,89%), sedangkan KEQA tanpa *embedding* memiliki nilai akurasi tertinggi 0,8815 (88,15%). Dibandingkan dengan penelitian sebelumnya [5], penggunaan TransP memiliki nilai akurasi yang lebih tinggi. KEQA dengan TransE memiliki akurasi 0,754 (75,40%), KEQA dengan TransH memiliki akurasi 0,749 (74,90%), dan KEQA dengan TransR memiliki akurasi 0,753 (75,30%). Namun perlu diperhatikan bahwa jumlah *triple* dan jumlah pertanyaan yang digunakan untuk penelitian ini dan penelitian sebelumnya berbeda, dimana [7][5] menggunakan keseluruhan data FB2M dan SimpleQuestions, dan penelitian ini menggunakan jumlah yang lebih kecil.

Walaupun akurasi *entity learning model* dan *predicate learning model* dari KEQA tanpa *embedding* kecil, namun akurasi akhir yang diperoleh masih tinggi. Hal ini disebabkan oleh dua hal. Pertama, *embedding* digunakan oleh KEQA ketika menentukan kandidat dengan skor terkecil, tetapi karena KEQA mampu untuk memilih *triple* kandidat yang benar sebelum *scoring* dilakukan, maka nilai prediksi *embedding* mempunyai pengaruh kecil terhadap akurasi KEQA. Kedua, jumlah *triple* yang digunakan tidak terlalu besar, sehingga jika terdapat hasil yang benar pada kandidat yang dipilih sebelum proses *scoring*, maka kemungkinan besar pertanyaan dapat dijawab dengan benar.

4.2.5 *Parameter Scoring*

Untuk menemukan *triple* kandidat yang paling dekat dengan hasil prediksi, digunakan rumus untuk *scoring* seperti pada Persamaan 1. Pada rumus tersebut, terdapat 4 parameter yang digunakan untuk menyamaratakan *weight* dari empat aspek yang berbeda. Dari ini, dicari kemungkinan nilai parameter yang dapat menghasilkan akurasi tertinggi. Kemungkinan terbaik dicari dengan mencoba semua kemungkinan dimulai dari nilai 0 sampai 3, dengan interval 0.25. KEQA dengan *embedding* akan mendapat akurasi terbaik dengan 10 kombinasi parameter sesuai pada Tabel 1, sedangkan KEQA tanpa *embedding* akan mendapat akurasi terbaik dengan 539 kombinasi parameter yang berbeda.

Pada kemungkinan parameter terbaik untuk KEQA dengan *embedding*, ditemukan bahwa nilai parameter β_1 dan β_2 seringkali lebih tinggi daripada nilai parameter β_3 dan β_4. Hal ini disebabkan karena β_1 dan β_2 merupakan parameter yang mengatur pengaruh dari nilai *embedding* terhadap skor yang diberikan. Semakin tinggi nilai β_1 dan β_2 maka semakin tinggi pengaruh nilai *embedding*. Maka dari itu, untuk KEQA dengan *embedding* lebih baik untuk meningkatkan nilai parameter β_1 dan β_2 untuk meningkatkan keefektifan *embedding* yang digunakan.

Tabel 1. Kombinasi Parameter Scoring dengan Hasil Terbaik pada KEQA dengan Embeddding

Akurasi	β_1	β_2	β_3	β_4
88,89%	3,0	2,5	1,0	2,5
	1,5	1,0	0,5	1,5
	3,0	0,5	0,5	1,5
	2,0	1,5	0,5	1,5
	2,5	1,0	0,5	1,5
	2,0	1,0	0,5	1,5
	3,0	1,5	0,5	1,5
	2,5	1,5	0,5	1,5

	3,0	1,0	0,5	1,5
	2,5	2,5	1,0	2,5

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil pengujian, dapat ditarik beberapa kesimpulan sebagai berikut:

- Hasil akhir dari TransP adalah nilai Mean Rank 5.390,25 dan HIT10 0,285.
- Akurasi terbaik dari KEQA dengan *embedding* adalah 0,8889 (88,89%) dan tanpa *embedding* adalah 0,8815 (88,15%)
- Waktu *training* yang dibutuhkan TransP untuk memproses keseluruhan data FB2M (*training* dan evaluasi) adalah kira-kira 135 hari. Nilai ini bisa berubah tergantung dari kemampuan mesin yang digunakan.
- Jumlah *triple* yang kecil dan jumlah *entity* yang banyak akan mengurangi keefektifan dari TransP, karena TransP hanya memperoleh sedikit informasi yang bisa dipelajari.
- Proses pemilihan *triple* kandidat pada KEQA sangat berpengaruh pada akurasi KEQA. Dengan jumlah *triple* yang kecil dan jumlah *entity* yang besar, jika terdapat hasil yang benar pada kandidat, kemungkinan besar pertanyaan dapat dijawab dengan benar.
- Jika menggunakan *embedding* pada KEQA, sebaiknya tingkatkan nilai parameter β_1 dan β_2 pada *scoring* untuk meningkatkan pengaruh *embedding*.
- Jika dataset yang digunakan kecil, sebaiknya digunakan KEQA tanpa *embedding* karena perbedaan akurasi yang tidak terlalu besar.

5.2 Saran

Setelah meninjau keseluruhan penelitian ini, beberapa saran yang bisa diberikan adalah:

- Menggunakan dataset yang lebih besar (terkhususnya jumlah *triple*). Selain dapat meningkatkan Mean Rank dan HIT10 pada model *embedding* yang digunakan, juga dapat meninjau lebih lanjut keefektifan *embedding* dan *scoring* pada KEQA.
- Mencoba metode lain untuk memilih *triple* kandidat pada KEQA seperti *Named-entity Recognition*, untuk semakin meningkatkan akurasi KEQA.
- Menggunakan *model embedding* yang lebih simpel pada dataset yang kecil, untuk mengurangi waktu pemrosesan.

6. DAFTAR PUSTAKA

[1] Bordes, A., Usunier, N., Chopra, S., & Weston, J. 2015. Large-scale Simple Question Answering with Memory Networks. *ArXiv, abs/1506.02075*.

[2] Bordes, A., Usunier, N., García-Durán, A., Weston, J., & Yakhnenko, O. 2013. Translating Embeddings for Modeling Multi-relational Data. *NIPS*. URI=https://dl.acm.org/doi/10.5555/2999792.2999923

[3] Dubey, M., Banerjee, D., Chaudhuri, D., & Lehmann, J. 2018. EARL: Joint Entity and Relation Linking for Question Answering over Knowledge Graphs. *International Semantic Web Conference*. URI=https://doi.org/10.1007/978-3-030-00671-6_7

[4] Hochreiter, S., & Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation, 9*, 1735-1780. URI=https://doi.org/10.1162/neco.1997.9.8.1735

[5] Huang, X., Zhang, J., Li, D., & Li, P. 2019. Knowledge Graph Embedding Based Question Answering. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. URI=https://doi.org/10.1145/3289600.3290956

[6] Lin, Y., Liu, Z., Sun, M., Liu, Y., & Zhu, X. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. *AAAI*. URI=https://dl.acm.org/doi/10.5555/2886521.2886624

[7] Lukovnikov, D., Fischer, A., Lehmann, J., & Auer, S. 2017. Neural Network-based Question Answering over Knowledge Graphs on Word and Character Level. *Proceedings of the 26th International Conference on World Wide Web*. URI=https://doi.org/10.1145/3038912.3052675

[8] Medhi, S., & Baruah, H. 2017. Relational database and graph database: A comparative analysis. *Journal of Process Management. New Technologies*, *5*(2), 1–9. URI=https://doi.org/10.5937/joupроman5-13553

[9] Pan, J.Z., Vetere, G., Gómez-Pérez, J.M., & Wu, H. 2017. Exploiting Linked Data and Knowledge Graphs in Large Organisations. *Springer International Publishing*. URI=https://doi.org/10.1007/978-3-319-45654-6

[10] Paulheim, H. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web, 8*, 489-508. URI=https://doi.org/10.3233/SW-160218

[11] Singhal, A. 2012. *Introducing the Knowledge Graph: things, not strings*. Retrieved December 12, 2019, from Google Official Blog: https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html

[12] Wang, Z., Zhang, J., Feng, J., & Chen, Z. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. *AAAI*. URI=https://dl.acm.org/doi/10.5555/ 2893873.2894046

[13] Zeng, P., Tan, Q., Meng, X., Zhang, H., & Xu, J. 2018. Modeling Complex Relationship Paths for Knowledge Graph Completion. *IEICE Trans. Inf. Syst., 101-D*, 1393-1400. URI=https://doi.org/10.1587/transinf.2017EDP7398

[14] Zheng, W., Yu, J.X., Zou, L., & Cheng, H. 2018. Question Answering Over Knowledge Graphs: Question Understanding Via Template Decomposition. *Proc. VLDB Endow., 11*, 1373-1386. URI=https://doi.org/10.14778/3236187.3236192