Deep Learning for Knowledge Graph Completion

Nick Joodi[†], Kevin Jesse[†], Cesar Bartolo-Perez[†], Doug Sherman[†]
[†] *University of California - Davis*

November 30th, 2017

Abstract

Contents

1 Introduction

With the advent of Google's knowledge Graph, Wikidata, Freebase, and many others, the representation of data in the form of a knowledge base or ontology is affecting our lives everyday. Data in this format has been used to power advanced reasoning systems, chat bots, and in general, Artificial Intelligence. However, these knowledge bases are many times incomplete. To counter this, knowledge graph completion has been researched extensively over recent years.

Knowledge graph completion is the act of inferring new "facts" over an existing knowledge base. To represent a knowledge base as a knowledge graph, one can treat the concepts/entities/objects within the knowledge base as nodes, and the relations/predicates between these concepts as edges. After applying this conversion, the knowledge will be a large interlinking web of objects relating to one another, I.e a knowledge graph. A fact, in this case, is an instance in a knowledge graph where two objects are linked by a predicate. The common notation for this fact, or triplet, is the following: (s,p,o) where s and o are nodes in the graph, and p is the edge. Given two objects, knowledge graph completion gives us the ability to predict, with a measurable certainty, whether or not a triplet exists.

There have been a variety of techniques used to perform knowledge graph completion: translation, logically based, and deep learning techniques. This paper will focus on applying deep learning to this problem domain. The two neural network architectures that we will apply are the neural tensor network and the multi layered perceptron.

The neural tensor network, introduced by Socher et. al, has been shown to provide intriguing results. Since its emergence, it has been used as a baseline to evaluate the newer architectures in the field. This model incorporates a tensor layer in addition to the standard components of a neural network, to express the relationship between two entities. More over, this work also incorporated the useful approach in representing each entity as a composition of its word embeddings. They've shown, that using pretrained word embeddings enhances the performance of the overall model.

Conversely, a more standard, simpler approach to this problem is the use of a multi layered perceptron. [citation] has shown comparable results to the more computationally intensive neural tensor network.

This paper has the following outline. In the next section, we will cover the methods and assumptions that we used to build and trained our models, followed by a section describing our results, then a discussion, and then a conclusion to summarize as well as state potential future work.

2 Methods

Multilayer Perceptron Model (MLP)

2.0.1 Data Formatting

The dataset we considered was queried from the WikiData, a free opensource knowledge base that acts as the central storage device for its sister projects including Wikipedia, Wikivoyage, and Wikisource [**?**]. Wikidata offers a SPARQL type queries for obtaining data on a variety of sources. We queried data regarding the lineage of many famous people in history including: Frederik, Crown Prince of Denmark, George Windsor, Earl of St Andrews, and Shigeko Higashikuni. The relations we considered were if one person, or entity, was the Father, Mother, Spouse, Sibling, or Child of another. See Figure 1 for an example of the query we used to obtain our dataset.

```
SELECT ?human ?humanLabel ?spouse ?spouseLabel
?child ?childLabel ?sibling ?siblingLabel
?father ?fatherLabel
?mother ?motherLabel
WHERE {
  ?human wdt:P31 wd:Q5.
  ?human wdt:P26 ?spouse.
  ?human wdt:P40 ?child.
  ?human wdt:P3373 ?sibling.
   ?human wdt:P22 ?father .
   ?human wdt:P25 ?mother .
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
LIMIT 10000
```

Figure 1: SPARQL query used to obtain our raw data from Wikidata

The raw data is a table of values that lists each person and their Father, Mother, spouse, sibling, and child; leaving an entry empty if unknown. We processed this table into positive triplets of the form `(Q193752, P25, Q229279, 1)` where `Q193752` and `Q229279` are a unique encoding of the target and related entities respectively, `P25` represents one of the relations, or predictes, and `1` indicates that this statement is true. In english this triplet implies that `Q229279` is the `P25` of `Q193752`. These positive facts represent all the data we

can pull directly from the knowledge graphs. However, to increase the robustness of our data, we can create negative facts as well. For example, since we know that Cleopatra VII Philopator, the last active pharaoh of Ptolemaic Egypt, was the daughter of Ptolemy XII Auletes, we can conclude she was not the daughter of Richard Nixon. Thus we can extend our data to contain these negative triplets of the form `(Q1,P1,Q2,-1)`. In all, we were able to produce 37395 positive triplets and 37391 negative triplets from our raw data.

In an effor to exploit the uniqueness in the names for each of our entities, we used word embeddings of each of the words in a name to generate entity embeddings. We used the Fasttext model to generate word embeddings from semantic anaylsis across Wikipedia. Fasttext is a library that represents words as bags of character n-grams for fast and efficient classification [**?**, **?**]. Thus, we used Fasttext to obtain 300×1 word embeddings of each of the unique words found in all of our entitie's names. For example, the name "Charles Stuart, 1st Earl of Lennox", we found an 300 dimensional embedding vector for Charles, Stuart, 1st, Earl, of, Lennox. Once each of the words had embeddings, we could construct entity embeddings for each of our entities. The method we used for this was to aggregate the word embeddings by taking the mean across each of the words within an entity name. For the MLP model, if Fasttext could not produce a valid embedding for a given word, then that word was ignored. Moreover, if an entity consisted entirely of unknown words, then that entity was thrown out. We did this to preserve the integrity of our input data because in the standard MLP model we would not be training these embeddings. However, for more complex models, another suitable technique is to randomize the embeddings for these entities, or all entities, to guarantee a more representative dataset. Overall, only about 6.4% of the entities had no suitable embedding. Once we encoded each entity with their embedding we produced a dataset where each row contains the numeric representation of a triplet. Hence, for a triplet given by `(EntityA,Pred2,EntityB,1)`, where the embedding for `EntityA` and `EntityB` is $[a_1, a_2, \cdots, a_{300}]^T$ and $[b_1, b_2, \cdots, b_{300}]^T$ respectively, the associated row in our data set is given by

$$\left[a_1, a_2, a_3, \cdots, a_{300}, b_1, b_2, \cdots, b_{300}, 0, 1, 0, 0, 0\right]$$

where the last 5 columns represents the truth values for each of the 5 predicates (Father, Mother, Spouse, Sibling, Child).

One we had a comprehensive dataset to train on, we designed our predictive model. We implimented a Multiyaler Perceptron (MLP) model using the RSNNS package following the Stuttgart Neural Network Simulator[**?**]. Figure 2 shows a general look at the architecture for our MLP design. We tested a few variations of the MLP model to determine the suitable parameters for our dataset; including a variable number of layers, activation functions, and nodes per layer. Moreover, we tuned the hyperparameters such as the learning rate and momentum term for backpropagation. We will reference Figure 2 as we discuss the effects of the different architectures.

Where we have used standard backpropagation and backpropagation with momentum methods in our simulation models.

Different MLP architectures are built, varying number of hidden layers, nodes and iterations in order to improve the accuracy and precision of our model.

Weighted SSE by number of iterations of the training and tetsing sets in addition to ROC curves will provide the references in our model architecture selection.

3 Results

In order to prevent overfitting in our model, we have compared our model with 100 and 50 interations (Figure 3). We can notice that under 100 iterations, the training error drops to a weigthed SSE of almost 500. However, our testing error, start increasing after 30 itearions aproximately. From this results, we can infer that we have reached an overfitting in our model and a smaller number of iterations are enougth. In that sense, we have run our model with 50 iterations, avoiding the increase in the SSE.

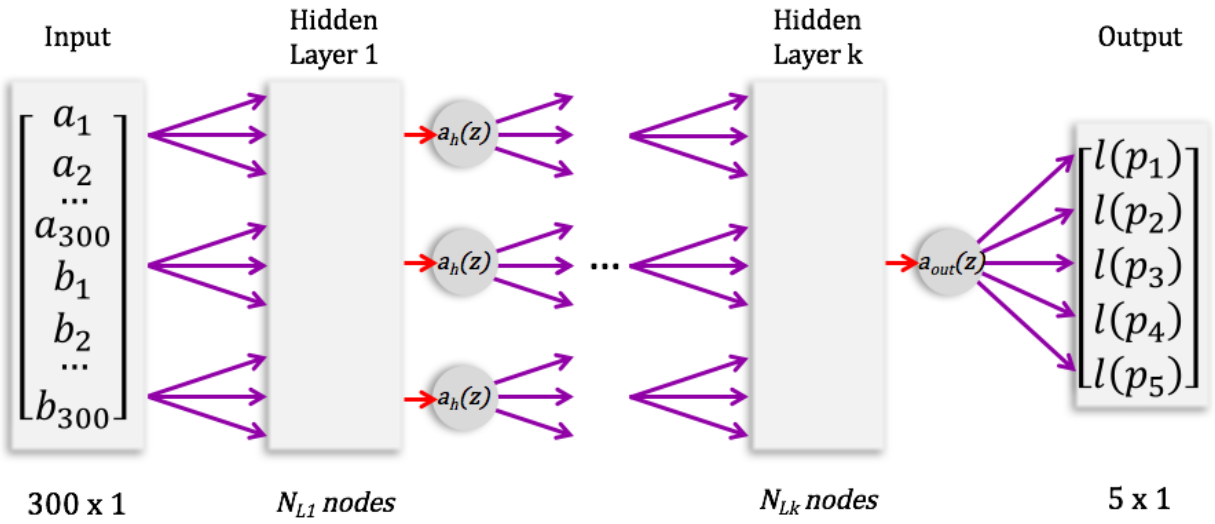

Figure 2: A general look at the Neural Network architecture used for a MLP model with k hidden layers with N_{L_i} nodes for the ith layer, a common activation function $a_h(z)$ for each layer, and a final output activation function $a_{out}(z)$. We have 600 features from our input, 1 for each dimension of both entity embeddings, and we are predicting the true/false value of our 5 output predicates. Note that $l(p_1)$ is the likelihood that the predicate p_1 is true.

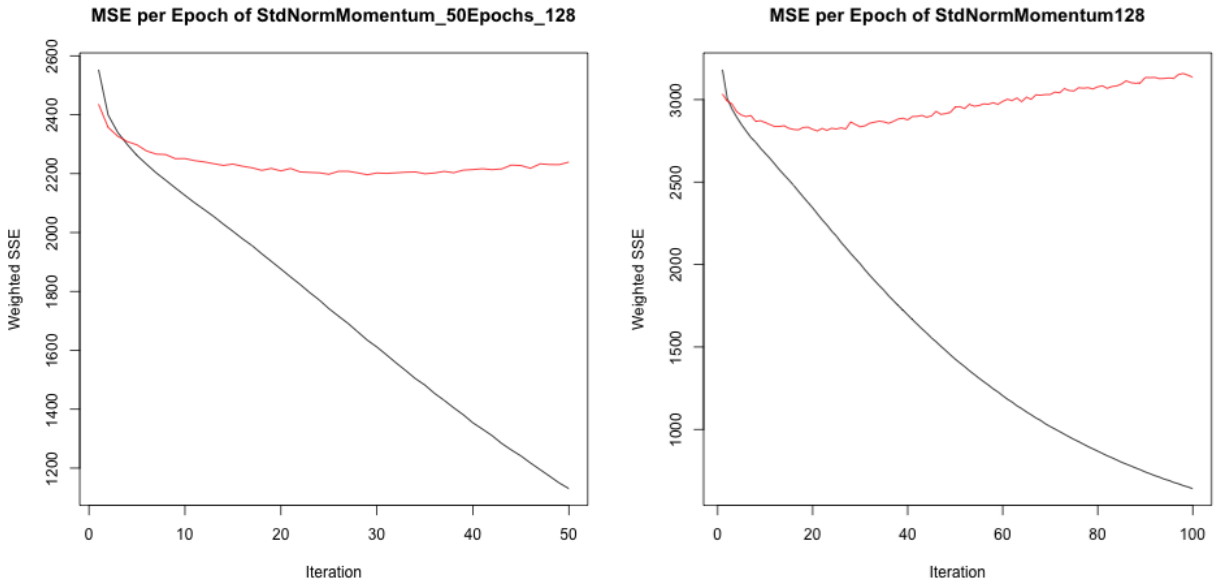

Figure 3: MSE at different iterations. The structure presents a single hidden layer with 128 nodes

In a second running of the model, we have added a new hidden layer with different number of nodes (16,64 and 128). Before 50 iterations, adding nodes to the second hidden layer, reduce the error in our model. However, after 50 iteration, the higher number of nodes, increase rapidly the overfitting of the model.

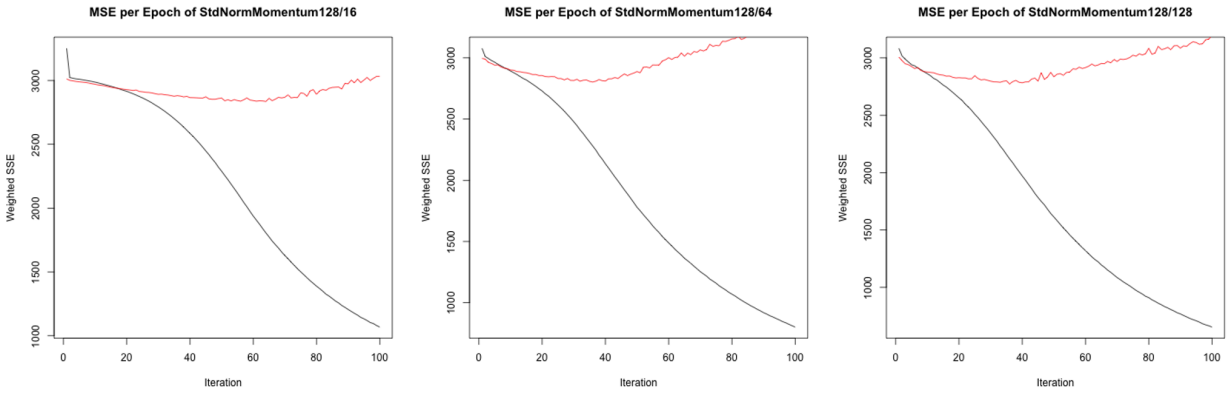

Figure 4: MSE for A NN with two hidden layers with different nodes at second hidden layer.

In a following set of tests, we compared different Neural Networks structures, increasing the number of hidden layers (hidden layers/nodes: 128, 128/16 and 128/128/16). Figure 4 shows that for the model with three hidden layers the training error is not decreasing considerably before 40 iteration. Suddently, the SSE in this NN structure drops abruptaly. The training error, shows that the error keep increasing at that range of iterations. This beahvior, can be attributed to an overfitting in our model.

Ths trend described above is similar for two hidden layers. For teh case of just one hidden layer with 128 nodes, the error does not drop abruptaly for the training set but we can see an increase in the error for after 30 iterations aproximately.

These results, make us suggest that only one hidden layer is enough for our MLP model, since there is not a big improvement in the reduction of error with more layers added and with the benefit of less computational resources are required. This decision is confirmed comparing their ROC curves for each NN structure.

4 Discussion

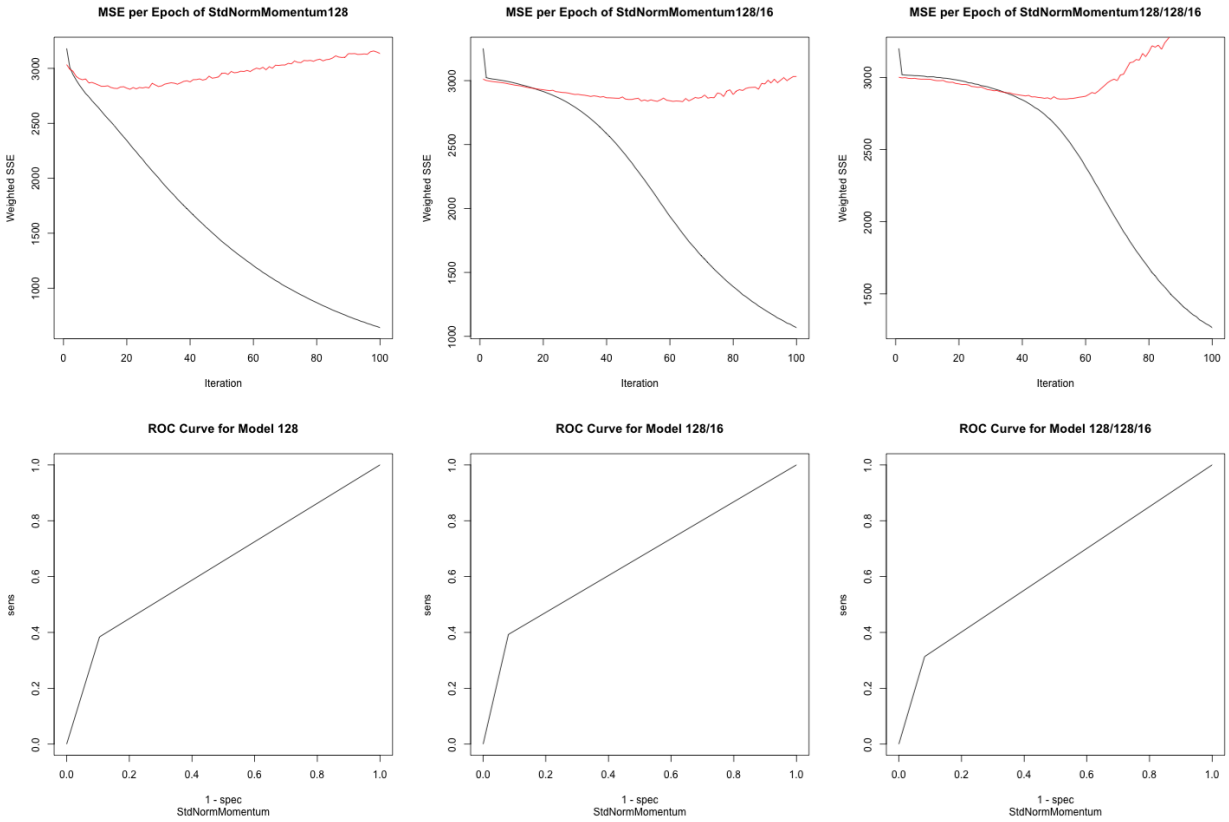

Figure 5: MSE and ROC curves for different levels of hidden layers