# APPLICATIONS OF DEEP LEARNING AND PARALLEL PROCESSING FRAMEWORKS IN DATA MATCHING

by

Bernardo Jose Najlis, Systems Analyst, Universidad CAECE, 2008

A Major Research Paper

presented to Ryerson University

in partial fulfillment of the requirements for the degree of

Masters of Data Science

In the Program of

Data Science and Analytics

Toronto, Ontario, Canada, 2018

**AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A MAJOR RESEARCH PAPER (MRP)**

I hereby declare that I am the sole author of this Major Research Paper. This is a true copy of the MRP, including any required final revisions, as accepted by my examiners.
I authorize Ryerson University to lend this MRP to other institutions or individuals for the purpose of scholarly research
I further authorize Ryerson University to reproduce this MRP by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.
I understand that my MRP may be made electronically available to the public.

BERNARDO JOSE NAJLIS

APPLICATIONS OF DEEP LEARNING AND PARALLEL PROCESSING FRAMEWORKS IN DATA MATCHING

Bernardo Jose Najlis

Masters of Science 2018

Data Science and Analytics

Ryerson University

Most of Data Science research work assumes a clean, deduplicated dataset as a pre-condition. In reality, 80% of the time spent in data science work is dedicated to data deduplication, cleanup and wrangling. Not enough research papers focus on data preparation and quality, even though it is one of the major issues to apply Data Science. The research subject of this paper is to **improve Data Matching techniques on multiple datasets containing duplicate data using parallel programming and Deep Neural Networks. Parallel programming frameworks** (like MapReduce, Apache Spark and Apache Beam) can dramatically increase the performance of computing pair comparisons to find potential duplicate record matches, due to $O(n^2)$ complexity of the problem. **Deep Neural Networks** have shown great results to improve accuracy on many traditional machine learning applications. The problem and solution researched are of general applicability to multiple data domains (healthcare, business).

# ACKNOWLEDGEMENTS

**To Dr. Kaamran Raahemifar for his continuous support during the elaboration of this Masters Research Paper**

**To Dr. Ceni Babaouglu for providing insight and guidelines into the basic Data Science Research Methodology followed during this research**

# DEDICATION

**To my family, more specially my partner and mother, for their endless love, patience and support during the time invested on the elaboration of this research work.**

# Contents

## LIST OF TABLES



## LIST OF FIGURES



## LIST OF ILLUSTRATIONS

N/A

## LIST OF APPENDICES

N/A

## ABSTRACT

Data Matching (also known as Record Linkage, Entity Resolution and Duplicate Detection) is a prevalent problem in all disciplines that use data: if a data record (database or data frame row, line on a data file) is a data representation of an entity that exists in the real world (i.e. a person, object or event) how can we determine when and which multiple data records represent the same real-world entity? This is the focus of Data Matching: by applying different techniques, identify a unique set of records and how duplicates are related to each other by representing the same entity.

The aim of the project is to use Deep Neural Networks in the context of a traditional data matching pipeline, to improve accuracy in the classification of matches, non-matches and potential matches.

For our model we create a Siamese Bidirectional Long-Short Term Memory Network optimized on Euclidean distance, that calculates a record similarity score based on the distance of the vector representation of each input record, generated by each side of the network.

Our results show an accuracy of 96.94% using random sampled validation sets and 97.93% accuracy on random sampled test sets.

This indicates that the use of this type of network topology can be effectively used as an alternative to traditional and more complex Data Matching / Record Linkage pipelines.

## INTRODUCTION

Data Matching (also known as Record Linkage, Entity Resolution and Duplicate Detection) is a prevalent problem in all disciplines that use data: if a data record (database or data frame row, line on a data file) is a data representation of an entity that exists in the real world (i.e. a person, object or event) how can we determine when and which multiple data records represent the same real-world entity? This is the focus of Data Matching: by applying different techniques, identify a unique set of records and how duplicates are related to each other by representing the same entity.

Data Matching applications in the real world are diverse: from census records reconciliation (one of its first and still major applications) to health records consolidation (where multiple medical institutions keep records of the same individual across isolated systems) and also in business environments (where different systems like CRMs, ERPs and Loyalty are not integrated, customer records store different information), most major systems face the problem of data duplication. Data Matching is also central to the newly nascent world of Data Science, where data accuracy is key to extract valid analytics and metrics out of it. Most of Data Science research work is performed under the assumption of a clean, pristine, de-duplicated and accurate data set; even though this situation is not very common outside of the world of academic curated data sets. Thus, having an array of techniques that approximate real world data sets to the ones that are usable for statistical analysis and machine learning is of utmost importance.

Data Matching and Record Linkage are disciplines known for several decades know, that started with the first research papers from 1960's and 70s looking at applying this to census data and medical records, where this issue is more prevalent. As new machine learning techniques like Deep Learning are now available to simplify data science work and also improve accuracy, we would like to research the impact of using these in this field.

The research work reported in this paper uses modern neural network techniques, more specifically Deep Learning, LSTM (Long-short Term Memory) and Siamese networks, to create a model that given a pair of records that can be a potential data match (i.e. represent the same real-world entity), it returns a highly accurate result.

## LITERATURE REVIEW

Our literature review spanned over both academic textbooks that cover the subject of Data Matching, as well as very recent research and also reference/foundational research papers.

The main reference and consultation book used throughout our research was **"Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution and Duplicate Detection"** (Christen, 2012). This covers all the phases on the traditional data matching approach (Data pre-processing, Indexing/Blocking, Record pair comparison, Classification, Matching, Evaluation and Clerical Review) and all traditional machine learning techniques used in each of these steps. The second major textbook used as reference was **"Entity Resolution and Information Quality"** (Talburt, 2011). This book presents also a very traditional machine learning approach to the subject of Entity Resolution and provides the main concepts and principles in this research area.

As for peer reviewed research papers, they fall under two major areas:

### Modern approaches to Entity Resolution (i.e. using parallel processing frameworks, using statistical techniques)

- ***"Performance Comparison of Apache Spark and Tez for Entity Resolution"*** (Haq, 2017)**.** This paper applies two of the major parallel programming frameworks (Apache Spark and Tez/Hive) for Entity Resolution and compares their performance. It provides a very comprehensive and descriptive list of all technical tools and techniques used.

- ***"FEBRL – Freely Extensible biomedical record linkage"*** (Peter Christen, 2005)**.** This paper provides details on many of the traditional statistical and machine learning techniques used on Entity Resolution. It is also the project that provides the data set used in this project.

- ***"Toward building end-to-end entity matching solutions"*** (Christopher, 2018)**.** This paper provides a very descriptive and complete dissertation on all the areas involved in the development of an entity matching solution on cloud technologies that uses a parallel programming framework.

- ***"A Bayesian Approach to Graphical Record Linkage and De-duplication"*** (Rebecca C. Steorts, 2015)**.** In this paper the authors propose the use of unsupervised machine learning techniques and graph theory to detect duplicate record probabilities and then match them.

- ***"Efficient Parallel Set-Similarity Joins using MapReduce"*** (Rares Vernica, 2010) This paper is the first one using the Hadoop MapReduce programming framework to the problem of Entity Resolution. It was consulted more for reference and historical value and as a stepping stone to understand the application of parallelization applied to ER.

- ***"Duplicate Detection on GPUs*** (Benedikt Forchhammer, 2011)**.** This paper presents the novel approach of using GPUs (graphical processing units) as a mean to increase efficiency when computing entity resolution in large data sets, due to the $O(n^2)$ nature of the problem.

- ***"Parallel Entity Resolution with Dedoop"*** (Lars Kolb, 2012)**.** This is the first most popular paper to apply the use of parallel programming frameworks (Hadoop MapReduce) in the area of Entity Resolution.

- ***"Data Partitioning for Parallel Entity Matching"*** (Toralf Kirsten, 2010). This paper focuses on the sub-area of data partitioning (to reduce the $O(n^2)$ nature of the Entity Resolution problem). Even though it is a bit dated (all programming was done using Java in the pre-Hadoop era), concepts and ideas for this are still valid to understand the value of partitioning.

- ***"Learning Blocking Schemes for Record Linkage"*** (Matthew Michelson, 2006)**.** This paper presents the novel approach of using statistical learning techniques to infer partitioning/blocking sets from the data itself.

- ***"Record Linkage"*** (Larsen, 2013). This thesis reviews the application of different blocking techniques to different series of datasets to find optimal results using data and concepts from the FEBRL paper.

- ***"An efficient spark-based adaptive windowing for entity matching"*** (Demetrio Gomes Mestre, 2017). This is one of the most recent papers in the subject on Entity Resolution, elaborating on the importance of parallel processing for entity resolution on large data volumes, and specifically focused on spark cluster nodes workload distribution aimed for this type of problem.

## Deep Neural Networks

- ***"Learning Text Similarity with Siamese Recurrent Networks"*** (Paul Neculoiu, 2016)**.** This paper explores the use of deep neural networks to obtain text similarity metrics. It uses bi-directional LSTMs (Long Short Term Memory) with a Siamese architecture.

- ***"Siamese Recurrent Architectures for Learning Sentence Similarity"*** (Thyagarajan, 2016)**.** This paper also focuses on the use of LSTM networks for labeled data comprised of pairs of variable-length sequences, that assesses semantic similarity between sentences.

- ***"Long Short-Term Memory"*** (Sepp Hochreiter, 1997)**.** This is the foundational paper on LSTM, so it was covered and studied for reference purposes.

- ***"Learning to Forget: Continual Prediction with LSTM"*** (Felix A. Gers, 2000)**.** Another foundational paper in the subject of LSTMs, consulted mainly for reference purposes in the understanding of Long Short-Term Memory networks.

## EXPLORATORY DATA ANALYSIS

The datasets were obtained from one of the most popular Record Linkage projects, referenced by almost every other paper reviewed: FEBRL (Peter Christen, 2005). The complete dataset is composed of several files: as the goal is to find linkage/duplicates between two sets, there are two subsets (dataset A and dataset C) and also there are 4 groups with different numbers of records each (1,000, 2,500, 5,000 and 10,000). Each file is in CSV (comma separated value) text format and includes the column names header in the first row. All files have the same schema (same number of columns and at the same position).

Here is a description of the file schema, that can serve as data dictionary:

| Col Name/ Feature | Description | Data Type | Null / Blanks | Min | Max | Average |
|---|---|---|---|---|---|---|
| **rec_id** | Unique identifier for each row | Integer | NO | 0 | 49,991 | 24,995.5 |
| **given_name** | First name of the patient. | String | YES | N/A | N/A | N/A |
| **surname** | Last name of the patient. | String | NO | N/A | N/A | N/A |
| **street_number** | Address number of Patient's address. | Integer | YES | 0 | 12,999 | 77.18 |
| **address_1** | Street name of patient's address. | String | NO | N/A | N/A | N/A |
| **address_2** | Additional information (apt number, others) of patient's address. | String | YES | N/A | N/A | N/A |
| **suburb** | Suburb name of patient's address. | String | YES | N/A | N/A | N/A |
| **postcode** | Postal Code of patient's address. | Integer | YES | | | |
| **state** | State name of patient's address. | String | YES | N/A | N/A | N/A |
| **date_of_birth** | Date of birth of patient. Codified as integer (4 first digits correspond to year, next two correspond to month, last two correspond to day). Months and days have leading zeroes if applies. | Integer | YES | 18971218 | 19999017 | 19491159 |
| **age** | Age of patient. | Integer | YES | 2 | 98 | 29.16 |
| **phone_number** | Phone number of patient. Format is: two digits for area code (with leading zeroes) and 8 digits for phone number. | String | YES | N/A | N/A | N/A |
| **soc_sec_id** | Social Security ID for patient. 7 digits | Integer | NO | 1000606 | 9998137 | 5531936.39 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **blocking_number** | Number of block assigned for comparison. This can be used as a label when applying blocking strategies. | Integer / Categorical | NO | 0 | 9 | 4.5 |

*Table 1 - FEBRL dataset schema*

To understand the nature of the dataset we will be working with during our project, we performed exploratory data analysis on the 10,000 records dataset.

As the purpose of this dataset is to find duplicates within the rows (patients), there is no sense on charting bi-variate exploratory data analysis: correlations between variables are of no use to solve our problem. All of the exploratory data analysis charts are based on the nature of the dataset (number of rows/patients per distinct features) and their distributions.

## Number of distinct values per feature

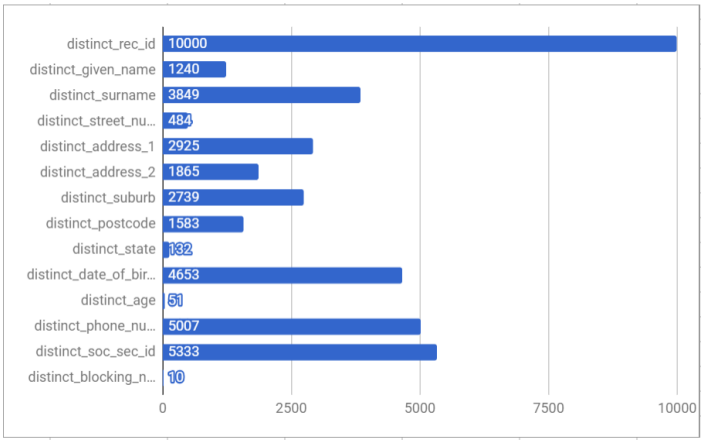


*Figure 1 - Number of distinct values per feature*

All records have a unique rec_id (as expected) and there are about 5,000 duplicate phone_numbers and soc_sec_id. This means those two variables are great candidates to be used as unique identifiers, together with date_of_birth (4,653 unique values) and surname (3,849 unique values).

## Number of null / missing values per feature (only for features that are nullable)

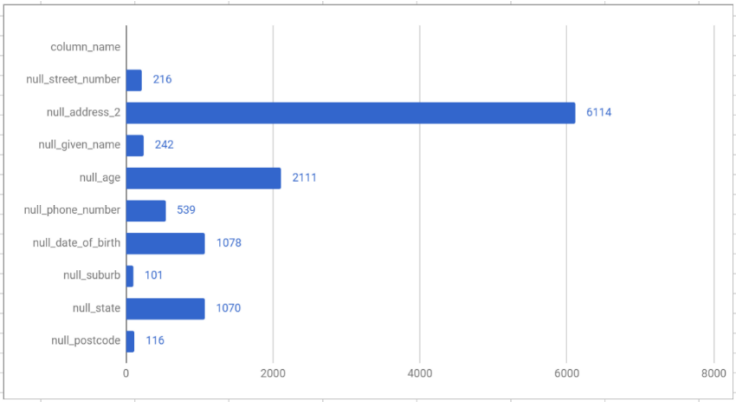


*Figure 2 - Number of null values per feature*

More than half of the records do not have address_2 data, and about 20% of the dataset has no age (although some of them may be inferred from the date of birth, as only 1,078 rows do not have dob).

All other fields are at less than 10% of missing / null values data.

## Number of patients per blocking number

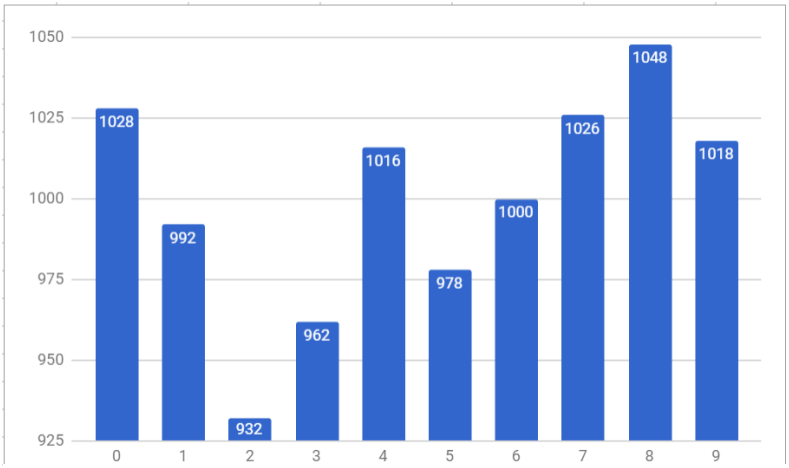


*Figure 3 - Number of patients per block*

There are 10 blocks (0 through 9) and all have about 1,000 records (max: 1048, min: 932) so we can consider this as a fairly balanced distribution set.

## Number of patients per age

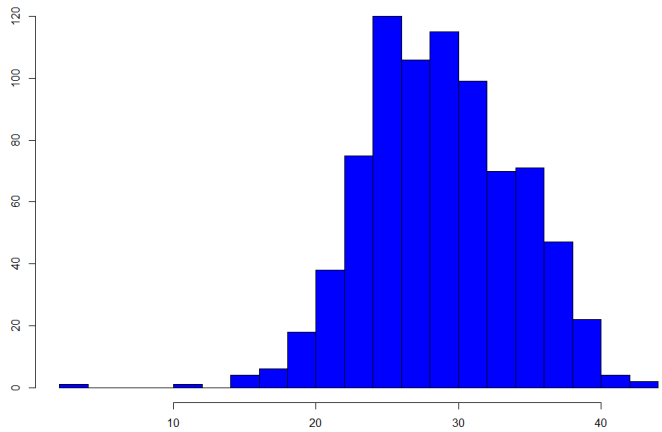


*Figure 4 - Number of patients per age*

All patients have ages in a reasonable range (min: 2, max: 98) with an average of 29.16 years old. As we can see in the histogram, this is a normally distributed set.

## Number of patients per Street Number

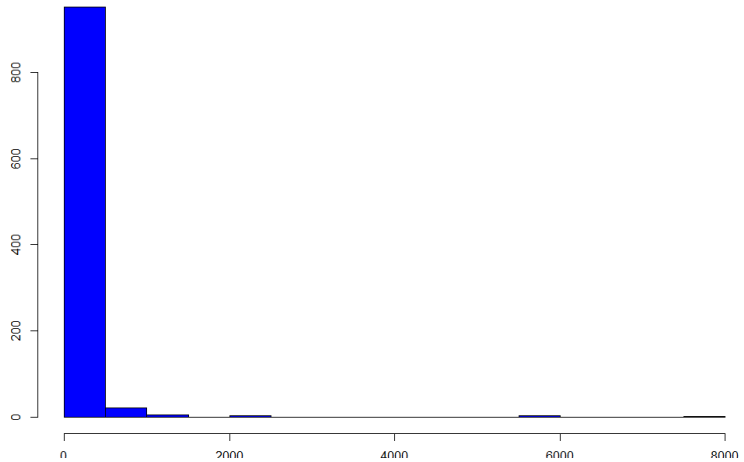


*Figure 5 - Number of patients per street number*

Although street number is not a traditional continuous variable, we decided to chart its histogram to understand if there is any correlation or bias in its distribution. Most of all address numbers have numbers less than 500, which indicates a very unbalanced set on this variable.

## Number of patients per Postal Code

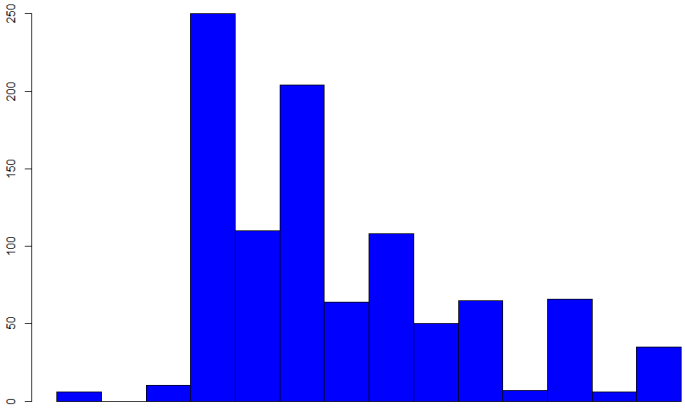


*Figure 6 - Number of patients per postal code*

Same as street number, postal code is more a categorical variable expressed with integers than a conventional continuous variable. Still its histogram shows a quasi-normalized distribution slightly skewed to the left.

## Number of patients per State

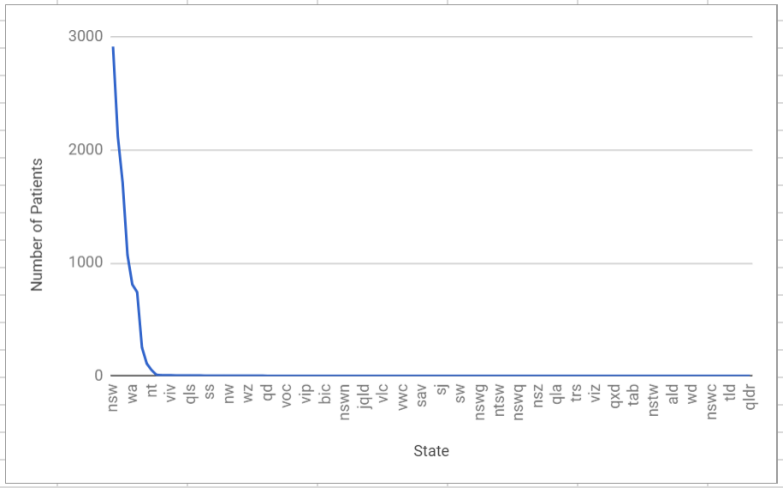


*Figure 7 - Number of patients per address state*

Same as with other variables, state is more a descriptive / categorical value. Its histogram shows that more than half of patients are located in 3 states, as the rest follows a very long tail distribution.

# Methodology and Experiments

## Aim of the Study

The aim of the project is to use Deep Neural Networks in the context of a traditional data matching pipeline, to improve accuracy in the classification of matches, non-matches and potential matches.

The first task is to modify the data transformation that happens before attempting to go into the matching process. Traditional Data Matching pipelines implement, as outlined in (Christen, 2012) describe the following steps:

1) Data pre-processing
2) Indexing
3) Record pair comparison
4) Classification
5) Matches / Non-matches / Potential Matches
6) Evaluation
7) Clerical Review (only for Potential Matches)

Our approach will modify the Record pair comparison step, to use a Deep LSTM Siamese network to evaluate text similarity using character based embeddings, as presented in (Thyagarajan, 2016). By looking at each record to compare as unique, fully concatenated string as opposite to traditionally comparing on a record field-by-field basis, we also simplify and potentially eliminate the need for) the Data pre-processing step.

## Data Pre-processing

As explained in the Exploratory Data Analysis section, the FEBRL dataset we are using is composed of multiple sets, so our methodology and experiments will be implemented across all the datasets (which contain different number of total and duplicate rows). The first activity in every data science project is data pre-processing: in this case we will transform the data set into the shape and schema required by our approach, to implement the Deep LSTM Siamese Network and compare it with the traditional classification algorithm used by FEBRL. As the comparison is performed at the record pair level, we have to transform a dataset that contains single records (original and duplicates) into a dataset that contains both **original and duplicate record fields in the same row**. Also, as described below, multiple datasets will be generated, each one containing different sets of fields, in order to perform independent experiments.

The FEBRL dataset groups duplicate records by using the rec_id integer value: all records that are part of the same decil (i.e.: 0 through 9, 10 through 19) are all duplicate records. By processing the records by their rec_id values, we can identify the original record as the one that is divisible by 10, and all other records in the same decil as its duplicates. Using this technique, we create a new transformed dataset with pairs of records: the record on the left is the original record (rec_id is divisible by 10) and the record on the right is a duplicate record (rec_id is in the same decil as the left record, but not divisible by 10). This generates n-1 records per decil (i.e.: if record 40 is the original, 41, 42 and 43 are duplicates; the dataset will have 3 entries with record 40 as the original on the left and records 41, 42 and 43 on the right as the duplicates). The idea is to process this dataset with the original record always on the left and

a duplicate record on the right, so the experiment can be trained in the data that is an original and a duplicate record.

This transformation has the following impact in the dataset:

- **Record count decreases**: For example, if the original dataset contains 1,000 records where half of them are duplicates, we will have 500 records now, each row containing both the original and duplicate record.
- **Schema changed**: Instead of having a single column for each field, we now have two columns: one for the value on the original record and another for the duplicate record.
- **Multiple datasets, depending on the experiment**: Some experiments (as it will be described later) will be run on a single field (i.e. name) , and others will require the complete set of fields (name, address, date of birth, phone, social security id).

The schema described in the Exploratory Data Analysis section must now be transformed from the one described in Table 1 into the following two schema:

| Col Name/ Feature | Description | Data Type | Null / Blanks | Min | Max | Average |
|---|---|---|---|---|---|---|
| **rec_id** | Unique identifier for each row | Integer | NO | 0 | 49,991 | 24,995.5 |
| **name** | Concatenation of first_name, blank space and surname field, to form the full name | String | YES | N/A | N/A | N/A |
| **name_duplicate** | Last name of the patient. | String | NO | N/A | N/A | N/A |

*Table 2 - Updated schema for analysis*

## Characteristics and Design of the Model

The approach selected for the experiment is based on a multilayer neural network with the following characteristics:

- **Input as Character embeddings**: we focus on the character values of the two input records we will compare, and create vector representations for them. This differs from other NLP approaches that use pre-trained word embeddings (like word2vec or glove) and create vectors for terms instead of characters. This is because our input space is not based on names and addresses, that will not exist in pre-trained models.
- **BLSTM layer (Bidirectional Long-short term memory):** LSTMs are a specific type of recurrent neural network. Where traditional RNNs have issues with long-term dependencies, LSTMs can remember information for long periods of time as their default behavior. More specifically, BLSTMs duplicate the first recurrent layer in the network and provide the input to the first layer, but a reversed copy of the same input to the second layer. This bidirectionality has been proven to improve the performance of LSTM for pattern recognition tasks, like the one we want to perform.
- **Temporal Averaging:** At the output of the BLSTM layer we average the output vectors to create an output with fixed dimension.
- **Dense Layer:** This is a typical Dense feed forward layer.

The output of these configuration creates an encoded vector representation of the input set of characters.

The additional characteristic to this is the **Siamese Network** arrangement. This consists of an exact duplicate of the network that was just described, and it is used on the second input record that we will be compared with the record fed to the first network. The final arrangement is done by connecting both networks based on a comparison function. In our case, this is a simple Euclidean cosine distance function. As we want to train the full network to recognize similarity between the two input character sets, we will optimize the loss function to reduce the distance for the two vector representations created by each of the networks.

The original Tensorflow code that we based our work on was originally created by Dhwaj Raj and available in Github at https://github.com/dhwajraj/deep-siamese-text-similarity

## Experimental Design

### Training / Validation / Testing dataset split

As with every data science series of experiments, we will be splitting the data into training and test data sets. The algorithm and experiments that allow to compare two records and determine they degree of similarity is not influenced by the distribution of the dataset, so we can use a random sample approach. We then will be using random sampling with a ratio of 0.7 / 0.1 / 0.2 (i.e.: 70% records used for training, 10% for validation and 20% will be used for testing). Training, validation and test sets have different purposes in our experiments: the training set will be used to optimize the weights, the validation set to alter network parameters (number and type of layers, hidden units, number of epochs, learning rate), and finally the testing set will be used to calculate accuracy.

In terms of algorithm validation, and due to the limited number of rows available on the dataset, cross-validation may be an option to calculate a more precise accuracy value on the experiments.

### Experimentation Factors

The network initially designed for experimentation has the following factors / parameters that will be altered to optimize its accuracy:

| Factor / Parameter | Description |
|---|---|
| **Dropout Keep Probability** | Used to control the dropout rate, controls percentage of elements to be randomly dropped out during training. |
| **L2 Regularization Lambda** | L2 regularization aims to reduce overfitting, by penalizing weights with large magnitudes. |
| **Hidden Units** | Number of units in the LSTM cell. |
| **Batch Size** | Number of records to process per epoch |
| **Number of Epochs** | Number of iterations over the training dataset |

*Table 3 - Experimentation Factors*

## Response Variable

To measure the results of the experiments with multiple factor values, we will use **accuracy**. The algorithm calculates accuracy as the number of row pairs that were correctly predicted as duplicates, over the total number of rows evaluated.

## Strategies of Experimentation

The main factors to experiment with, and the ones with the most expected positive correlation with accuracy are **Number of Epochs** and **Hidden Units**. Number of epochs increases the number of processing iterations to calculate the network weights; hidden units alters the structure of the network and increases the complexity of the network by adding more weights which can better approximate the target function. Both factors unfortunately have a direct impact on the processing time spent to calculate network weights. This means part of the experimentation strategies include adding more processing power (by resorting to external computing resources like cloud platforms) or time-boxing the network training by capping the accuracy after a certain threshold was reached.

The next factor that can be used during experimentation and expected to have an impact on accuracy is the **Batch Size**, as the number of records processed at a time directly influences the training process calculates weights and approximates the target function.

The rest of the factors (**L2 Regularization Lambda**, **Dropout Keep Probability**) are not expected to have a very significant impact on the accuracy.

With an understanding of how these factors impact the network, the main strategy will be to test them all independently, in order of expected correlation (Number of Epochs, Hidden Units, Batch Size, L2, Dropout) to find bands / ranges of values where accuracy reaches its maxima. By later combining factor by adding one factor at a time and experimenting with values in each of their found ranges, we aim to find how factor interaction may alter network accuracy.

In addition to experimenting with factor value ranges, multiple training iterations will be performed with the following datasets:

- 1000, 2500, 5000 and 10000 records only comparing patient name (first_name + last_name)
- 1000, 2500, 5000 and 10000 records comparing patient name and address
- 1000, 2500, 5000 and 10000 records comparing all patient fields

## Results

The first initial run on training data produced an accuracy of **0.978626**. As this is a very high and desirable number, investigation followed to prove that the dataset created was highly unbalanced: we were training the network on all positive matching cases with no negative cases created. This caused to review our approach to dataset manipulation before training. The decided approach was to create negative matching cases based on shifting existing matching records by one in the existing data frames.

With this new more balanced dataset, the training run produced the following numbers:

- **Training Accuracy**: 0.655714
- **Validation Accuracy**: 0.531
- **Test Accuracy**: 0.4995

Considering these as our baseline values, we will start the experimentation phase, as described in the previous section. These results used the following default settings:

| Parameter | Value |
|---|---|
| **Dropout keep probability** | 1.0 |
| **L2 Regularization lambda** | 0.0 |
| **Hidden Units** | 50 |
| **Batch Size** | 64 |
| **Number of Epochs** | 300 |
| **Embedding Dimension Size** | 300 |

*Table 4 - Iteration 0 parameter settings*

### Embedding Dimension Size

This parameter is determined by the size of our input: records have a maximum number of characters and in our data set (first name, last name and address), this is 103. All experiments will be performed with this value.

### Hidden Units

Hidden units is, according to our personal hypothesis, the factor that can influence model accuracy. The number of units have a direct correlation with the complexity and expressiveness the model can have to generate the target function.

Increasing the number of hidden units to 300 only created a marginal increase in training accuracy and no increase in validation or test accuracy:

- **Training Accuracy**: 0.668095
- **Validation Accuracy**: 0.531
- **Test Accuracy**: 0.4995

Further increasing the number of hidden units to 1,000 to a significantly higher time to train the network, also leading to a very marginal increase in training, test and validation accuracy:

- **Training Accuracy**: 0.673982
- **Validation Accuracy**: 0.5529

- **Test Accuracy**: 0.4997

From these results, we can conclude that keeping the number of hidden units does not influence accuracy beyond the default 50 units from our baseline values. As it has a very high impact on the training time, all of our next experiments will keep the Hidden Units value fixed to 50.

### Batch Size

The second parameter that we hypothesize will have a great impact is the batch size. By increasing the number of samples per batch, we get higher accuracy in the gradient calculation (also requiring more memory).

As the batch size is related to the number of rows in the training set, and we have 3630 rows in our set, we chose numbers divisible by that. Our first attempt is with 121 rows, and these are the results.

- **Training Accuracy**: 0.707071
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

Next iteration is with 605 rows per batch, and these are the results:

- **Training Accuracy**: 0.634527
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

From this we can conclude that 121 rows is the best value for our training data set.

### Dropout Keep Probability

This parameter can help normalizing the distribution of data we have as input to our network. It selects a random number of nodes to be dropped out of the network at each iteration. Many papers recommend using 0.5, so we will experiment with three values: 0.5, 0.75 and 0.25 to validate this approach.

For dropout keep probability = 0.5:

- **Training Accuracy**: 0.662075
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

For dropout keep probability = 0.75:

- **Training Accuracy**: 0.657484
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

For dropout keep probability = 0.25:

- **Training Accuracy**: 0.677686
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

An initial observation leads us to conclude that different values do not have an impact on our training accuracy. We will leave this as 0.25.

### L2 Regularization Lambda

Regularization is a common technique applied in Machine Learning in general to penalize coefficients and reduce potential overfitting. As our network includes an L2 regularization parameter in its layers, we want to take advantage of it and find the most optimal value.

$$Cost = Loss + \frac{\lambda}{2 * m} * \sum \|w\|$$

In all previous experiments, we had set L2 lambda to 0 (which means the regularization is not applied in the cost function). We will now experiment with the following values for L2 lambda: 0.01, 0.5 and 1.0.

For L2 lambda = 0.01:

- **Training Accuracy**: 0.665748
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

For L2 lambda = 1.0:

- **Training Accuracy**: 0.665748
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

For L2 lambda = 0.5:

- **Training Accuracy**: 0.678604
- **Validation Accuracy**: 0.0405405
- **Test Accuracy**: 0.0260116

Different values of lambda for L2 regularization do not seem to have an impact on accuracy. We will leave L2 regularization off by setting its value to 0.0.

## Conclusion

The main conclusion is that a Deep Neural Network can effectively be used as a method for Data Matching / Record Linkage. By replacing a more complex pipeline, each one of those steps originally listed in are here revised in the light of our findings:

1) **Data pre-processing**: still required but mostly to provide a balanced and rich training data set, as opposite to custom methods for data cleanup.
2) **Indexing**: Still required if a high volume of records is required, as it optimized the number of pairs that would be fed to the network.
3) **Record pair comparison**: This is the core step that the network is responsible for. Instead of using traditional clustering or classification approaches, that require heavy tuning and are field specific, our approach is more generalizable (independent of language, grammar, type of data used) and repeatable once trained.
4) **Classification**: This is now performed by the loss function; in our case this is the cosine similarity calculation that we optimize for based on the vector representation output from each sub-network. This creates a score that, normalized across all pairs, can also produce a 0-1 similarity score.
5) **Matches / Non-matches / Potential Matches**: Based on the cosine similarity calculation (or its normalized version) we can now determine if input pairs follow into any of these three classifications and label them.
6) **Evaluation**: This step does not change at all: based on the newly labeled pairs, we compare them to the labelled inputs and calculate the performance of our network.
7) **Clerical review**: This step is also required, and it provides a control step for pairs classified as Potential Matches. These can be reviewed, manually labelled accordingly and fed to the network as part a new training set during periodic maintenance re-training.

The deep learning approach removes the complexity of tuning multiple parameters, and heavy customization required with the traditional Data Matching / Linkage approach (i.e.: an n-gram score matching). It also provides a generic engine that can be used across any type of record input, as long as it is fed as a concatenated string representation of its characters. This also alleviates tasks like reference table based name synonym substitution (i.e. “Bob” = “Robert”), address disambiguation and cleanup (i.e. “Rd” = “Road”), as all these features are learned by the network based on the training data set.

After training networks with different values for the parameters, we found these to be the best set:

| Parameter | Value |
|---|---|
| **Dropout keep probability** | 0.25 |
| **L2 Regularization lambda** | 0.0 |
| **Hidden Units** | 50 |
| **Batch Size** | 121 |
| **Number of Epochs** | 300 |
| **Embedding Dimension Size** | 103 |

*Table 5 - Optimal set of network parameters*

The first conclusion based on the results from our analysis is that we will have to invert the results from the network in order to achieve the highest accuracy: by turning true matches into false and vice versa,

we can invert validation accuracy from 0.0405405 to 0.9594595 (1 minus 0.0405405) and test accuracy from 0.0260116 to 0.9793884 (1 minus 0.0260116). It is not exactly the result we were anticipating with our model, but it achieves the result of having a high accuracy for matches.

From all these iterations, we can conclude that the main factor that has an impact network accuracy is the **number of epochs**. **Hidden units** improve accuracy only marginally and they have a very high influence on the training time. Alas, it is not productive to increase them beyond the reasonable limit permitted by the computing power available. As the maximum number of characters from the training/validation/test dataset determine the **embedding dimension size**, this is pre-fixed. The dataset also determines the best possible cases for **batch size** and after some experimentation we found the optimal value to be at 121 rows. **Dropout keep probability** and **L2 lambda for regularization** have a very marginal impact on the network performance, so they should only be used to fine tune results where a more specific data domain is known: as our case is very generic (by comparing proper names, last names and addresses) is reasonable to expect this behaviour.

## Future Work

The first series of improvements that can be done on this work are related to the network layout. By adding more BLSTM layers or using alternative activation functions.

A second series of improvements are related to the optimization function. In our case we calculated loss based on the cosine distance of the two input records representations. An alternative would be to use other type of distance calculations like Manhattan or Minkowski.

A third series of improvements can be more related to the actual technical implementation. By using Tensorflow, the codebase tends to be very verbose and more complicated and cumbersome than desired. Utilizing a simpler framework like Keras, can be an alternative to facilitate code maintainability and transparency.

# References


Benedikt Forchhammer, T. P. (2011). Duplicate Detection on GPUs.

Christen, P. (2012). *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution and Duplicate Detection.* Springer.

Christopher, P. S. (2018). Toward building end-to-end entity matching solutions.

Demetrio Gomes Mestre, C. E.-V. (2017). An efficient spark-based adaptive windowing for entity matching.

Felix A. Gers, J. S. (2000). Learning to Forget: Continual Prediction with LSTM.

Haq, I. U. (2017). *Performance Comparison of Apache Spark and Tez for Entity Resolution.*

Lars Kolb, E. R. (2012). Parallel Entity Resolution with Dedoop.

Larsen, A. A. (2013). Record Linkage.

Matthew Michelson, C. A. (2006). Learning Blocking Schemes for Record Linkage.

Paul Neculoiu, M. V. (2016). Learning Text Similarity with Siamese Recurrent Networks.

Peter Christen, T. C. (2005). FEBRL - Freely Extensible Biomedical Record Linkage.

Rares Vernica, M. J. (2010). Efficient Parallel Set-Similarity Joins using MapReduce.

Rebecca C. Steorts, R. H. (2015). A Bayesian Approach to Graphical Record Linkage and De-duplication.

Sepp Hochreiter, J. S. (1997). Long Short-Term Memory.

Talburt, J. R. (2011). *Entity Resokution and Information Quality.* Morgan Kaufmann.

Thyagarajan, J. M.–A. (2016). Siamese Recurrent Architectures for Learning Sentence Similarity.

Toralf Kirsten, L. K. (2010). Data Partitioning for Parallel Entity Matching.

# Appendix A – Hardware configuration used

All processing on this project was done in Google Cloud Platform using an n1-highcpu-64 machine (64 core server with 57.6 GB of RAM) running Ubuntu Linux 16.. The cloud environment allowed us to provision more processing capacity when required (during training) and less when not (during data preparation and model evaluation).