DeepRepair: A framework for error detection and correction

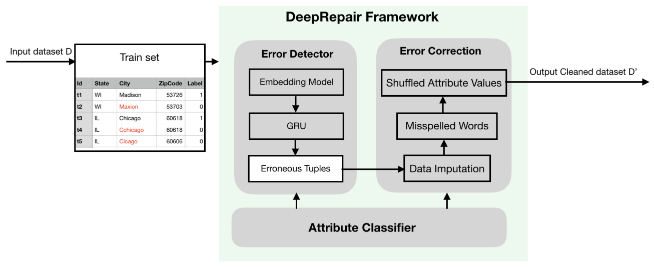
Project Report

Meghana Moorthy Bhat, Yogesh Chockalingam, Manjunath N S
University of Wisconsin Madison
mbhat2,ychockalinga,shettar@wisc.edu

1 PROBLEM AND MOTIVATION

Data analysis using machine learning and other inference methods are common today in industries. The machine learning algorithms are quite sensitive to data errors. The raw data obtained from various external resources are often dealt with inconsistencies, integrity constraints violation, missing values and other uncertainties which make them unsuitable for training without cleaning. Hence, the need for elimination of uncertainty in data has become significantly important. As a result, there have been lots of work/efforts attempts made in industry and academia [5, 8, 10]. The existing methodologies are found to be heavily human-dependent and very few systems provide end-to-end solution of the data cleaning pipeline. Hence, it remains active research area for finding better solutions minimizing human dependency. In the past few years, deep learning (DL) has become a major direction in machine learning [6, 14, 18, 20] and has provided solid results for images, speech and text. By labeling very few samples, DL can automatically construct important features, thereby eliminating the need for manual feature engineering. Recently, DL has also gained the attention of the database research community [4, 16, 20]. Naturally, the question arises if DL can solve the problem of handling inconsistencies and has this approach been tried so far? If yes, what are the solutions proposed and what are the tasks being addressed? How do they perform with respect to existing non-DL solutions? Do they identify uncertainty in dataset and resolve them with minimal human help?

In this project, we answer these questions with the goal of understanding the benefits and limitations of DL for data repairing tasks. The figure 1 shows our end-to-end system incorporating word embeddings in DL models for data repairs. We propose a new framework DeepRepair - a system that automatically detects and corrects errors in a dataset incorporating word embeddings using deep learning models. The data repair is accomplished in two main phases: (i) Error Detection (ii) Error correction. In the error detection phase, we consider the classic setting where DL model is trained on labeled data and the trained model is applied to test data. The erroneous tuples found undergo repair as two different tasks - imputation for missing values, correction for resolving other uncertainties identified from the error detection phase. The framework has a feed-in module called Attribute Classifier. This is a pre-trained DL model from the clean dataset used to classify the attributes.



Id	State	City	ZipCode	Label
t1	WI	Madison	53726	1
t2	WI	Maxion	53703	0
t3	IL	Chicago	60618	1
t4	IL	Cchicago	60618	0
t5	IL	Cicago	60606	0

Figure 1: DeepRepair framework workflow

2 BACKGROUND AND RELATED WORK

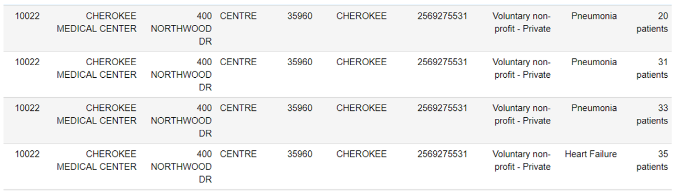
Recently, there has been work on data cleaning using active learning ActiveClean [13], data cleaning using weak supervision HoloClean [17]. ActiveClean requires custom methods to feed-in error indices to the model and requires labeling of sample of correct tuples. HoloClean requires user to provide denial constraints to determine the violations. The existing methods require significant attention of the domain expert but we aim to reduce human effort on data cleaning yet attain good performance results. There has also been some work on embedding structured data using structure2vec [3]. It focuses on embedding latent variable models into feature spaces and learning such feature spaces using discriminate information. While this work adopts the ideas similar to graphical model inference procedures, recent work by fastAI [7] focuses on using deep learning for tabular data, in particular, the creation of embeddings for categorical variables. We focus our efforts on detecting and cleaning the erroneous cells in the structured data through embedding records. To the best of our knowledge, there has been no study made on applications of DL models for error detection and correction.

3 APPROACH AND DESIGN

The setup of our system takes datasets consisting of injected synthetic data errors(0.5% of null values, 0.15% of misspelled words). 0.05% tuples from the dataset are labeled as error '0' or correct '1' (except for hospital). The key approach can be summarized in following steps: (i) represent the text in the form of numerical vector representation (using pre-trained vectors or training from scratch) (ii) summarize the input using a neural network model (iii) classification of the input sequences (correct and wrong labels for error detection) (iv) fill in the missing value using semantic information (v) rectify the misplaced values in cells using word embeddings.

3.1 Embedding Choices

We have set templates in our framework to choose between pre-trained or local trained embeddings. The choice of word embeddings, such as word or attribute-level embeddings (e.g., word2vec[15]) or character-level embeddings (e.g., fastText[1]) can be selected by the user. The pre-trained embeddings offer two distinctive advantages: (1) they lead to smaller training times, and (2) they have



10022	CHEROKEE MEDICAL CENTER	400 NORTHWOOD DR	CENTRE	35960	CHEROKEE	2569275531	Voluntary non-profit - Private	Pneumonia	20 patients
10022	CHEROKEE MEDICAL CENTER	400 NORTHWOOD DR	CENTRE	35960	CHEROKEE	2569275531	Voluntary non-profit - Private	Pneumonia	31 patients
10022	CHEROKEE MEDICAL CENTER	400 NORTHWOOD DR	CENTRE	35960	CHEROKEE	2569275531	Voluntary non-profit - Private	Pneumonia	33 patients
10022	CHEROKEE MEDICAL CENTER	400 NORTHWOOD DR	CENTRE	35960	CHEROKEE	2569275531	Voluntary non-profit - Private	Heart Failure	35 patients

Figure 2: Few sample rows from the dataset

been trained over large corpora, such as Wikipedia or GoogleNews. On the other hand, local trained embeddings have the advantage of faster execution of test tokens due to small search space and consumes significantly lesser memory. We train the vectors from scratch to make the embedding space domain specific. Every tuple is converted into a sentence separated by commas and map each attribute value as "word" in vector space. A brief description of embedding methods are provided below:
word2vec: It is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. We use the skip-gram architecture where the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words. The clean dataset obtained after error detection is used to build the word2vec model. The dataset is modelled like a sentence, which each attribute in a tuple constituting a word. This allows the model to learn words which occur in close proximity to one another and thus word2vec captures the information across the row. The figure below shows a few sample rows from the dataset and the results obtained after training the model.

```
        >>> closest(CHEROKEE, topN = 5)
         ('2569275531', 0.762696385383606),
         ('35960', 0.757010817527771),
         ('CENTRE', 0.7564683556556702),
         ('400 NORTHWOOD DR', 0.7488652467727661),
        ('CHEROKEE MEDICAL CENTER', 0.6274447441101074)
```

As seen above, the top predictions are the closest to the "CHEROKEE" cell.
FastText: It is an open-source library maintained by Facebook, which lends itself to representing sentences with bag of words and bag of n-grams, as well as using sub-word information, and sharing information across classes through a hidden representation. It employs a hierarchical softmax that takes advantage of the unbalanced distribution of the classes to speed up computation. FastText treats every word as composed of character ngrams and generates better word embeddings for rare words as a word with few occurrences will have only few neighbors in word2vec whereas in FastText (due to character ngrams), more neighbors will be present in its vicinity. For example, *Apple* and *Apples* will be 2 different word representations in word2vec closely spaced in the embedding space whereas in FastText, *App, Appl, Apple, ppl, pple, ple, Apples, ples* are the ngram representations which are projected in the embedding space.

The clean dataset obtained after error detection is used to build the fastText model. The dataset is modelled like a sentence, which each attribute in a tuple constituting a word. This allows the model to learn words which are syntactically similar and thus fastText captures words which are similar to the current word. Shown below are the results obtained after training the model.

```
    >>> closest(BUNTERVILLE, topN = 5)
        ('GUNTERSVILLE', 0.9831987619400024),
         ('THOMASVILLE', 0.7798212766647339),
         ('HUNTSVILLE', 0.7334578037261963),
         ('RUSSELLVILLE', 0.7136012315750122),
         ('PRATTVILLE', 0.7130258083343506)
```

Since FastText uses character ngrams, it performs better for capturing the syntactical context while word2vec perform better for semantic analysis.

3.2 Error Detection

This is the first step of the framework DeepRepair. The labeled training instances are fed to a 3-layered neural network. An embedding layer serves as the first layer and is used to embed each input string in the embedding space. These vectors are then passed to a bidirectional Gated Recurrent Unit[2]. The bidirectional units essentially connect two hidden layers of opposite directions to the same output. A batch normalization layer[9] is added to normalize the gradients to make the network converge faster. The final layer produces the output prediction. The network is trained using binary cross-entropy loss, and uses the RMSProp[19] optimizer. The trained model is run on the entire dataset to obtain the clean and dirty datasets. Figure 3 provides the architecture of error detection model.

3.3 Error Correction

The error correction phase in DeepRepair would perform repairs on candidates/cells if they are: null (missing values), misspelled or cell values shuffled with other values of the same column.
Data Imputation: Imputation is the process of replacing missing data with substituted values. The detected errors are filtered to obtain the tuples with one or more null valued cells using *isnull()* function from Pandas (McKinney et al. [2010]). The non-null cells in that tuple is fed to the word2vec model. The cells closer to the missing cell in the dimensional space R have a higher probability of providing the correct value for the missing cell when compared to cells farther away. The attribute classifier module determines the potential value for the missing cell with highest confidence among the top predictions returned by the word2vec model[15].
Handling misspelled errors: The other errors detected and uncleaned are corrected using fastText[1]. These errors are fed to fastText with n-character grams. The top predictions returned by fastText are filtered by Attribute Classifier to measure the choice

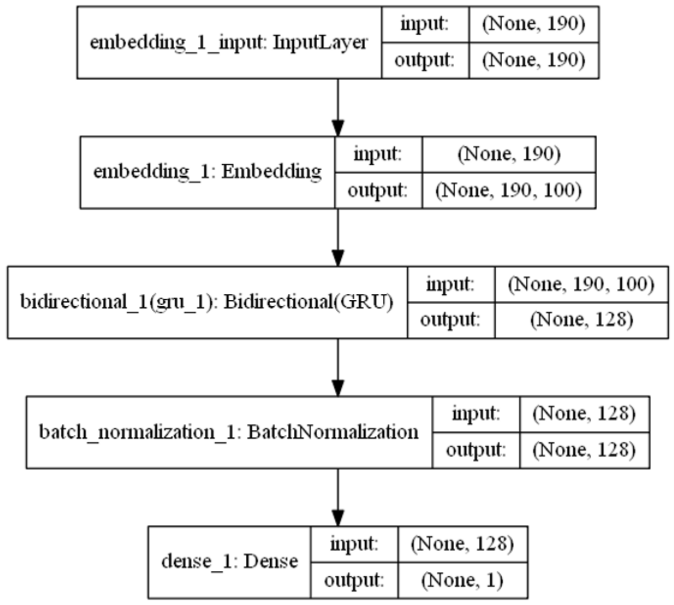


Figure 3: Error Detection

with highest confidence. Following is the algorithm for Error correction:

```
Result: Imputes missing cells
procedure DataImputation($X_i$)
    $X$ : set of erroneous cells
    for each $X_i$ in $X$ do
        if $X_i$ is NULL then
            $relatedCells$ : Non-none cells in the same tuple as $X_i$
            $T$ : Attribute type of $X_i$
            $confidence$ : 0
            for each cell in relatedCells do
                predictions = word2VecModel.mostSimilar($cell$, $topN$=10)
                for each p in predictions do
                    $C$ : Confidence of prediction $p$
                    if AttributeClassifier(p) == $T$ then
                        if $C > confidence$ then
                            $confidence$ = max($confidence$, $C$)
                            $X_i = p$
                        end
                    end
                end
            end
        end
    end
end procedure
```

Algorithm 1: Data Imputation

```
Result: Misspelled cells from potential error candidates
$X$ : set of erroneous cells
$errorSet \leftarrow \emptyset$
for each $X_i$ in $X$ do
    $T$ : Attribute type of $X_i$
    $Y = \{y_i \mid y_i$ set of unique tokens in column T$\}$
    if $x_i$ not in $Y$ then
        $errorSet$ += $x_i$
    else
        continue
    end
end
```

Algorithm 2: Find misspelled cells

3.4 Attribute Classifier

As seen by the predictions returned by the word2vec and the fastText models in the earlier section,the set of top predictions contain cells across all the attributes. When we perform imputation or spell correction, we need to fill or correct the cell with a value that is of the same attribute as the cell. The attribute classifier serves to filter the list of predictions and identify the right attribute valued cell forerror correction.The attribute classifier consists of a deep neural network and is depicted in figure 4, on the left. We obtain the training data for each column by considering all the cell values in a column and labeling them with the column name. This process is repeated for all columns in the dataset. This is an auxiliary module aiding error correction phase by providing categorical/columnar information. It is a neural network consisting of 2 hidden layers and a softmax layer to predict the attribute a cell value belongs to. It uses a categorical cross-entropy loss and the Adam optimizer[11] to train the network.

4 RESULTS AND CONTRIBUTIONS

In Table 1, we show the datasets being used in our experiments. The datasets are picked from Kaggle Machine Learning repository[12]. The charts in Figure 3 show the performance of our models with respect to detection and correction. Table 2 tabulates the runtime performance of DeepRepair. This table does not record the time for training embeddings of DeepRepair. It takes around 2mins(Hospital)-4hrs (Census).

Table 1: Datasets used in our experiments.

Dataset	Size	Attributes	Labeled Tuples	Errors (# of cells)
Hospital	1,000	17	700	301
Adult	32537	12	1626	10172
Census	93590	30	4679	50444

4.1 Analysis

We analyzed the causes behind the performance by plotting a histogram of the attributes the model got wrong most often for Hospital dataset. As seen in figure 6 for data imputation, three attributes,

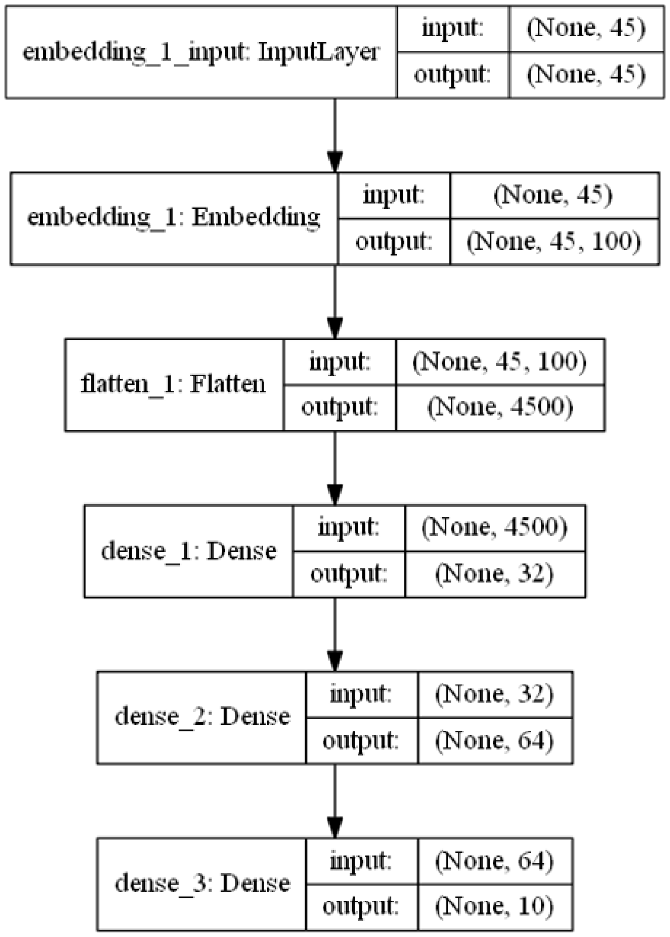


Figure 4: Attribute Classifier

Table 2: DeepRepair Overall performance

Dataset	Precision	Recall	F1
Hospital	0.87	0.68	0.76
Adult	0.63	0.41	0.51
Census	0.51	0.42	0.46

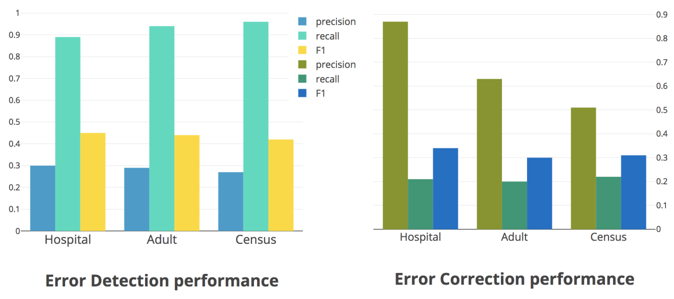


Figure 5: Performance measurements of DeepRepair

Table 3: Runtime performance in seconds

Dataset	DeepRepair
Hospital	18
Adult	782
Census	8996

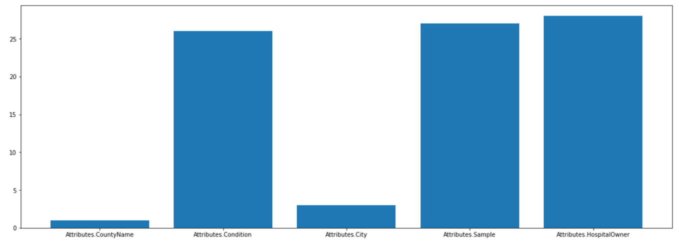


Figure 6: Error Analysis for Data Imputation

namely *Condition, Sample and HospitalOwner* contribute to most of the errors encountered. *Condition and HospitalOwner* are categorical attributes and hence this makes it very difficult for the word2vec model to uniquely associate categorical attributes with a single tuple. *Sample* is an alphanumeric attribute and imputing numerals via word2vec is particularly challenging.

Table 4 represents a few tuples which were not corrected by the model. The main reason behind incorrect predictions were due to ambiguity in attribute classifier's predictions when the cell values were repeated for two or more attributes. For example, we had a misspelling for City named 'FAYETTE'. We also had a County named 'FAYETTE'. The attribute classifier misclassified the prediction as attribute County. We witnessed similar issues with another instances.

Provider Number	HospitalName	City	Zip Code	County Name
10045	FAYETTE MEDICAL CENTER	FAYETTE	35555	FAYETTE
10045	FAYETTE MEDICAL CENTER	FAYETTE	35555	FAYETTE

Table 4: Results: Spelling Correction

Contributions: 1. To the best of our knowledge, we are the first to build end-to-end pipeline for data repairs using DL models. 2. DeepRepair scales and generalizes well.

Technical Challenges: 1. Our model do not fit well on continuous data. 2. The existing implementation do not scale for columns >32 due to the hard limit of numpy against dataframe query.

Future Work: In future, we would like to change our implementation to other engines for column scalability. We aim to improve the performance of data cleaning by handling errors with respect to continuous values as they account to 80% incorrect predictions currently. We would like to explore unsupervised mechanisms to eliminate labeling of data.

ACKNOWLEDGMENTS

The authors would like to thank Professor Theodoros Rekatsinas for his tremendous patience, experience, guidance and valuable inputs during the project.

REFERENCES

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606* (2016).

[2] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* abs/1412.3555 (2014). arXiv:1412.3555 http://arxiv.org/abs/1412.3555

[3] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 2702–2711. http://dl.acm.org/citation.cfm?id=3045390.3045675

[4] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2017. DeepER - Deep Entity Resolution. *CoRR* abs/1710.00597 (2017). arXiv:1710.00597 http://arxiv.org/abs/1710.00597

[5] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* 33, 2, Article 6 (June 2008), 48 pages. https://doi.org/10.1145/1366102.1366103

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[7] Jeremy Howard et al. 2018. fastai. https://github.com/fastai/fastai.

[8] Ihab F. Ilyas and Xu Chu. 2015. Trends in Cleaning Relational Data: Consistency and Deduplication. *Found. Trends databases* 5, 4 (Oct. 2015), 281–393. https://doi.org/10.1561/1900000045

[9] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 448–456. http://proceedings.mlr.press/v37/ioffe15.html

[10] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 3363–3372. https://doi.org/10.1145/1978942.1979444

[11] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14

[12] Ron Kohavi. 1996. Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*.

[13] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Jiannan Wang, and Eugene Wu. 2016. ActiveClean: An Interactive Data Cleaning Framework For Modern Machine Learning. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 2117–2120. https://doi.org/10.1145/2882903.2899409

[14] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (27 5 2015), 436–444. https://doi.org/10.1038/nature14539

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13)*. Curran Associates Inc., USA, 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959

[16] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 19–34. https://doi.org/10.1145/3183713.3196926

[17] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[18] Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks. *Neural Netw.* 61, C (Jan. 2015), 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958. http://dl.acm.org/citation.cfm?id=2627435.2670313

[20] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Rec.* 45, 2 (Sept. 2016), 17–22. https://doi.org/10.1145/3003665.3003669