Learned Cardinalities — MSCN Revisited

Asoke Datta
adatta2@ucmerced.edu
University of California Merced

May 2021

Abstract

MSCN [6] is a multi-set convolutional network, tailored to representing relational query plans, that employs set semantics to capture query features and true cardinalities. Though MSCN model was builds on sampling-based estimation , addressing its weaknesses when no sampled tuples qualify a predicate, and in capturing join-crossing correlations, in this report, we reduce the problem to single table and evaluate MSCN using four real world dataset for single table cardinality estimation problem.

1 Model Introduction

Multi-set convolutional network (MSCN) [6] model is inspired by recent work on Deep Sets [13], a neural network module for operating on sets. To use Machine Learning model in database first challenge was to learn a way to represent data and query(Can contain multiple predicates/conditions), and pass that representation to a ML model which learn co-relation between data and query, finally will return a scaler/output. As our query can contain multiple predicates and sample size can be large, that makes using the set representation an ideal candidate.

Model takes one hot encoded set representation of sample and predicates columns, and numerical value in a normalized form $\in$ [0,1] as input 1.4. Lets assume we have 5 samples, 4 predicates, and 3 three hidden layers as Figure 1. At L0(input layer) model takes 2 different set as input using two MLP's. Samples and predicates for respective query , model applies linear activation function followed by ReLU(L1) $\rightarrow$ Linear(L2) $\rightarrow$ (L3)ReLU. Then next layer(L4) takes output of Samples and Predicates, concatenate them and applies linear activation function followed by ReLU(L5) $\rightarrow$ Linear(L6) $\rightarrow$ Sigmoid(L7).

At L7 (sigmoid) layer, model predicts a value. Then model's optimizer calulate the loss function by comparing the predicted value with true value and backpropagate the loss information to the model.

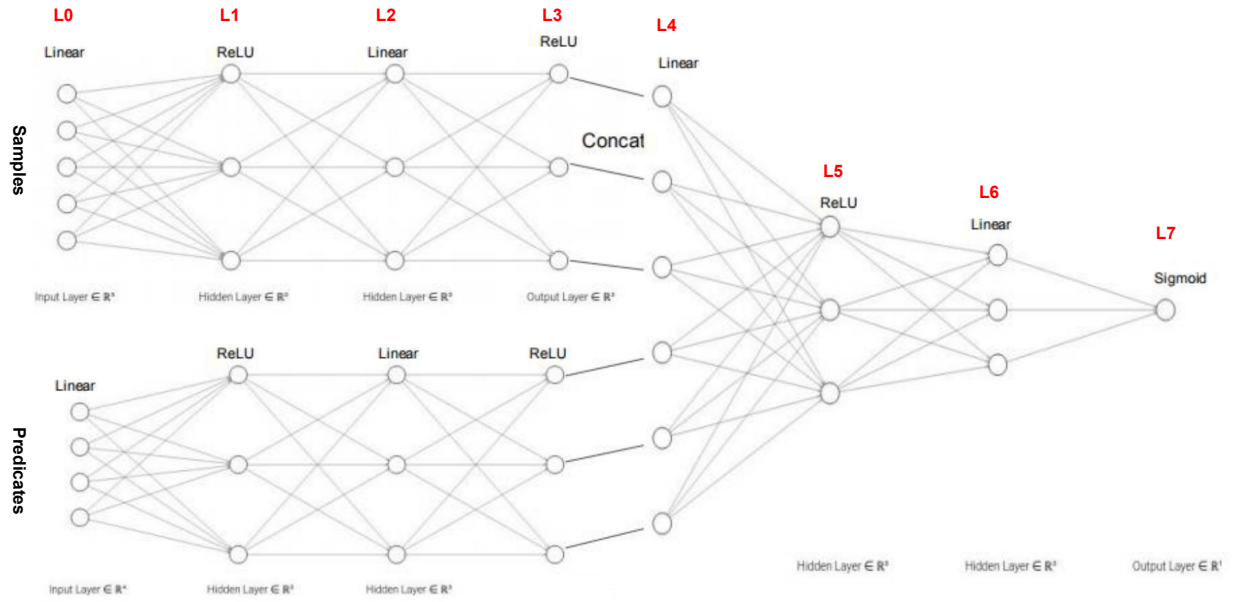


Figure 1: Model Representation for 5 sample input features and 4 Predicate features

1.1 Activation Layers

In simple words, an activation function is a function that is added into an artificial neural network in order to help the network learn complex patterns in the data. It takes in the output signal from the previous layer and converts it into some form that can be taken as input to the next layer. This is important because input into the activation function is

$$W * x + b$$

where W is the weights of the cell(hidden unit/ neuron) and the x is the inputs and then there is the bias b added to that. This value is not restricted to certain limit, it can go very high in magnitude especially in case of very deep neural networks that have millions of parameters. This will lead to computational issues. In MSCN, author's use three different activation functions across the model. Now I will discuss what each of them do.[11]

- **Linear**: It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input.

  $$f(x) = cx$$

  Where $f(x)$ Linear function, x - input, c - some constant.
  After plotting values for x, the output linear function f(x) will look like Figure 2

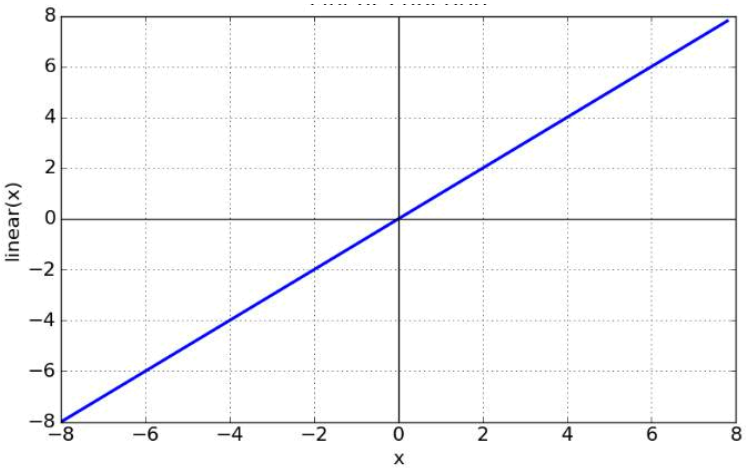


Figure 2: Linear Function

- **ReLU**: The rectified linear activation function or ReLU is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. ReLU (Rectified Linear Unit) is defined as

$$f(x) = max(0, x)$$

Where $f(x)$ ReLU function, x - input, **max** is a function which determine maximum between two items.
After plotting values for x, the ReLU function f(x) will look like Figure 3

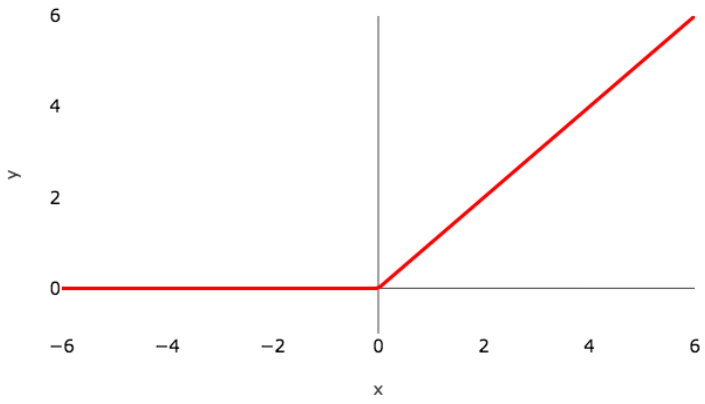


Figure 3: ReLU function

- **Sigmoid**: A sigmoid function is a type of activation function, and more specifically defined as a squashing function, which limits the output to a range between 0 and 1.

$$S(x) = \frac{1}{1 + e^{-x}}$$

Where S(x) - Sigmoid function, e - Euler's number.
After plotting values for x, the sigmoid function S(x) will look like Figure 5

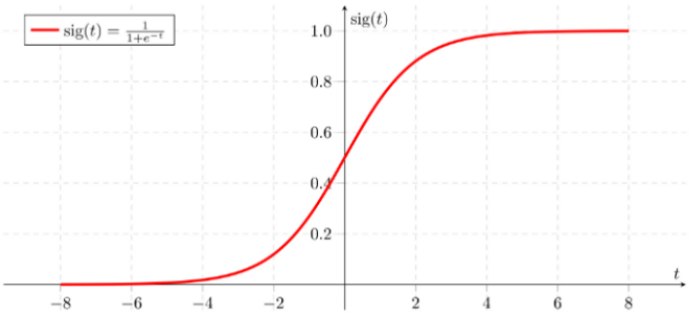


Figure 4: Sigmoid function

1.2 Loss Functions

Neural networks are trained using stochastic gradient descent and require to choose a loss function when designing and configuring your model. Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply "loss." [4] . In calculating the error of the model during the optimization process, a loss function must be chosen. In MSCN, author's considered three different loss functions.

- **Mean q-error:** q-error is the factor between an estimate and the true cardinality (or vice versa). It minimize the mean q-error while $(q \geq 1)$.
- **MSE (Mean Squared Error):** Optimize the squared differences between the predicted and true result.
- **Geometric mean q-error:** It optimizes the geometric mean of the q-error.

Goal of the MSCN model is to minimize relative difference between predicted and true cardinality. Whereas MSE minimizes the squared difference which doesn't align with the MSCN objective and Geometric mean q-error put less emphasis on heavy outliers. Which makes Mean q-error best choice for loss function as this optimizes factor difference between predicted and true cardinality.

1.3 Optimizer : Adam

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. Optimization algorithms are responsible for reducing the losses and to provide the most accurate results possible. [2]. Various optimizers i.e.

Gradient Descent,Stochastic Gradient Descent (SGD),Adaptive Gradient (AdaGrad), RMSprop, Adam etc. are researched within the last few couples of years each having its advantages and disadvantages. In MSCN, author's used **Adam** (Adaptive Moment Estimation) optimizer. Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum.

- It is fast compare with other optimizers and converges rapidly.
- Rectifies vanishing learning rate, high variance.
- Provide an optimization algorithm that can handle sparse gradients on noisy problems.
- Adam is relatively easy to configure where the default configuration parameters do well on most problems.
- Based deep learning community, In practice Adam is currently recommended as the default algorithm to use.

1.4 Set-Based Query Representation

To use Machine Learning for cardinality estimation, the most important question is how to represent queries. In the remainder of this sub-section, we address these questions how to featurize query information and represent materialized samples.

Single table queries can be represented as set of predicates(P). Each predicate(p) can be represented by a unique one-hot vector(a binary vector of length $|P|$ with a single non-zero entry, uniquely identifying a specific predicate) and optionally the number of quali- fying base table samples or a bitmap indicating their positions. DMV dataset [9] has 11 columns. One hot representation will look like,

Columns	One Hot Representation
Body_Type:	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Color:	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
County:	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Fuel_Type:	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
Record_Type:	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
Reg_Valid_Date:	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
Registration_Class:	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
Revocation_Indicator:	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
Scofflaw_Indicator:	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
State:	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
Suspension_Indicator:	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

Each of these above predicates are represented uniquely using a binary vector with single non-zero entry. Operators ($\leq$,$\geq$,=) can also be represented in a similar fashion.

```
<=:     [1, 0, 0]
=:      [0, 1, 0]
>=:     [0, 0, 1]
```

The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. For

example dmv dataset contain 11 columns. Each of the column has different domain of values. The great difference in the scale of the values could cause problems when attempt to combine the values as features during modeling. Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model [7]. As an example, let's consider below query,

```
SELECT COUNT(*) FROM dmv11
WHERE Color = 100
```

Domain of color column in dmv dataset is 1 to 225. In MSCN, author's normalize to interval [0,1] using Min and Max value(logarithmized) obtained from dataset. To normalize 100 to [0,1] interval we use below formula,

$$\frac{value-min}{max-min} = \frac{log(100)-log(1)}{log(225)-log(1)} = \frac{2-0}{2.35-0} = 0.85$$

1.5 Sampling

Sample is the subset of the dataset. The process of selecting a sample is known as sampling.And, Number of elements in the sample is as known as sample size. A key idea of MSCN [6] approach is to enrich the training data with information about materialized base table samples. For every query, author's evaluate the corresponding predicates on a materialized sample and annotate the query with the number of qualifying samples s ($0 \leq s \leq 1000$ for 1000 materialized samples) for this table. MSCN takes this idea one step further and annotate each table in a query with the positions of the qualifying samples represented as bitmaps. An example from dmv is depicted below about bitmap representation. Let's consider after obtaining sample size of 10 from dmv data set we get below rows.

```
0.VEH | ORG | NY | SUFFOLK    | SUBN | GAS | 20181204 | WH | N | N | N
1.VEH | PAS | NY | NIAGARA    | SUBN | GAS | 20180710 | GY | N | N | N
2.BOAT| BOT | NY | DUTCHESS   | BOAT | ELEC| 20180318 | N  | N | N | N
3.VEH | PAS | NY | SUFFOLK    | SUBN | GAS | 20170224 | WH | N | N | N
4.VEH | MOT | NY | CHAUTAUQUA | MCY  | GAS | 20180404 | GR | N | N | N
5.VEH | PAS | NY | ALBANY     | 4DSD | GAS | 20190103 | BL | N | N | N
6.VEH | PAS | NY | BRONX      | SUBN | GAS | 20170616 | BL | N | N | N
7.VEH | PAS | NY | RICHMOND   | 4DSD | GAS | 20170313 | WH | N | N | N
8.BOAT| BOT | NY | LEWIS      | BOAT | GAS | 20180726 |    | N | N | N
9.VEH | COM | NY | NIAGARA    | PICK | GAS | 20171211 | WH | N | N | N
```

Now, we run below query in above sample.

```
SELECT COUNT(*) FROM dmv11
WHERE Record_Type = BOAT;
```

Only, 2nd and 8th row satisfy this query. So, bitmap representation will be

```
[0 0 1 0 0 0 0 0 1 0]
```

1.6 Masking and Padding

Masking is a way to tell sequence-processing layers that certain timesteps in an input are missing. And, Padding is a special form of masking where the masked steps are at the start or the end of a sequence [10]. Number of set elements in each data sample in a mini-batch can vary. Author's pad all samples with zero-valued feature vectors that act as dummy set elements so that all samples within a mini-batch have the same number of set elements. And, mask out dummy set elements in the averaging operation, so that only the original set elements contribute to the average.

2 MSCN to single-table cardinality estimation Problem

Single table cardinality estimation, a fundamental and long standing problem in query optimization [3]. It is the task of estimating the number of tuples of a table that satisfy the query predicates. To understand the problem, lets consider an example query from dmv dataset [9].

```
SELECT COUNT(*) FROM dmv
WHERE RecordType = VEH AND
RegistrationClass = PAS AND
RegValidDate >= 20180801;
```

As like above query, we can have equality predicate like *RecordType = VEH*, open range queries like *RegValidDate* $\geq$ *20180801* or a closed range query like a $\leq$ A $\leq$ b. The goal is to estimate the cardinality of this query. An equivalent problem is called selectivity estimation, which computes the percentage of tuples that satisfy the query predicates.

In "Are We Ready For Learned Cardinality Estimation?" paper [12] author's considered three type of queries for model training and test. Which are as follows:

- **Point queries** A point query is a query with a single equality in the condition. An example query from dmv and set representation 1.4 as follows.

```
SELECT COUNT(*) FROM dmv11
WHERE Color = GY;

{[0 1 0 0 0 0 0 0 0 0 0   0 1 0   0.85]}
column                    op      val
```

- **Range queries** A range query is a common database operation that retrieves all records where some value is between an upper and lower boundary. An example query from dmv and set representation 1.4 as follows.

```
SELECT COUNT(*) FROM dmv11
WHERE Reg_Valid_Date >= 20180801;
```

```
{[0 0 0 0 0 1 0 0 0 0 0   0 0 1  0.73]}
column                    op      val
```

- **Conjunctions** It's contains multiple predicates which can be point predicates or range predicates or both type of predicates. An example query from dmv and set representation 1.4 as follows.

```
SELECT COUNT(*) FROM dmv11
WHERE Record_Type = VEH AND
Registration_Class = PAS AND Color = GY;

{[0 1 0 0 0 0 0 0 0 0 0 0 1 0   0.25],
[0 0 0 0 1 0 0 0 0 0 0  0 1 0   0.31],
[0 0 0 0 0 0 1 0 0 0 0  0 1 0   0.85]}
column                    op      val
```

3 Implementation Details

From high level,we can depict MSCN complete workflow as follows -

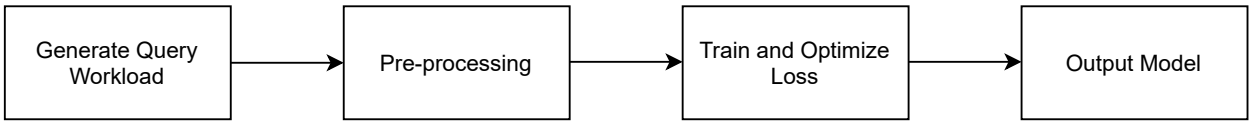


Figure 5: MSCN Workflow

3.1 Generate Query Workload

One key challenge of all learning-based algorithms is the cold start problem, i.e., how to train the model before having concrete information about the query workload [6]. MSCN obtain an initial training workload by generating random queries based on schema information which happens in three steps.

- Uniformly draws number of predicate ($0 \leq$ number of predicate $\leq$ number of columns).
- Uniformly draw a operator ($\leq, \geq, =$).
- Draw a value from the corresponding column for respective predicate (actual data).

After generating query, then execute these queries to obtain their true result cardinalities, while skipping queries with empty results. Using this process, author's obtain the initial training set for our model[6].

3.2 Pre-precessing

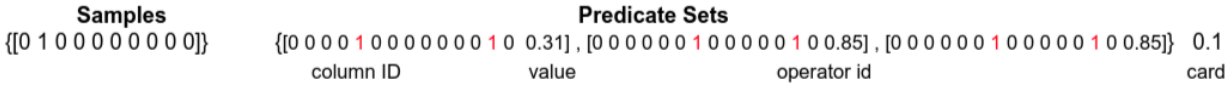
A training sample consists of table predicates, annotated bit vector of sample and true cardinality of the query. Processing of generated training data includes below steps -

- Transform the predicates into set representation 1.4.
- Run query on sample data and annotate bit vector 1.5.
- Normalize literals using domain minimum and maximum values 1.4.

Final representation of the query

```
SELECT COUNT(*) FROM dmv11
WHERE Record_Type = VEH AND
Registration_Class = PAS AND Color = GY;
```

Will be



And, we repeat this process for all the training queries. Python open source library torch,pandas,numpy are used to prepare and process the dataset.

3.3 Model Training

Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization [5]. In MSCN, model learns from training queries by iterating over them multiple times. Those iteration called **epochs**, which means simply one iteration over complete dataset.Default number of epoch is 200. Dataset's are normally divided in batches often known as **batch size**. Based on heuristic and trial-error one can determine what is a suitable batch size for a model. For MSCN, author's use 1024 as default batch size. In each epoch, optimizer (default optimizer is Adam 1.3) learns in a fixed rate which also known as a **learning rate**(sometimes called step size). It controls how big the jumps your model makes, and from there, how quickly it learns. If we choose a learning rate that is too small, neural network will take a long time to converge, on the other hand if learning rate that is too big, model will not learn properly [1]. So, its one of the most important parameter for the model. Author's found at rate 0.001 model converges to minimum with most accuracy and reasonable time. After learning one batch, model calculate the error through loss function1.2 and back propagates information about the error, in reverse through the network, so that it can fix the error by altering the parameters [8]. So, minimizing the loss function when improving the accuracy is the goal of training in learning algorithms.

Once training process is finished, model is prepared to predict for similar kind of data, for our case queries. Often we can call this final model as output model.

4 Experimental reproducibility

Goal of these experiment is to observe, if we can reproduce the reported result in "Are We Ready For Learned Cardinality Estimation?" [12]. Reproducibility is important because it is the only thing that can verify and guarantee about a study. So reproducibility is important not because it ensures that the results are correct, but rather because it ensures transparency and gives us confidence in understanding exactly what was done. In this section, we repeat the experiments with default setting. And, at the end, we do a comparative analysis with reported result in paper.

4.1 Experimental setup

We run the experiments on a Ubuntu 20.04 LTS machine with 12 CPU cores (Intel(R) Core(TM) i7-9850H CPU @ 2.60GHz), 15GB RAM, SSD storage, and an GeForce GTX 1650 Mobile/Max-Q GPU. To train MSCN, we need to set values for certain parameters which are dataset, number of samples, number of hidden units, number of epochs, batch size and number of training examples. Default values for all of these parameters are -

```
dataset = dmv/census/power/forest,
number of samples = 1000,
number of hidden units = 16,
number of epochs = 200,
batch size = 1024,
number of training examples = 100000
```

We used four widely used dataset because of their deference is sizes and the ratio between categorical and numerical columns varies. More details about dataset can be found here [12]. Rest of the parameters are self explanatory.

4.2 Result

DMV : DMV contains 11.6M tuples which is largest across all dataset. Out of its 11 attributes 10 is categorical and 1 numeric. Total time for training takes 55.64 minutes.

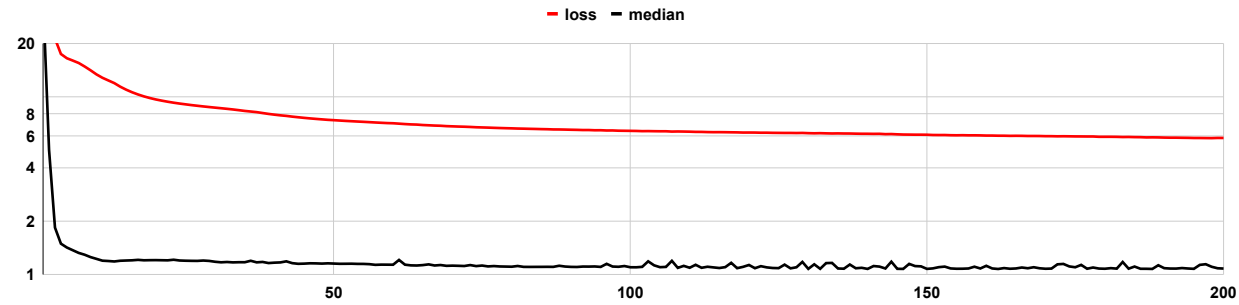


Figure 6: DMV : loss and median

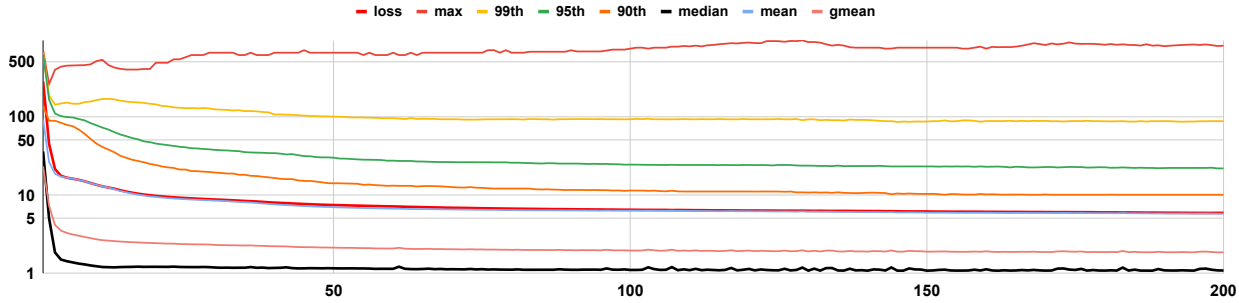


Figure 7: DMV : All stat

In Figure 6 and 7, we depict how loss,median and other evaluation metric are changing as number of epochs change. Here, X axis refers to number of epochs.

Census : Census contains 49K tuples and 13 attributes, out of which 8 is categorical and rest of them are numerical. Total time for training takes 56.83 minutes.

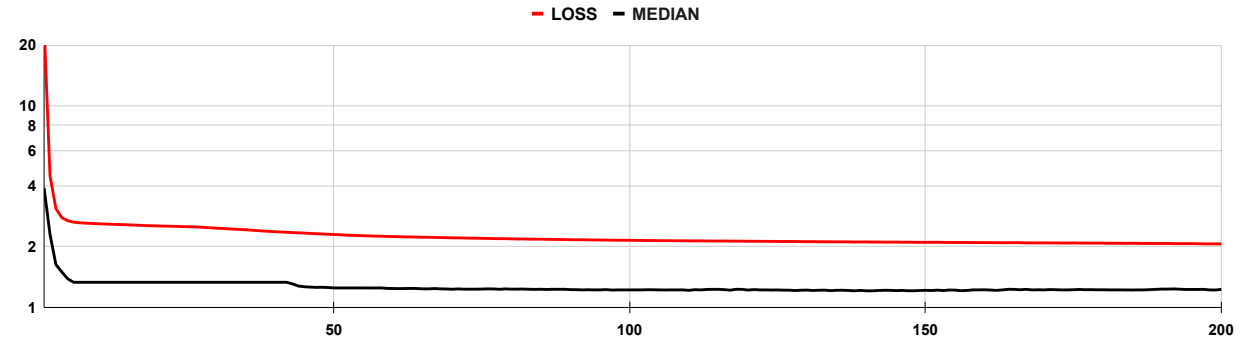


Figure 8: Census : loss and median

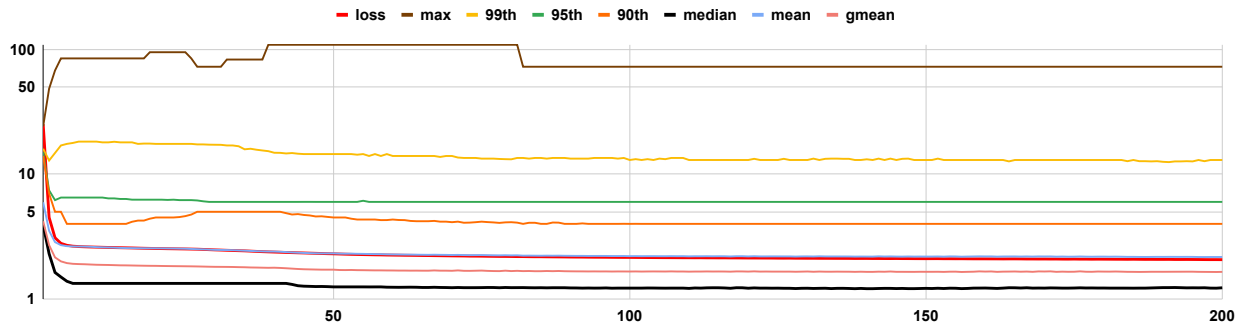


Figure 9: Census : All stat

In Figure 8 and 9, we depict how loss,median and other evaluation metric are changing over different epochs. Here, X axis refers to number of epochs.

Forest : Forest dataset contains 581K tuples and 10 attributes, all of them are numerical. Total time for training takes 58.51 minutes.

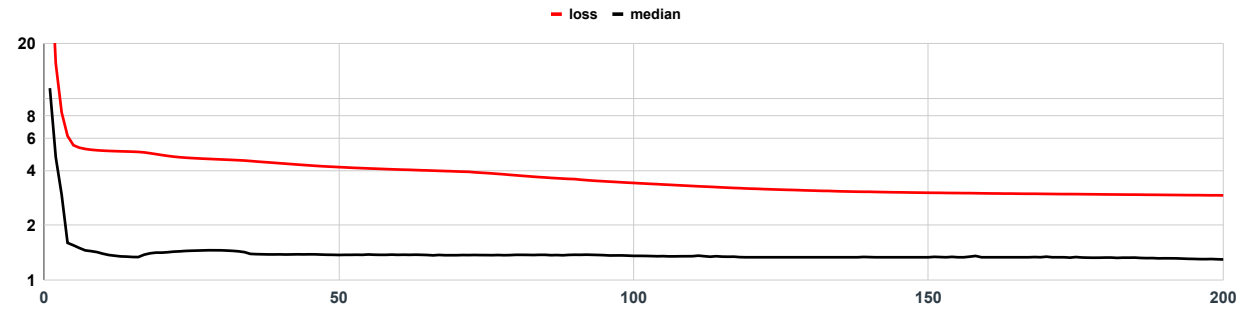


Figure 10: Forest : loss and median

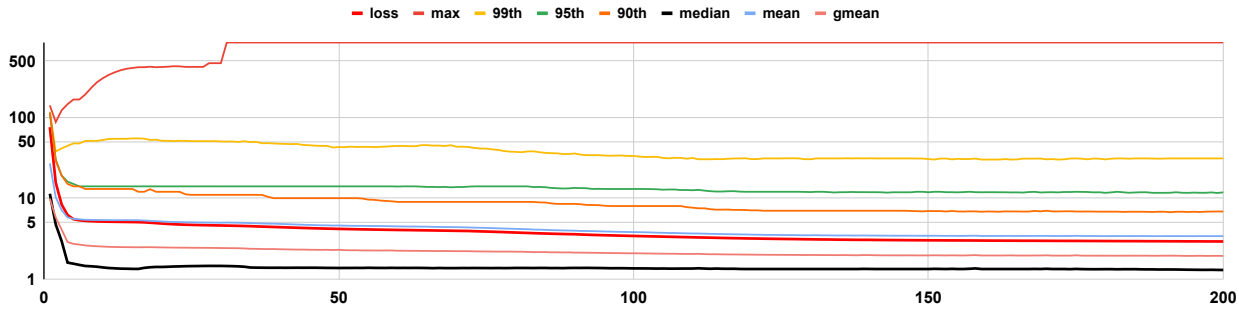


Figure 11: Forest : All stat

In Figure 10 and 11, we depict how loss,median and other evaluation metric are changing over different epochs. Here, X axis refers to number of epochs.

Power : Power dataset contains 2.1M tuples and 7 attributes, all of them are numerical. Total time for training takes 59.87 minutes.

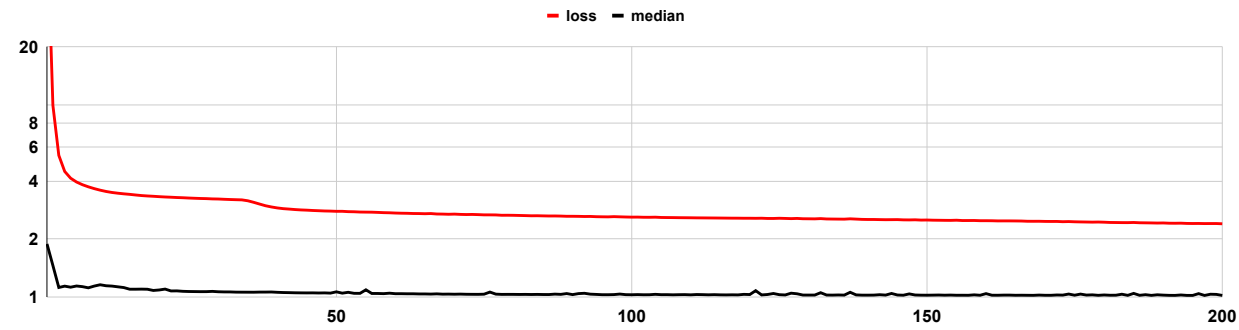


Figure 12: Power : loss and median

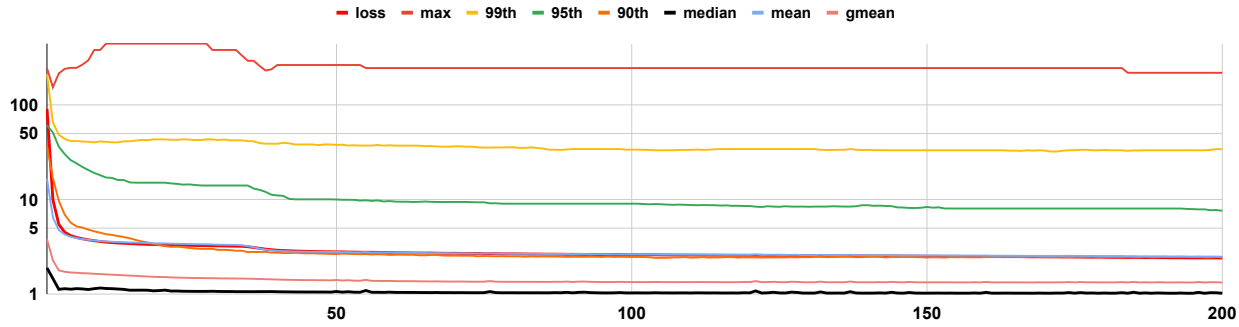


Figure 13: Power : All stat

In Figure 12 and 13, we depict how loss,median and other evaluation metric are changing over different epochs. Here, X axis refers to number of epochs.

4.3 Some observation and Comparison with "Are We Ready for Learned Cardinality Estimation?"

Based on training figures in section 4.2, we can safely say within 50-75 epochs model coverages to minimum and that is true for all the dataset. Based on this observation, we can safely limit number of iteration to 100. Reduce number of epochs will reduce the total training time in half (around 30 minutes) as default setting for number of epochs is 200.

In rerun with default setting, MSCN performs better in **census** and similar in **power**. For **forest** and **DMV** dataset reported result in paper [12] stays better. There are random components involved in training process of MSCN like samples from where model learns about the distribution of data. It is possible that if we draw 1000 sample randomly 2 times, we will get very different set of tuples

which will lead to learn a very different distribution of data. And, MSCN is heavily dependent on sample which might be the cause of observing difference between report result and rerun result.

Dataset	MSCN				MSCN - rerun			
	50th	95th	99th	Max	50th	95th	99th	Max
Census	1.38	7.22	15.5	88	**1.23**	6	12.71	72.66
Forest	**1.14**	7.62	20.6	377	1.3	11.72	31	838
Power	**1.01**	2	9.91	199	**1.01**	7.57	33.83	217.22
DMV	**1.02**	5.3	25	351	1.08	21.68	87	792

Figure 14: Comparison : MSCN rerun default vs Paper

5 Proposed modifications

Based on observation from previous section 4.2, we have seen MSCN model is sensitive to samples. It will be really interesting to see how MSCN perform with varying number of Sample. Based on author's claim in MSCN paper [6] regarding hyperparameter tuning and dataset size to number of training queries ratio, we plan to observe model behavior by designing few experiments on those.

5.1 Idea

We propose three different experiments to observe model behavior. Those are

- **Varying sample sizes :** Run training with sample size of 1,10,100,500. And compare with existing result to see how sensitive is MSCN towards sample.
- **Varying Hidden Layers :** Author's claim in MSCN [6] model performs best with 256 hidden layers whereas reported result in "Are We Ready for Learned Cardinality Estimation?" [12] paper author's used 16 hidden layer as default. We will run the training with 256 hidden layers and compare result with reported result.
- **Dataset size vs Number of training examples :** In section 4 and "Are We Ready for Learned Cardinality Estimation?" paper [12] we have used four different data which varies significantly in sizes(number of rows). But for all of these case we have 100K training example. So, can we achieve similar or better performance using less training example ?

Dataset	Num of rows	Ratio with training example
Census	49000	0.49
Forest	581000	5.81
Power	2100000	21
DMV	11600000	116

Figure 15: Number of rows vs Number of training queries

5.2 Implementation

The experiments we propose in section 5, doesn't require any changes in code provided in . By controlling the parameters of the model we were able to perform our experiments.

5.3 Experiments

Varying sample sizes We perform training for all the dataset using sample size 1,10,100 and 500.

- Sample size 1 - We obtain median census(2.5), dmv(9.09), forest(2.6), power(1.81). For, census,forest and power is within two order of magnitude compare with reported result [12]. But, dmv is way out range from reported result in [12].

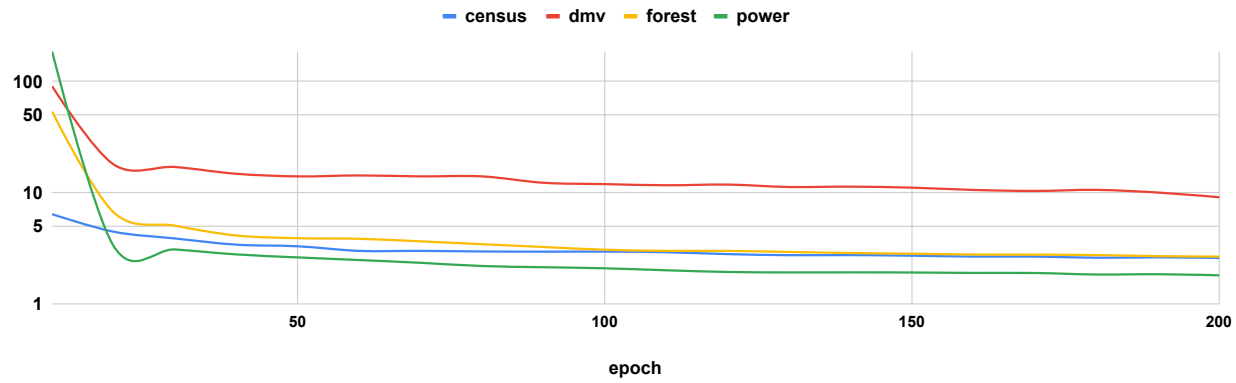


Figure 16: Sample Size 1

- Sample size 10 - We obtain median census(2), dmv(3.74), forest(2.87), power(1.2). For dataset census, power and dmv improves compare with sample size 1. Forest doesn't improve.

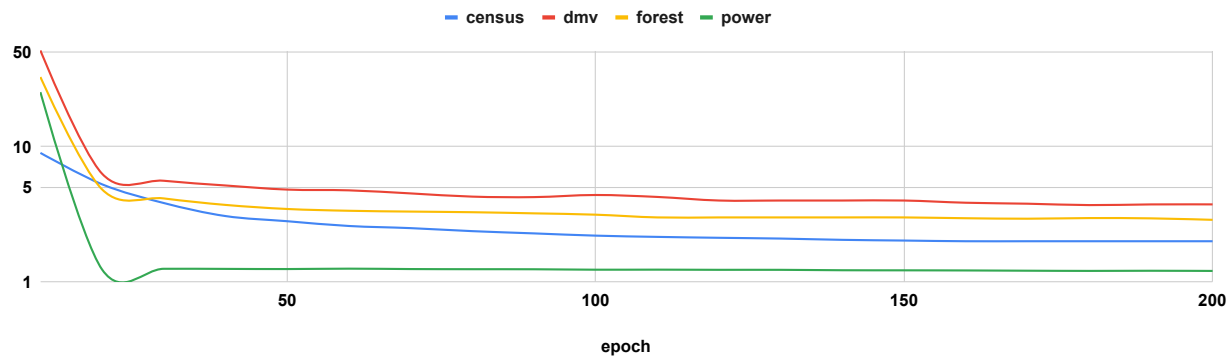


Figure 17: Sample Size 10

- Sample size 100 - We obtain median census(1.58), dmv(1.38), forest(1.78), power(1.06). For all dataset, median improves.

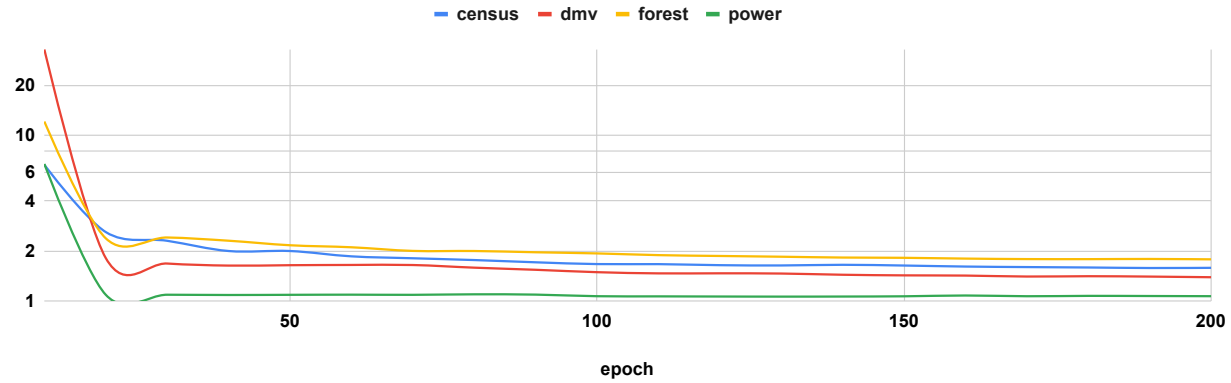


Figure 18: Sample Size 100

- Sample size 500 - We obtain median census(1.33), dmv(1.15), forest(1.4), power(1.01). For census and power model match the reported result in [12]. DMV and forest improves compare with sample size 100.

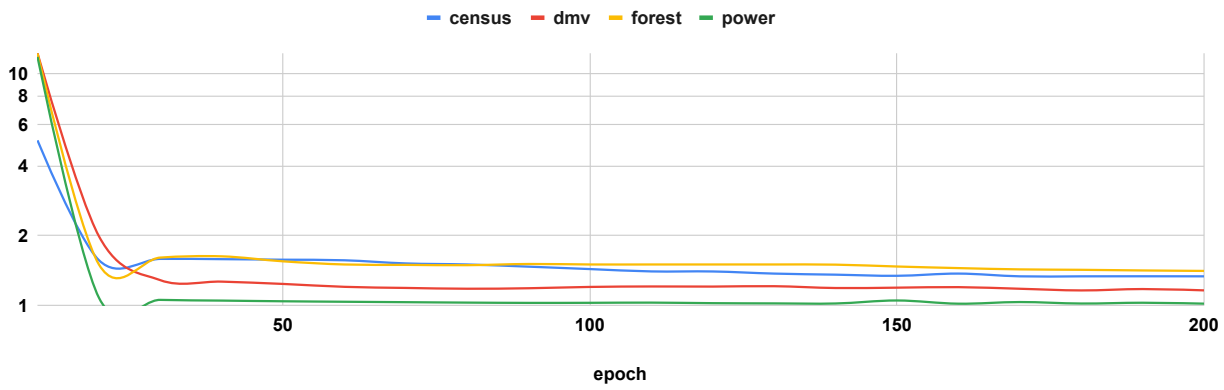


Figure 19: Sample Size 500

Varying Hidden Layers Default setup for MSCN model in [12], contains 16 hidden layer. For this experiment, we set it to 256, and leave other parameters as default.

Dataset	MSCN				MSCN - 256			
	50th	95th	99th	Max	50th	95th	99th	Max
Census	1.38	7.22	15.5	88	**1.09**	4.53	9	84
Forest	**1.14**	7.62	20.6	377	1.22	8.45	23	2730
Power	**1.01**	2	9.91	199	1.04	3.39	17.4	478
DMV	**1.02**	5.3	25	351	1.04	12	53.9	1088

Figure 20: MSCN(16) vs MSCN(256)

Based on the result we obtain, model performs better only for census dataset. And, for other datasets result reported in paper [12] remains similar or better.

Dataset size vs Number of training examples Among four dataset the smallest one is census with 49k rows and largest one is DMV with 11.6 M. Default configuration for number of training example is 100k for both of this dataset. As per figure 15, for DMV there is one training query per 116 rows whereas for census ratio is for every row there are two queries. Based on this observation, seems author's assume there is no corelation between dataset size and number of training example as it was 100k for all of the dataset. So, is it true that there is no co-relation ? We run the experiments with 50k training example and 100 epochs as there is no change in evaluation metric after 100 epochs 4.3.

Dataset	MSCN				MSCN - 50k			
	50th	95th	99th	Max	50th	95th	99th	Max
Census	1.38	7.22	15.5	88	**1.26**	6.8	17.5	109
Forest	**1.14**	7.62	20.6	377	1.39	15	39.5	838
Power	**1.01**	2	9.91	199	1.06	10	38	335.1
DMV	**1.02**	5.3	25	351	1.2	28	97	677

Figure 21: MSCN(100K) vs MSCN(50K)

Result we obtain is decipt in 21. For, census dataset we obtain better result compare with original(MSCN 100K) one. For larger dataset, model perform reasonable compare with the MSCN(100K) model.

5.4 Discussion

Based on observation from experiments in section 5 we have seen parameters are sensitive to dataset sizes, type of data and also distribution of data. In **sample size** experiment we have seen model to achieve reported result in paper [12] for census and power using half of sample size compare with default(1K). As dmv has 11.6M tuple, mostly likely model will require more sample to learn about distribution of data. Among all the dataset, we were not able to reproduce the reported result for forest dataset. We can consider it as a outlier. Regarding hidden layers experiment, MSCN is tailored to capture join co-relation, so the proposed configuration may performs well for multi-table dataset. But for single table cardinality estimation problem, smaller number of hidden layers outperforms best configuration of paper [6]. We can keep in mind, the ratio between dataset size and number of training example during parameter tuning as we have observed using small number of training example for relatively smaller dataset we can achive reported performance [12] which can save time and memory.

6 Summary

Machine learning models are very sensitive to parameter choices. Instead of fixing generalize parameters we can optimize model performance and save time in training and memory by tuning parameters specific to dataset.

References

[1] A. Bilogur. Tuning your learning rate. `https://www.kaggle.com/residentmario/tuning-your-learning-rate`.

[2] S. Doshi. Various Optimization Algorithms For Training Neural Network. `tinyurl.com/wvxbfn`.

[3] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. In *45th International Conference on Very Large Data Bases (VLDB 2019)*, August 2019.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] M. L. Google. Descending into ML: Training and Loss. `https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss`.

[6] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

[7] Microsoft. Normalize Data. `https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data`.

[8] C. Nicholson. A Beginner's Guide to Backpropagation in Neural Networks. `https://wiki.pathmind.com/backpropagation`.

[9] S. of New York. Vehicle, Snowmobile, and Boat Registrations. `https://catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations`.

[10] tensorflow. Masking and padding. `https://www.tensorflow.org/guide/keras/masking_and_padding`.

[11] M. Vandit Jain. Everything you need to know about "Activation Functions" in Deep learning models. `tinyurl.com/wnwmbbd8`.

[12] X. Wang, C. Qu, W. Wu, J. Wang, and Q. Zhou. Are we ready for learned cardinality estimation? *CoRR*, abs/2012.06743, 2020.

[13] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.