Design Space Exploration of Model Serving
Group 3 Project Report for the course of Machine Learning System

Anonymous Authors[1]

Abstract

The goal of this project is to explore the configuration space of model serving to minimize the latency from users' perspective. Experiments are conducted upon three DNN models, GAN, OPENNMT-tf, and ResNet. We use Tensorflow serving to serve each model, and Docker to simulate an experimental setting of multiple servers in a server center. We use Bayesian optimization to search for the optimal configuration and compare with random configurations. Based on our experiments, we find that performance gain of best configuration given by Bayesian optimization over the best random configuration will trend up when the number of concurrent requests increases.

1. Introduction

Machine learning projects take a lot of effort to train and validate. However, once the model is finished how does it go about reaching the end user? Traditionally, a typical framework to accomplish this would be deploying an API using flask. However, this will cause the system to incur technical debt with "Glue Code" (Sculley et al., 2015). With all the excitement surrounding machine learning, it begs the question of how a system can be put into production. Another option that will reduce the technical debt is model serving. Model serving is a better alternative because it will handle performance, scaling, and version.

Model severing, in addition to the models themselves, is a highly configurable system. Our project seeks to optimize a machine learning system and model serving configuration in order to minimize latency. We will compare two methods of sampling configurations, random selection and Bayesian optimization applied to different models with different DNN architectures. The models we attempt to implement are a GAN model for semi-supervised learning on street view house numbers, OPENNMT-tf, a recurrent neural net for translating English to German, and finally Deep residual network (ResNet) for image classification.

The importance of optimizing for minimal latency is two fold. The first would be that most models will be deployed to cloud service providers. Providers, such as Amazon web services will charge for hosting on the basis of service usage (ama). This incentives users to be efficient with their model serving. Additionally, minimizing the latency will increase user satisfaction. When the system is reactive it will in turn "build user confidence, and encourage further interaction" (Bonr et al., 2014). These factors show the importance of why model serving design space should be explored and optimized. The results from the experiments show the advantages of using Bayesian optimization to select a configuration. This is particularly true when the number of concurrent request is low.

In order to find the optimal configuration with respect the latency, we use a naive random configuration method in addition to Bayesian optimization. Through the use of Docker, the system could be configured with a max CPU and memory allotment in addition the the number of replicas. All three tested systems were able to be configured in this way along with other, model specific parameters. Random selection of parameters is straightforward, given all of the provided parameters, choose within their valid range. Bayesian optimization on the other hand, uses machine learning to replace a domain expert (or graduate student) when experimenting with parameters to find the optimal configuration. Bayesian optimization's advantage over other techniques such as grid search is that by design, it does not use the search budget on parameters that have no influence. Because we assume that similar inputs will produce similar outputs we use Gaussian processes to track results from previous configurations and predict with variance what the latency will be given a new configuration. We iterate this process and pick a new configuration to maximize the expected improvement.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. **AUTHORERR: Missing \icmlcorrespondingauthor.**

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Background and Related Work

Optimal configuration for any system is paramount to efficiency. For example, the supervised design-space exploration of hardware design performed can be optimized among multiple objectives from power to size. Multi-objective Pareto-set optimization can be applied to "synthesis-centric methodologies" or CAD (Computer Aided Design) tools as an additional layer to the design process(Lui, 2015). This process found an approximate and component Pareto set by iterative refining the approximation with promising parameters and was guided by synthesis-result estimation with a variety of models. For the purpose of our project, we only seek to optimize latency.

OptEx is a deadline driven cost optimization model for data intensive large-scale processing jobs created for spark. Because service level objective jobs are now performed on cloud services, designers must take into account a myriad of "virtual machine instances" (Sidhanta & Mukhopadhyay, 2016). Given a deadline, OpEx will calculate the cost optimal cluster composition given the virtual machines that the cloud hosting service offers. However, the optimization method of OptEx is an interior point algorithm to minimize the cost of the service level objective.

3. Data

Three data sets are used for the experiments with three models accordingly, details of which are described in subsection 3.1, subsection 3.2 and subsection 3.3

3.1. The Street View House Numbers Dataset

The Street View House Numbers (SVHN) Dataset (Netzer et al., 2011) is an open source offered by Stanford University. It is obtained from house numbers of Google Street View images and used for building machine learning models for object recognition. Door numbers in each Google Street View image are cropped with a bounded square and then each cropped digit is converted into an image. And each such kind of image in SVHN is labeled with a number between 0 and 9. The training set (73257 images) is used to train the Generative adversarial network (GAN) model (GAN), which is used in subsection 5.1 to recognize images of door numbers. In this project, only a part of testing set (3413 images) is used in the experiment.

3.2. OpenNMT-tf using WMT'14 English German Data

OpenNMT (Klein et al., 2017) is an open source (MIT) initiative for neural machine translation and neural sequence modeling. OpenNMT-tf [1] is a general purpose sequence learning toolkit using TensorFlow designed to mainly for neural machine translation but also supports sequence to sequence mapping, sequence tagging and sequence classification. OpenNMT-tf is based on Recurrent Networks and use Tensorflow Serving to serve the pretrained network for converting English to German. This network is model is exported as tensorflow graph and can only be used for inference. For inference, WMT'14 English-German data [2] (newtest2015.en) is passed to the network for inference and the network performance is observed.

3.3. CIFAR-10 dataset

CIFAR-10 [3] is an image database of 60000 32x32 colour images in 10 classes, with 6000 images per class. The respective classes of images are airplane, automobile, bird, cat, deer, dog, frog horse, ship, and truck. This data set was chosen because the classes are completely mutually exclusive (Krizhevsky et al., 2009). This prevents overlap between categories such as truck and automobile and made it simpler to measure accuracy.

4. Methods

Given the number of concurrent requests, we iteratively search for approximate optimal configuration of server center by using Bayesian optimization such that user perspective latency could be minimized. Here, latency is measured as average round-trip time of a request. A round-trip time measures the time elapsed from the point of sending out a request to the point of receiving a response from the server. For each iteration of Bayesian optimization process, a large amount of requests will be sent to a server center, each of whose round-trip time will be collected and then used to compute the average round-trip time. Because Bayesian optimization can only handle maximization problems, we convert our problem to maximize the negative average round-trip time, as shown in equation 1.

$$\begin{aligned} \underset{\mathbf{P}}{\text{maximize}} \quad & -\mathit{average\_round\_trip\_time}(\mathbf{P}) \\ \text{subject to} \quad & \mathit{constraint}(\mathbf{P}) \ . \end{aligned} \tag{1}$$

Since three machine-learning models are investigated in our project, configuration spaces for the serving of three models have some differences. For example, for the GAN model, the configurable parameters are only the number of servers, maximum capacity of CPU and maximum memory in each server; while, for OPENNMT-tf model, two extra configurable parameters are considered, which are GPU frequency and the size of batching individual model inference requests.

[1] https://github.com/OpenNMT/OpenNMT-tf/

[2] https://nlp.stanford.edu/projects/nmt/

[3] https://www.cs.toronto.edu/~kriz/cifar.html

More details are discussed in the corresponding parts in ~~the~~ section 5.

We use TensorFlow serving (Goo) to serve machine learning models. TensorFlow serving is a flexible serving system for machine-learning models, could easily deploy changes of models and experiments without changing the architecture of the system. To enable the serving of multiple copies of a model, we apply docker (Doc) to instantiate multiple containers, each of which runs with one model. The overview of the serving system is shown in Fig. 1.

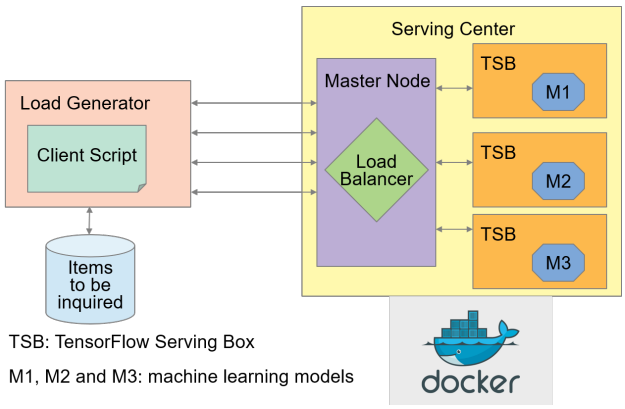

Figure 1. Architecture Overview of Model Serving

5. Experiments

In this section, experiments are conducted with three separate DNN models, GAN, OpenNMT and Resnet. The measurement is the same, the latency of a request, while the configuration space for each serving model is slightly different. But the overall experiment setting is the same for all three models, using the combination of TensorFlow serving and Docker containerization. The Bayesian optimization python package used here is obtained from Github (BO).

5.1. GAN model

The Impact of Concurrency. How will the number of concurrent requests affect the latency for the serving GAN model? We run experiments with different numbers ([1, 5, 10, 20, 30, 40, 50, 60]) of concurrent requests, where, in each experiment, the total number of requests is the number of testing images, 3413. An interesting observation is that, for each experiment, round-trip time of 3413 requests are clustered. The higher number of concurrency, the more number of clusters. The experimental results of using 5 concurrent requests and 20 concurrent requests are shown in Fig. 2 and Fig. 3. While, the averaged round-trip time is linearly increased with the number of concurrent requests, as shown in Fig. 4.

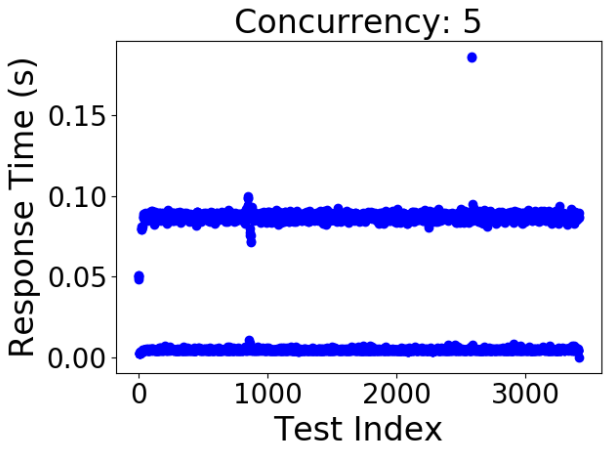

Figure 2. Round-trip times of sending 5 concurrent requests.

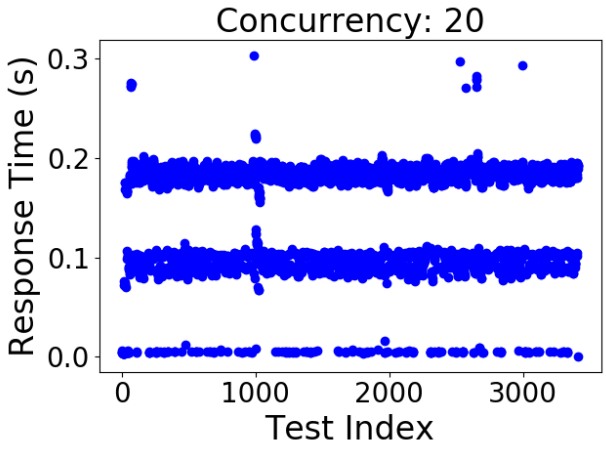

Figure 3. Round-trip times of sending 20 concurrent requests.

Problem Formulation. For the serving of the GAN model, three configuration parameters are considered, which are the number of replicas, N, the CPU capacity, C, and memory capacity of each replica, M. We use (N, C, M) to denote one configuration. For example, $(2, 0.5, 500)$ means that two replicas will be instantiated, each of which has 0.5 CPU capacity and 500M memory capacity. During experiments with the configuration $(1, 1, 2000)$, we observe that CPU usage changes in the range $[0.3, 0.85]$ and memory usage bounces between 500M and 800M. Since we use more-than-one replicas in the following experiments, these two ranges could be considered as the bounded constraints when configuring CPU and memory capacities. The problem is formulated in equation 2.

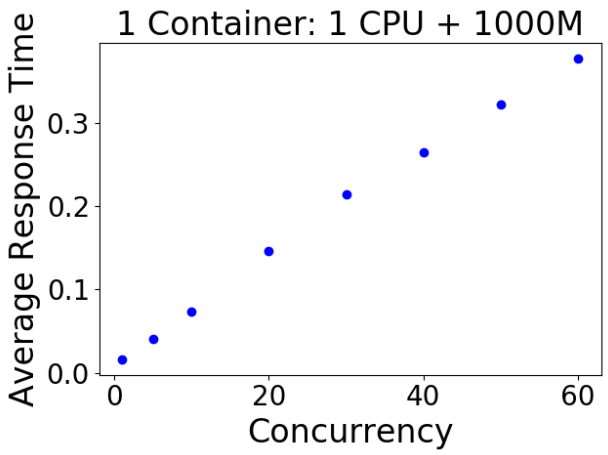

Figure 4. Average round-trip time under 1, 5, 10, 20, 30, 40, 50 and 60 of concurrent requests.

$$\begin{aligned} \underset{(\mathbf{N},\mathbf{C},\mathbf{M})}{\text{maximize}} \quad & -average\_round\_trip\_time((\mathbf{N},\mathbf{C},\mathbf{M})) \\ \text{subject to} \quad & \mathbf{N} \in [1,8], N \in Z+ \\ & \mathbf{C} \in [0.3,0.85], C \in R+ \\ & \mathbf{M} \in [500,800] \in Z+ \end{aligned} \tag{2}$$

Experiment Environment and Setting. The influence of the number of concurrent requests on the average round-trip time is monotonically increasing. But is it a good idea to just linearly change each of the three configurable parameters when the number of concurrent requests changes? Actually, the answer is no. Here, we apply Bayesian optimization to approach an approximately optimal configuration. The setting of Bayesian optimization are: 1) 25 initial points, 2) 25 iterations and 3) using "Expected Improvement" as the acquisition function and $1e-4$ as the stopping criterion. And the hardware environment for the experiments are: 18 cores (2 threads per core) and 32G memory.

Result and Discussion. We conduct experiments with 8 numbers of concurrent requests, which is mentioned earlier. For all testing scenarios, Bayesian optimization gives a better configuration solution than a random configuration. Considering page limits, we only present the process of running Bayesian optimization with a concurrent requests of 20. As seen in Fig. 5, the red points denote 25 random configurations, while the rest 25 points (blue and green points) represent configurations determined by Bayesian optimization along 25 iterations. And the green point, named best BP point, is the best configuration found by Bayesian optimization. To see the possible trend of performance gain of best BP point over best random point along concurrency, we assembly such pair of points for each of 8 experiments and plot in Fig. 6. As we can see, the performance gain is trending up when the number of concurrency increase.

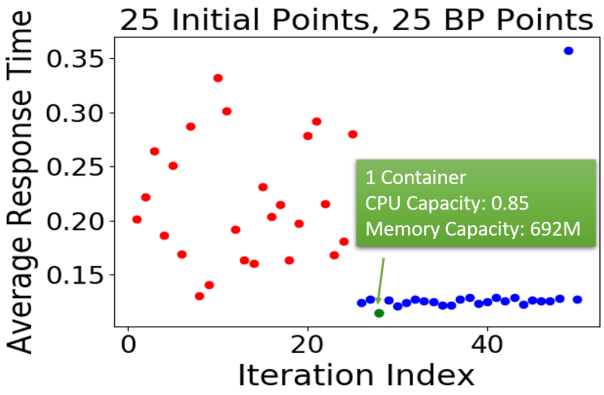

Figure 5. Bayesian optimization process when sending 20 concurrent requests.

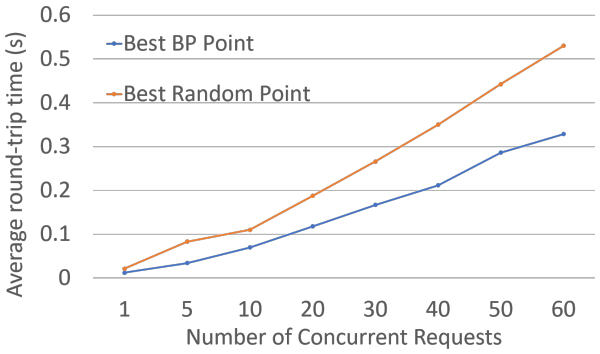

Figure 6. Trend of performance gain.

5.2. OpenNMT-tf: Neural Machine Translation

OpenNMT-tf has exported model that can be served using Tensorflow Serving. For production purposes, tensorflow serving is generally deployed as a nvidia-docker image. Nvidia-docker image is similar to docker image which can run upon nvidia-gpu. We try to find the effect of different configurations on the average inference time.

Configuration Paramters

While serving a TensorFlow model, batching individual model inference requests together can be important for performance. In particular, batching is necessary to unlock the high throughput promised by hardware accelerators such as GPUs. Tensorflow serving uses batching scheduler to create a synchronous API and create a callback on a seperate thread to process the batches. We can change some

parameters of batch scheduler which can change the performance of tensorflow docker and add these parameters to the system configuration. Since we are using nvidia-docker, we add some docker parameters as well. Finally, inference time also depends on the GPU frequency. The configuration parameters for the system are as follows:

- max_batch_size: The maximum size of any batch. This parameter governs the throughput/latency tradeoff, and also avoids having batches that are so large they exceed some resource constraint (e.g. GPU memory to hold a batch's data).
- batch_timeout_micros: The maximum amount of time to wait before executing a batch (even if it hasn't reached max_batch_size). Used to rein in tail latency.
- No. of Replicas: No. of replicas of the nvidia-docker container to be used for inference in production environment.
- Max CPU Utilization: The maximum fraction of CPU that a single docker container can use. The CPU usage by each container is limited to this value.
- Max Memory Utilization: The maximum memory that a single docker container can utilize in MB.
- GPU Frequency: Application GPU Frequency of the GPU device.

Environmental Parameters

Environmental parameters are the factors that are outside the control of the system but affect the system performance. For inference, the number of requests sent will affect the inference time. If large number of requests are sent, then the load will increase and so will be the inference time. For higher load, we need more resources. Also, the batch size of the request can change the inference time as more data needs to be handled. The batch size is a compromise between throughput and latency. With more batch size, we can get more inferences at a point but at the cost of increased latency. Since the batch size of request depends on the user, it is considered as environmental variable. Thus we consider two enviromental variables:

- Concurrency: Number of asynchronous concurrent request
- Batch Size: Batch Size of Input Request

Random Configuration Sampling

First we tried to check for some possible configurations how the inference time changes. Since inference time is also dependent on environmental parameters, it would be impossibe to generate configurations for all sets of environmental parameters. We collected some data by varying both environmental and configuration parameters. Some typical data collected is summarized in Table 1.

During the random smapling, the domain of the configuraiton variables is fixed. as follows:

- max_batch_size: discrete variable from 4, 8, 16, 32, 64, 128
- max_timeout(μs): continuous variable from 100 to 10000
- Relicas: discrete variable from 1 to 20
- Max CPU: continuous variable from 0.2 to 0.6
- Max Memory (MB): discrete variable from 1000 to 10000
- GPU Frequency: discrete variable from a list a permissible GPU application clock frequencies

The average inference time is calculated as per request per batch.

$$Avg.\,Inference\,Time = \frac{Time\ taken\ for\ all\ requests}{\#requests * batch\_size}$$

It was evident after some data collection that on increasing the no. of concurrent requests and batch size significantly increases the inference time. For the sake of experiment time, concurrency was set to 5 and batch size to 4. Moreover, if we create large number of container replicas that required the communication overhead becomes significant. Thus, max number of replicas was set to 5. Since OpenNMT-tf is RNN framework the prediction also depend on the number of words in each batch. The input to the model is obtained by parsing a WMT'14 English text file into sentences. Then, these sentences arre broken down into word tokens. For instance, in an input of batch size 4, there will be 4 randomly selected sentences and width of each batch is maximum no. of words among those sentences. Smaller sentences are padded with space to make sentences of equal length. Thus, prediction time also depends on the width of the batch. Since, the sentence on the dataset were of relatable lenght, we do not consider sentence length as a parameter. Moreover, this can be considered as randomness in the data that can vary the inference time. If we train/optimize any model for configuration space, this randomness might help to create a more robust model.

Environmental Variables		Configuration Parameters						
Concurrency	Batch Size	Max Batch Size	Batch Timeout (in μs)	Replicas	Max CPU (Fraction)	Max Memory (MB)	GPU Frequency	Avg. Inference Time
2	2	64	9080	16	0.5	8700	288	69.65
5	4	8	2658	4	0.39	4200	169	34.78
5	4	16	2622	4	0.36	4900	324	82.69
10	2	128	8837	7	0.5	4300	1306	75.90
10	2	8	6278	19	0.5	5700	771	112.63
10	4	32	2816	2	0.22	2700	1267	226.87
10	4	32	8998	6	0.5	5900	366	98.38
10	8	64	1874	4	0.5	8500	1032	56.65
10	8	64	5249	1	0.36	1700	666	107.38

Table 1. Data Collected Using Random Sampling

Bayesian Optimization First a few more samples with concurrency 5 and batch size 4 were obtained and bayesian optimization was performed to obtain the best configuraiton for these environmental parameters. For bayesian optimization, Gpy (GPy, since 2012) was used to perform bayesian optimization for 5 iterations. The best performing model was recorded in a file and bayesian optimization was again run. After multiple runs, the optimized configuration did not change and converged. The optimal configuration was found as:

- max_batch_size : 32
- batch_timeout (μs): 2778
- Replicas: 1
- Max CPU: 0.6
- Max memory (MB): 4400
- GPU Freq: 173 MHz

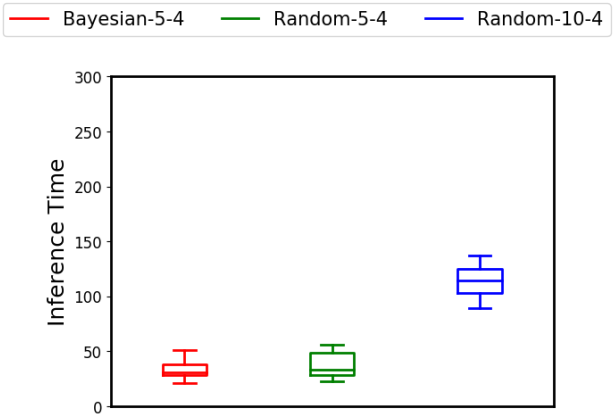

Figure 7. Box plot of average infernce time comparing the best run of bayesian optimization in red and Random values in green with concurrency 5 and batch size 4. The average inference time with random configuration, concurrency 10 and batch size 4 is shown in blue for reference.

Fig. 7 shows the box plot of inference time of best runs from bayesian optimization and random configurations with concurrency 5 and batch size 4. Even though, there was not much improvement in the median, we can see that configurations obtained from bayesian optimization are more consistent and have low variance.

Exploration vs Exploitation Even though bayesian optimization consistently finds lower inference time, it might run into local optima rather than global. Also, the random data we collected might not have enough initialization information for global minimization. We plan to employ epsilon greedy sampling policy to explore new random configuration at the same time exploit the best configurations. The epsilon greedy sampling policy

- randomly sample with probability ϵ
- exploit the best configuration with probability 1-ϵ

5.3. Resnet

Deep residual networks or ResNets enable training of very deep convolutional neural networks. For this experiment, a pre-trained network provided by TensorFlow [4] was used used to serve the model. The client would then generate a docker compose file. Finally it would generate a load from the CIFAR-10 dataset to query the server with a set concurrency. Unfortunately, the experiment was not able to be realistically conducted for a numerous reasons. the First, being that the testing machine could only realistically handle single threaded request. Upon increasing the concurrency beyond one, the server would experience a TCP timeout error. Additionally, almost all of the single concurrency model server configurations failed to succeeded. This brings up the next issue, the size of valid configurations on the machine. Upon beginning random sampling, the docker-compose file did not give possible ranges for parameters [5]. It took con-

[4] http://download.tensorflow.org/models/official/20181001_resnet/savedmodels/resnet_v2_fp32_savedmodel_NHWC_jpg.tar.gz

[5] https://docs.docker.com/compose/compose-file/#service-configuration-reference

siderable effort of trial and error just to get a reasonable range of parameters, specifically with respect to how many replicas a model server is able to create. These limitations prevented any meaningful data from being collected.

6. Conclusion

The aim of this project is to explore the configuration space of model serving to minimize the latency from users' perspective. Three DNN models, GAN, OPENNMT-tf, and ResNet, are investigated. We use Tensorflow serving to serve each model, and Docker to simulate an experimental setting of multiple servers in a server center. To obtain the optimal configuration, Bayesian optimization is adopted. Based on our experiments, we find that performance gain of best configuration given by Bayesian optimization over the best random configuration will trend up when the number of concurrent requests goes up.

References

Bayesian optimization python package. URL https://github.com/fmfn/BayesianOptimization.

Docker. URL https://www.docker.com/.

Gan model for discriminating door-number images. URL https://github.com/Vetal1977/tf_serving_example.

Tensorflow serving. URL https://www.tensorflow.org/serving/.

How does aws pricing work. URL https://aws.amazon.com/pricing/?nc2=h_ql_pr.

Bonr, J., Farley, D., Kuhn, R., and Thompson, M. The reactive manifesto, 2014. URL https://www.reactivemanifesto.org/.

GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

Klein, G., Kim, Y., Deng, Y., Crego, J. M., Senellart, J., and Rush, A. M. Opennmt: Open-source toolkit for neural machine translation. In *ACL*, 2017.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Lui, H.-Y. *Supervised Design-Space Exploration*. PhD thesis, 2015.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL http://ufldl.stanford.edu/housenumbers.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pp. 2503–2511, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969442.2969519.

Sidhanta, W. G. and Mukhopadhyay, S. *OptEx: A Deadline-Aware Cost Optimization Model for Spark*. PhD thesis, Louisiana State University, University of Waterloo, 2016.