# AI_System_UCB

## Kinds of AI-Systems research:

- AI for system: Advances AI are being used to address fundamental challenges in systems.
  - E.g. RL for Pandas code generation, SQL join planning, Network packet calssification, Auto scaling, Bnadit algorithms for radio link adaptation. wireless link quality estimation, multi-task learning for straggler, VM Selection using trees.
- System for AI: Anvances in systems are enabling substantial progess in AI
  - Distributed Training, Distributed Auto-ML, The run-time system for CNN...

## AI sys big ideas:

- For system:
  - Sources of complexity in systems:
    - Emergent properties: 系统中的单个组件不能明显的反映出整个系统的属性
    - Propagation of Effecrs: 一个组件的结构很有可能影响到另外一个
    - Incommensurate Scaling: not all parts of a system scale at the same rate
    - Tradeoffs: time, space ...
    - Excessive Generality: if it is good for everything, it is good for nothing
  - Coping with complexity:
    - laying and hierarchy, using modularity and abstraction
    - iteration or incrementally
    - adopt sweeping simplications

    - minimal design(保持设计的简单性)
    - The robust principle: Betolerant of inputs and strict on outputs

## Readings:

*MLSys: The New Fontier of Machine Learning Systems*

- Bottle-neck of the **systems that support the ML in real world applications: ML-based app require distinctly new types of software, hardware and engineering system to support them.**
- “ML” is called “Software 2.0” because:
    - The developed way is different: shifting models and adusting parameters instead of coding
    - diffenrent deployed way: utilizing specialized hardware, new types of quality assurance methods, new end-to-end workflow.

  → All this changes bring challenges and opportunities around high-level interface for ML development, low-level systems for executing ML models, and interfaces for embedding learned components in the middle of traditional computer system code. (Very Important)

因此这篇论文旨在创建一个新的community: the itersection between system and ML.

另外这篇论文为这个community提供了一个许多新的问题：

1. How should software systems be designed to support the full machine learning lifecycle, from programming interfaces and data preprocessing to output interpretation, debugging and monitoring?
2. How should hardware systems be designed for machien learning?
3. How should machine learning systems be designed to satisfy metrics beyond predictive accuracy, such as power and memory effiececy, accessibility, cost, latency, privacy, security, fairness, and interpretability?

另外这篇论文还提出了关于这个community的另外一个方向：ML for system.

Examples include replacing the data structures, heuristics, or hand-tuned parameters used in low-level systems like operating systems, compilers and storage systems with learned models.

**Why now? The rise of Full Stack（全栈的） Bottlenecks in ML**

- system ML work has moved to the forefront recently due to leaps in machine learning perfromance on challenging benchmark tasks(指的是图像识别这些工作上的飞跃), and the growing realization that a range of new sysmtes up and down the computing stack are needed to translate this academic promise to real-world practice.
- 也就是说，ML在各领域方面的成功，要求system在high level和low level上都为ML融入real world 提供帮助。

---

*A Few Useful Things to Know about ML*

- Learning = Representation + Evaluation + Optimization

---

## Machine learning Life-cycle

*Hidden Technical Debt in Machine Learning System*

Thesis: This paper focus on system-level interaction and interfaces as an area where ML technical debt may rapidly accumulate.

- First Tech-debt: Complex Modle Erode Boundaries (of the abstraction boundaries抽象和实现的界限)
    - Because the desired behavious cannot be effectively expressed in software logic without dependency on external data.
    - Several ways that may significantly increase technical debt in ML systems:
        - Complex Models Erodes Boudarires
            - Entanglement(纠缠): 改动一个数据会对其他的数据也产生很大的影响，比如调整一个feature的权重，那么其他feature的权重也要改变，或者加入一个新的feature，那么会牵扯到很多其他数据的改变（data dependency）→ called CACE principle: Change anthign changes everything
                - Ways to weak this debt: 1. Isolate models and serve ensembles. 2. Focus on detecting changes in prediction behavious as they occur.
            - Correction Cascades

        - Undeclared Consumers: (未经权限控制的访问，改变了模型，并且这个是很难detect的)，但也有方法：Acess restrictions or strict service-level agreement (SLAs)
    - Data dependency cost > Code denpendency
        - 传统的Code denpendency 有相关的static algorithms（compilers and linkers）来分析, 但对于Data dependency，我们缺乏这样的方法
            - 方法：create a frozen version, 用这个version来分析dependency。uddate这个version直到这个改变被审批
        - Underutilized Data Dependency:
            - 应该说是模型update的过程中一些冗余（未被考虑的data dependency导致的）
            - The ways underutilized data dependencies can creep into a model:
                - Legacy Features: 一个新加入的feature导致一个旧的feature redudent
                - Boundles Features
                - To improve the accuracy even when the accuracy gain is very samll or when the complexity overhead might be high.
                - Correlated Features
            - 解决方法：underutilized dependencies can be detected via exhaustive leave-one-feature out evaluations. These should be run regularly to identift and remove unnessary features.
            - 也存在一些静态的分析方法
        - Feedback loops
            - 因为ML system的一大特点是它每次迭代的产出结果会影响到它后面的behaviour，所以it is difficult to predict the behaviour of a given model before it is released.
                - Direct Feedback Loops:
                - Hidden Feedback Loops
        - ML-System Anti-Patterns

    - Glue code, 在采用一个general-purpose packages的时候，it tends to freeze a system to the peculiarities of a specific package; testing alternativees may become prohibitively expensive.
        - An important solution is to wrap black-box packages into common APIs
    - pipeline jungles
    - Dead experimental codepaths
    - Abstraction Debt: There is a dsitinct lack of strong abstraction to support ML systems, especially the distributed learning, there remains a lack of widely accepted abstractions.
        - The lack of standard abstraction makes it all too easy to blur the lines between components.
- Configuration Debt
    - A wide ranges of configurable options, including which features are used, how data is selected, a wude variety of algorigthm-specific learning settings, potential pre- or post-processing, verification methods.
        - The principles of good configuration systems:
            - It should be easy to specify a configuration as a small change from a previous configurations.
            - It should be hard to make manual errors, omissions, or oversights.
            - It should be easy to see, visually, the difference in configuration between two models.
            - It should be possible to detect unused or redundant settings.
            - Configurations should undergo a full code review and be checked into a repository.
- Dealing with Changes in the External World.

        - Fixed Threshold in Dynamic Systems: It is often necessary to pick a decision threshold for a given model to perfrom some action. (e.g. true or false, email spam or not spam)
        - Prediction Bias
        - Action Limits
        - Up-Stream Producers

        因为ML system需要不停的和外部的世界发生交互，ML system也需要不断的对changes in real-world作出反应，这通常需要人工的调整。但对于一些time-sensitive issues, 这个是很brittle的。所以，creating system to that allow automated response without direct human intercention is often well worth the investment.

- Other Areas of ML-related Debt
    - Data Testing Debt: 因为在ML system中data取代了code的重要性，对data的testing就有很强的重要性
    - Reproducibility Debt: re-run similar experiments and get similar results.
    - Process management Debt: 在一个ML system中，我们存在很多的Model在同时训练的情况，如何进行process management就很重要
    - Cultural Debt: There is sometimes a hardline between ML research and engineering, （比如research可能会想尽一切办法提高准确率，但这样的代价可能是其应用在工业上的价值，或scalibility的价值（some debts). It is important to create team cultures that reward deletion of features, reduction of complexity, improvements in reproducbility, stability and monitroing to the same degree that improvements in accuracy are valued.

Conclusion:

1. How easily can an entirely, new algorithms approch be tested at full scale?
2. What is the reansitive closure of all data dependency?
3. How precisely can the impact of a new change to the system be measured?
4. Does improving one model or signal degree others?
5. How quickly can new members of the team be brought up to speed?

*TFX: A TensorFlow-Based Production-Scale Machine Learning Platform*

The requirement to build such a paltform:

1. Building one machine learning platform for many different learning tasks
2. Continuous trainning and serving: not only trained over fixed data, but also over evolving data.
3. Human-in-the-loop: simple user interfaces. 同时也要兼顾various levels of ml expertise
4. 现实生产环境中的各种debt和考虑

- The design principles for TFX:
  - One machine learning platform for many learning tasks:
    - Use Tensorflow as the trainer but the platform desgin is not linited to this specific library. Tensorflow provides implementations of linear, deep, linear and deep combined, tree-based, sequential, multi-tower, multi-head, etc. BUT this is not enough, There are requirements on all other components: Data analysis, validation, and visualization tools need to support sparse, dense, or sequence data. Model validation, evaluation and serving tools need to support all kinds of inference type, including regerssion, classification, and sequence.
  - Continuous trainning:
    - Data visitation can be configured to be static or dynamic.
    - Warm-startign: initializesa subset of model parameters from a previous state.
  - Easy-to-use configuration and tools:
  - Production-level reliability and scalability
    - If the platform handles and encapsulates the complexity of machine learning deployment, engineers and scientists have more time to focus on the modeling task.
- Data analysis, transformation, and validation.
  - 因为trainning data对于模型很重要，我们需要及时检查出数据中的异常

        - 这个components的难点在于：wide range of data-analysis; ;deploy a basic set of useful checks without requiring excessive customization; help the user to monitor and react to the data.
        - 3.1: Data Anlysis: 包括很多对数据的描述，包括一些统计上的描述mean, var...
        - 3.2: Data Transformation: 比如从feature 映射到数字，vocabularies.
        - 3.3: Data validation: 检查数据时候healthy
    - Modle trainning
        - 4.1: Warmming state: 类似transfer learning， 这在continuously learning中也有很大的用处，加速模型更新换代的速度。
        - 4.2: High level model specification API
    - Mdel evaluation and validation
        - Evaluation包括两个维度：safe to serve(the model should not crashed in the serving system) + prediction quality
    - Model serving:
        - Serving systems for production environments require lw latency, high efficiency, horizontal scalability, reliability and robustness.
        - 6.1: multitenancy: ser multiple machine-learned models concurrently
            - provide soft model-isolation → the performance characteristics of one model has minimal impact on others.
    - 利用这个系统构建了Google Play的recommendation system。
- In data science, 80% of time spent prepare data, 20% of time spent complain about the need for prepare data —- Borat
- Fine Tuning: Using small learning rates to train pre-trained or partially pre-trained model for a new dataset or prediction task → enbles both faster training and improved accuracy.

---

Week 4: Lec

- Why do ML in a Database system

    - Proximity（亲近） to data → Avoid data dupliation → inconsistency
    - DB is Opimized for efficient acess and manipulation of data
    - Predictions With data: trained models often used with data in the database
    - Security: control who and what models have access to what data
- Challenges of Learning in DB
    - Abstraction: How does DB expose algorithm access data?
    - Access Patterns: How does algorithm access data?
    - Cost Models and Learning: How does db aid in optimizing learning algorithm execution?
    - Data type: does data fit in the relation models?

→ Most database systems have support for analytics and ML.

Algorithm也要兼顾out-of-core computation (这不是os干的吗)？→ DB are typically designed to operate on databases larger than main memory. → algorithm must manage memory buffers and disk.

---

*Towards a Unified Architecture for in-RDBMS Analytics ——> BISMARCK(the architecture's name)*

- The challenge is that: each new statistical technique must be implemetned form scratch in the RDBMS. → due to lack of a unified architecture for in-database analytics
- A keybottle-neck of the development: 每当引入一个new data analytics technique requires several ad hoc steps: a new solver is emplyed that has new memory requirements, new data access methods, etc. → little code reuse
- The contributions of this work:
    - Describe a novel unified architecture, BISMARCK, for intergrating many data analytics tasks formulated as Incremental Gradient Descent into an RDBMS using featyres availble in almost every commercial and open-source system.
    - develop a strategy to improve the performace from the "bad" order of data
    - study how to adapt existing approaches to make BISMARCK run in parallel.

- The need to shuffle the data:
  - 因为关系型数据库里的数据通常是以某种方便数据库操作的方式储存起来的，这种ordered data很有导致不好的训练结构（比如前一半全是<0的数，后一半全是>0的数），所以我们在训练前要进行shuffle，但shuffle的开销很大，所以BISMARCK只在训练开始前suffle一次，而不是每个epoch都shuffle一次。
- Parallelizing Gradient Computations:
  - Pure UDA
  - Shared-Memory UDA
    - The model to be learned is maintained in shared memory and is concurrently updated by parallel threads operating on segments of the data.
- 它的实验设计分为了两个大方向：
  - Development Overhead: 开发新的ML model只需要改动几行代码
  - Runtime Overhead:
    1. Compare it with DBs provides native analytics tools (focus on end to end runtimes of the various tools on LR, SVM, LMF)
    2. Scalibility
    3. Data order
    4. different shuffling way
    5. paralleling

---

**Week 5: Lec**

Backward Differentiation: Performs well when computing large gradients relative number of function outputs.

- Requires caching or recomputing intermediate activations from forward pass.

---

*TensorFlow: A system for large-scale machine learning*

Main contents: The TensorFlow dataflow model and demonstrate the compelling performance that Tensorflow achieves for several real-world applications.

**Intro:**

The success of the ML lies on more sophisticated machine learning models + availability of large datasets + development of software platform

**Background and motivation:**

Previous system: DistBelief:

- parameter server
- 它的limitation包括（其实主要是限制了研究者的自由发挥）
  - Defining new layers(用c++实现的，a barrier for machine learning researchers who seek to experiment with new layer architectures)
  - Refining the trainning algorithms
  - definning new trainning algorithm: 他用gradient descent的话是要往后传播，但对于更复杂的算法，比如recurrent neural networks, RL, 这些就不支持
  - 在并行计算上是heavy-weight的，并且在不同的环境下不一定能做到统一，但Tensorflow provides a single programming model and runtime system for all of these environments.

Design principles:

- Dataflow graphs of primitive operators: 提供了很多原始的operations，比如矩阵乘法什么的 → This approach makes it easier for users to compose novel layers using a high-levl scripting interface. (自由度更高)
- Deferred execution: 因为Tensorflow分成了两个阶段：第一个是define the program as a symbolic dataflow graph with placeholders for the input data and variables that represent the state. 第二个阶段是：Executes an optimized version of the program on the set of available devices. 如果defer execution until the entire program is available, TF就能optimize the execution phase by using global information about the computation.
- Common abstraction for heterogeneous accelerators

Partial and concurrent execution

- Each invocation of the API is called a step, and Tensorflow supports multiple concurrent steps on the same graph. Stateful operations allow steps to share data and synchronize when necessary.

Distributed execution:

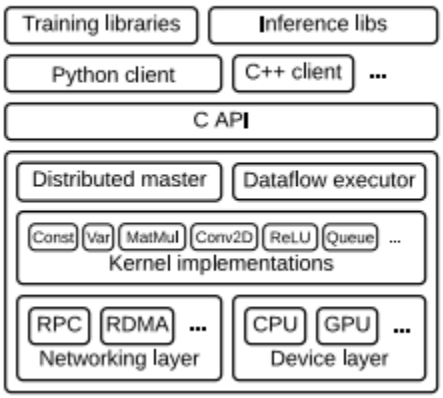
- Tensorflow permits great flexibility in how operations in the dataflow graph are mapped to devices: While simple heuristics yield adequate performance for novice users, expert users can optimize performance by manually placing operations to balance the computation, memory and network requirement across multiple tasks and multiple devices within those tasks.
- Tensorflow also support automatic differentiation.

Fault tolerance:

- implement user-level checkpointing for fault tolerance, using two operations in the graph. （其实是提供了两个API：Save, Restore, 用户可以用定时用Save来存下checkpoint file）.

Synchronous replica coordinate:

**The implementation of TensorFlow:**




- The core TensorFlow runtime is a cross-platform library.
- The core TF library is implemented in C++ for portability and performance.
- The distributed master translates user requests into execution across a set of tasks.
  - partition, cache and applies standard optimizations.
- Dataflow executor in each task handles requests from the master, and schedules the execution of the kernels that comprise a local subgraph.
- users can register additional kernels that provide an efficient implementation written in C++

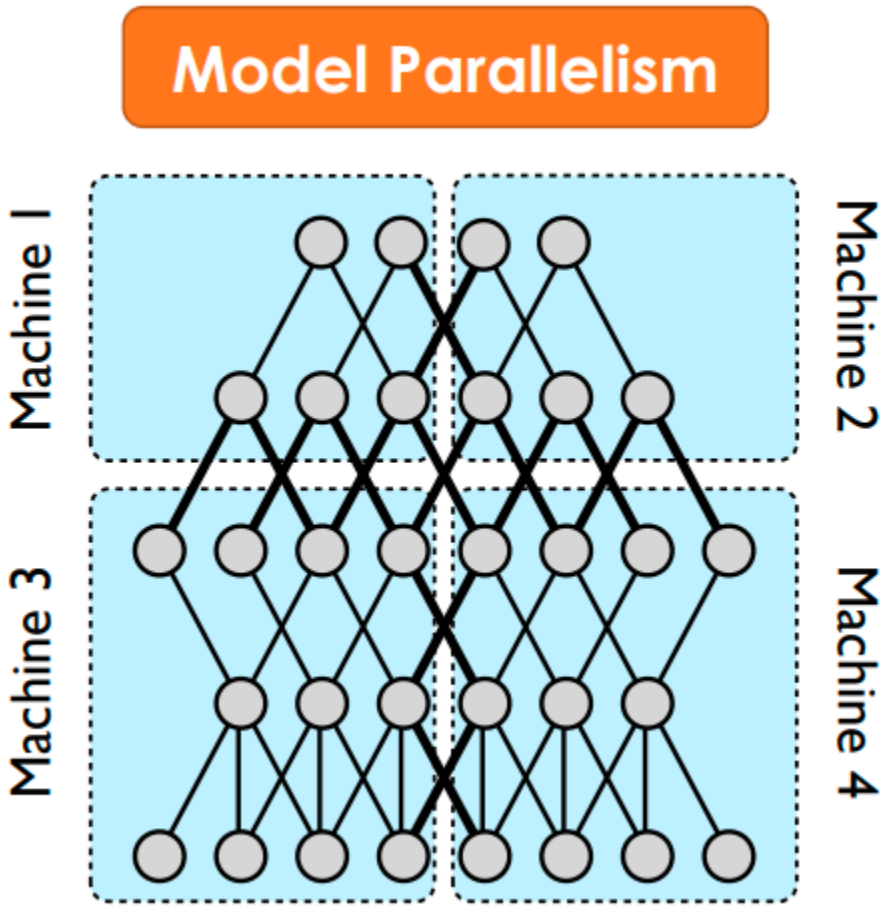
## Week 6: Distributed Trainning

- Distributed Trainning: Trainning across multiple devices → different local and remote memory speeds / network
- Why do we need distributed trainning?
  - Fast trainning by parallel computing
  - Additional memory for larger model
  - Reduce or eliminate data movement(Privacy → federated learning, limited bandwidth)

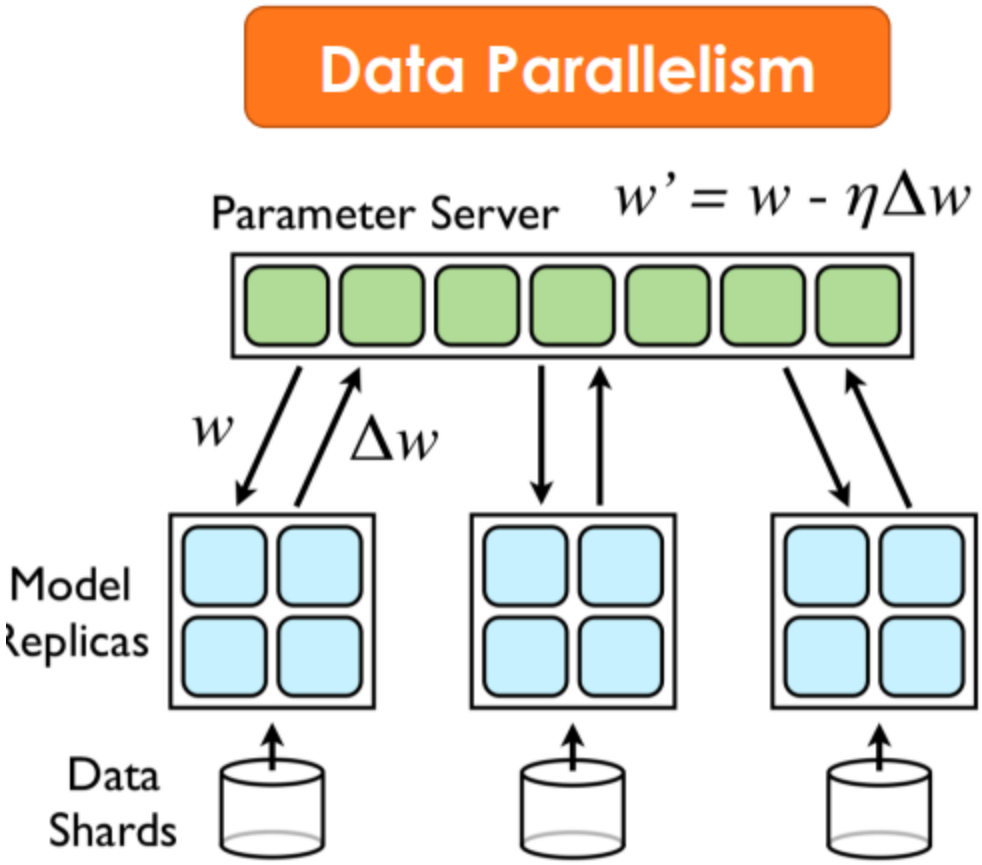
Metrics of Success → Minimize trainning time.

两种parallism：

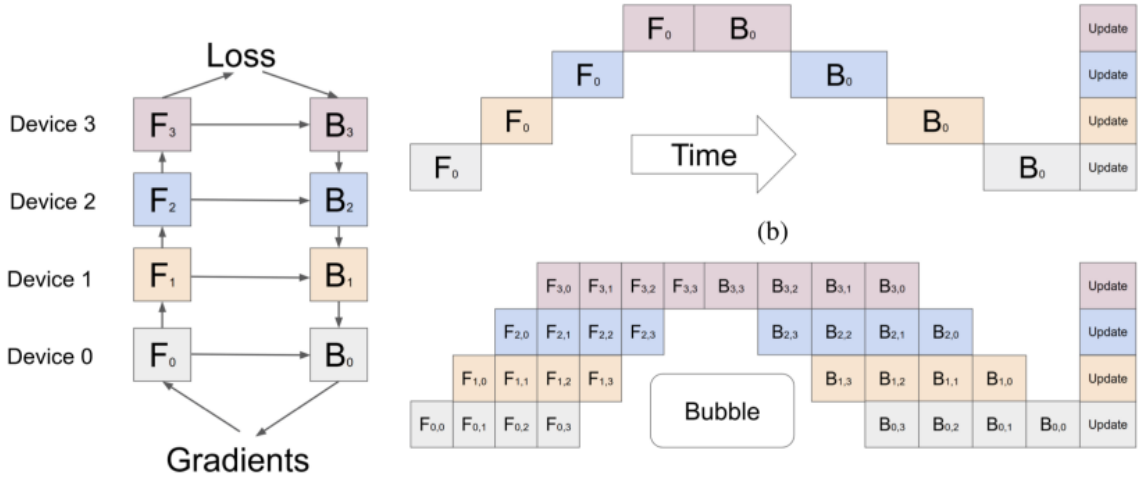
- Model Parallelism



- Data parallelism




- Pipeline Parallelism:



- Operator Level Parallelism:

  Exploiting the parallelism within linear algebra and concolution operations

---

*Scaling Distributed Machine Learning with the Parameter Server*

(Paper describing the parameter server system)

- Breif Architecture
  - Both data and workloads are distributed over worker nodes, while the server nodes manintain globally shared parameters, represented as dense or sparse vector and matrices.
  - manage asynchronous data communication between nodes, supports flexible consistency models, elastic scalability, and continuous fault tolerance.
- The chanllenges that shared-parameters impose:
  - Accessing the parameters requires an enormous amount of network bandwidth.
  - Many machine learning algorithms are sequential. The resulting barriers hurt performance when the cost of synchronization and machine latency is high.
  - At scale, fault tolerance is critical. Learning tasks are often performed in a cloud environment where machines can be unreliable and jobs can be preempted.
- Contributions:
  - by factoring out commonly required components of machine learning systems, it enables application-specific code to remain consice
  - as a shared platform to target for systems-level optimizations, it provides a robust, versatile, and high-performance implementation capable of handling a diverse array of algorithms from sparse logistic regression to topic models and distributed skectching.
- The paramter server provides five key features:
  - Efficient communication (使用asynchronous communication不会block communication)
  - Flexible consistency models不懂这是什么
  - Elastic Scalability:

        - New nodes can be added without restarting the running framework
    - Fault Tolerance and Durability:
        - Recovery from failures within 1s
    - Ease of Use
- Two challenges arise in constructing a high performance parameter serve system:
    - Communication
        - 感觉是数据类型的问题，一次传一个大的数据类型（segment, row of matrix）会比一个个传parameter效率更高
        - Fault tolerance
- Risk minimization
    - 在训练的过程中，in each iteration, every worker independently uses its own training data to determine what changes should be made to w in order to get closet to an optimal value.
        - 因为每个worker是自己算自己的，the system needs a mechanism to allow these updates to mix.
- Architecture of a parameter server communication with several groups of workers

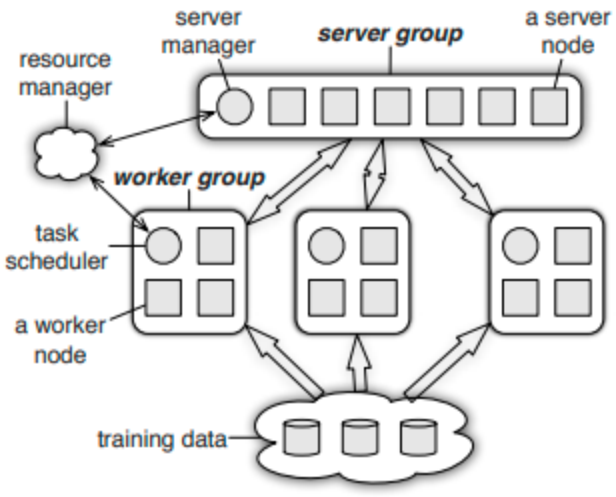



Figure 4: Architecture of a parameter server communicating with several groups of workers.

    - 每个server node储存了一部分的globally shared parameters
        - The parameter server is designed to simplify developing distributed machine learning applications.
    - 每个worker group代表了一个app(也可以多个worker group运行同一个ML)，每个node里有一部分的trainning data来进行学习。
    - Data is sent between nodes using push and pull (这很像每一个node就是一个core, push and pull are like send and recv)
        - In the algorithm specified in the paper, each worker pushes its entire local gradient into the servers, and then pulls the updated weight back.
    - User-defined functions on the server → server nodes can execute user defined function
    - Asychronous Tasks and Dependency:
        - 这里的Dependency是指用户可以自己给tasks增加dependency，（比如execute-after-finished可以让两个工作串行化执行）。
    - Flexible Consistency:
        - 这里指在并行化计算中可能导致的数据inconsistent
            - the parameter server gives the algorithm designer flexibility in defining consistency models.
- Implemention
    - The servers store the parameters using consistent hashing.
    - For fault tolerance, entries are replicated using chain relication
    - Vector Clock
        - each (key, value) pair is associated with a vector clock, which records the time of each individual node on this pair.
    - Message
        - Nodes may send messages to individual nodes or node groups. A message consists of a list of (key, value) pairs in the key range R and the associated range vector clock.

        - Because machine learning problems typically require high bandwidth, message compression is desirable.
    - Consistent Hashing: 在parameter server中存数据的结构
    - Server Management: The following steps happen when a server join:
        1. The server manger assigns the new node a key range to serve as master. This may cause another key range to split or be removed from a terminated node.
        2. The node fetches the range of data to maintains as master and k additional ranges to keep as slave
        3. The server manager broadcast the node changes. The recipients of the message may shrink their own data based on key ranges they no longer hold and to resubmit unfinished tasks to the new node.
    - Worker Management

---

*PipeDream: Generalized Pipeline Parallelism for DNN Trainning*

(The first pipeline trainning method for the DNN)

- Background: 之前的方法是intra-batch parallelization, where a single iteration of trainning is split over the available workers,  but suffer from diminishing returns at higher worker counts.
    - PipeDream是inter-batch pipelining → helping to better overlap computation with comunication and reduce the amount of communication when possible.
    - DNN的特点是，这是一个双向的Pipeline，有forward pass和backward pass
    - PipeDream的目的：correct + schedule forward and backward passes of different minibatches concurrently on different workers with mininal pipeline stalls.
- Background 2 (归纳得很好所以就记下来了)：Data parallelism partitions the input data across workers, model parallelism partition operators across workers, and hybrid schemes partition both.
    - PipeDream → a system that uses pipeline parallelism

- 工作原理：PipeDream divides the model among available workers, assigning a group of consecutive(连续的) operators in the operator graph to each of them, and then overlaps the compuation and communication of different inputs in a pipelined fashion. (Moreover, this communication is peer-to-peer, as opposed to all-to-all).
- The scheduling algorithm of PipeDream: 1F1B
  - Used PipeDream uses a scheduling algorithm called 1F1B to keep hardware well utilized while achieving semantics similar to data parallelism.
  - 1F1B also uses different versions of model weights to maintain statistical efficiency comparable to data parallelism
- 为了让pipeline更高效，我们也需要每个stage尽可能take the same amount of time.
  - PipeDream automatically determines how to parttion the operators of the DNN based on a short profiling run perfromed on a single GPU.
  - 因为DNN不一定总是能envenly 分配工作，所以在一定程度上一些stage要采用 data parallelism. —> The combined algorithm: 1F1B-RR
- **Contributions:**
  - Help reduce the communication overheads that can bottleneck intra-batch parallelism
    - PipeDream automatically partitions DNN trainning across workers, combining inter-batch pipelining with intra-batch prallelism to better overlap computation with communication while minimizing the amount of data communicated.
- Background and related work
  - Data parallelism: In data parallelism, inputs are partitioned across workers. Each workers maintains a local copy of the model weights and trains on its own partition of inputs while periodically synchronizing weights with other workers, using either collective communication primitives like all_reudce or parameter servers.
  - Model parallelism: Model parallelism is an intra-batch parallelism approch where the operators in a DNN model are partitioned across the available workers.

- Pipeline parallelism: combines intra-batch parallism with inter-batch parallelsim.
  - Pipeline paralllism的优势
    - Pipeline communicates less: Instead of having to aggregate gradients for all parameters and send the result to all workers, as is done in data-parallel approaches, each workers in a PP execution has to communicate only subsets of the gradients and output activations, to only a single worker.
    - Asynchronous communication of forward activations and backward gradietns across stages results in significant overlap of communication.

- Challenges:
  1. Work Partition:
     a. Solution: Dynamic programming

        a fact: DNN trainning shows little variance in computation time across inputs. （对于其他模型，或者更general的模型，这或许是个值得继续研究的点）。

        PipeDream records the computation time taken by the forward and backward pass. This profiling is used as the input to the optimizer's partitioning algorithm.
  2. Work Scheduling (forward, backward都有所以比较复杂，还有dependency):
     a. Solution: 1F1B-RR
  3. Effective learning
     a. Solution: weight stashing: 其实就是用一个缓存存下来各个版本的w

- Implementation:
  - Pipeline first profiles the model on a single GPU with a subset of inputs from the trainning dataset. Then profiling. The PipeDream runtime then assign each stage to a single worker according to its 1F1B-RR schedule.
  - Stage Replication
  - Checkpointing

*Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD*

Dynamic averaging approach to distributed trainning

→ To design distributed SGD implementations, where gradient computation and aggregation is parallelized across multiple worker nodes. (a criticle need to make distibuted SGD fast, yet robust to system variablity)

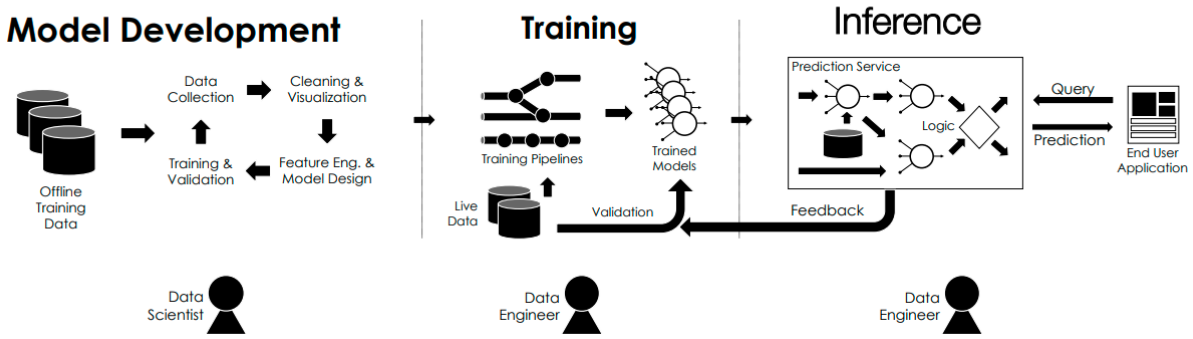
- 如何度量分布式SGD的速度呢？
  - 传统的是用the error in the trained model versus the number of iterations,
  - 但在distributed SGD下还应该考虑the number of iterations completed per second, 传统的并不考虑第二个factor是因为 it is generally a constant when SGD is run on a single dedicated server.
- 针对上面的度量角度，in order to achieve the fastest convergence speed, 我们要的是：
  1. optimization techniques(e.g. variable learning rate) to maximize the error convergencrate with respect to iterations
  2. shceduling techniques(e.g. straggler mitigation, infrequent communication) to maximize the number of iterations completed per second.
     - These two directions are inter-dependent and need to be explored together rather than isolation.
- **A popular distributed SGD implementation is the parameter server framework.**
- Main contribution of this paper:
  - 提出了PASGD（periodic averaging stochastic gradient descent）, where each node perfroms local updates and their models are averaged(什么叫model被平均？) after every t iterations.

---

Week 7: Prediction-serving (ML的另一半部分，模型训练好后开始预测的部分)

Lec:

# Machine Learning Lifecycle




**Machine learning lifecycle**

*Pretzel: Opening the Black Box of Machine Learning Prediction Serving System*

- Optimizing prediction serving pipeline using compiler and database system techniques.
- Abstract:
  - The prediction serving require low latency, high throughput and graceful performance degradation under heavy load.
  - Background: Current prediction serving system consider models as black boxes, whereby **prediction-time-specific optimizations are ignored in facor of ease of deployment.**
  - Contribution: **PRETZEL, a prediction serving system introducing a novel white box architecture enabling both end-to-end and multi-model optimizations.**
- Introduction
  - Two phase when deploy and serve model pipeline:
    - off-line phase: statistics from trainning and state-of-art techniques from in-memory data-intensive systems are used in concert to optimize and compile operators into model plans.

        - on-line phase: memor and CPU resources are pooled amond plans, an event-based scheduling is used to bind computation to execution units.
    - Contributions:
        - 总结了black box的不足
        - design principles
        - System Implementation
        - Evaluation

Black box的缺点们：

1. Memory waste: Containerization of pipelines disallows any sharing of resourses and runtimes between pipelines, therefore only a few models can be deployed per machine.
2. Prediction Initialization: 花了很多时间在initial上,真正的computation非常少
3. Infrequent Accesses: In order to meet milliseconds-level latencies, model pipelines have to reside in main memory. 但这样就会有很大一部分的model在内存外面，如果用到的时候放进来warm-up要花很长的时间。
4. Operate-at-a-time Model
5. Coarse Grained Scheduling: 因为lack visibility into pipelines execution and they do not allow models to properly share computational resources.

- **Design principles for White Box prediction Serving:**
    - White Box prediction Serving：因为透明，方便了很多schedule and allocation stragetry.
    - End-to-end OPtimizations: 减少在runtime时候的decision
    - Multi-model Optimizations：shareable components have to be uniquely stored in memory and reused as mush as possible to achieve optimal memory usage.

  Two phases:

  1. Off-line Phase: 生成pipeline stages，和plan

2. On-line Phase: when an inference request for a registered model plan is received physical stages are parameterized dynamically with the proper values maintained in the Object store. The scheduler is in charge of binding physical stages to shared execution units.
    - Offline: Pre-materialize Predictions — Online: Compute Predictions on the fly

- Big idea: Levearge visibility into pipeline achieved by high-level pipeline DSL to optimize execution across pipelines and stages within a pipeline.

---

# 半期总结：System-for-AI

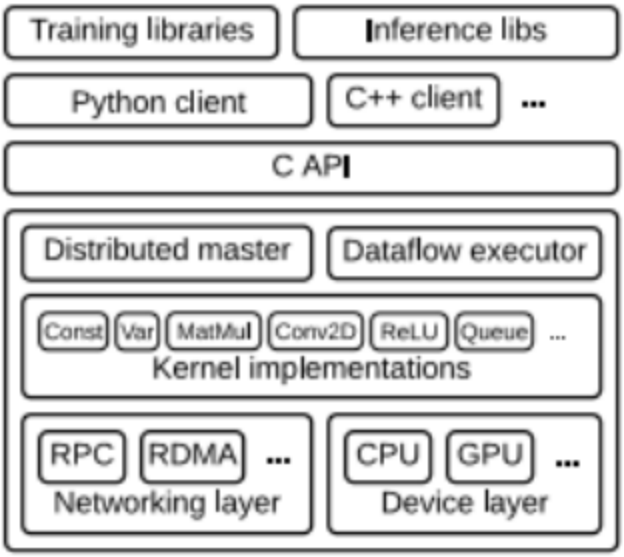
- 主题包括了：
    - **新的社区的诞生**：MLSys: The New Frontier of Machine Learning Systems
        - 诞生原因：ML很热且遇到了在现实生活中应用的瓶颈 + ML不同于传统的软件开发，开发方式不同：shifting models and adjusting parameters instead of coding. 部署方式也不同: 硬件，workflow等等。这些变化要求我们打造全新的系统。
        - 要求：high-level interface for ML development + low-level systems for executing ML models.
    - ML开发中**隐藏的Technique-Debt（一些日后会对性能产生影响的东西）：（这篇我认为是聚焦在High-level development上的）**
        - 模型过于复杂:
            - CACE: Change anything changes everything
                - 解决方法：isolate models and serve ensembles. focus on detecting changes in prediction behavious as they occur
            - Correction Cascades
            - Undeclared  Consumers(没有设置访问模型的权限)
        - Underutilized Data dependency:
            - 导致原因：Legacy Features(一个新加入的feature导致另一个旧的feature冗余) + 一味强调准确率，即使模型已经非常复杂了 + Correlated

Features

- Abstraction Debt...
- The principles of good configuration systems:
  1. It should be easy to specify a configuration as a small change from a previous configurations.
  2. It should be hard to make manual errors, omissions, or oversights.
  3. It should be easy to see, visually, the difference in configuration between two models.
  4. It should be possible to detect unused or redundant settings.
  5. Configurations should undergo a full code review and be checked into a repository.

- 从机器学习的整个**生命周期**出发，来打造一个完整连贯用户友好的平台：TFX：A TensorFlow-Based Production-Scale Machine Learning Platform**（这篇我认为是站在整个Lifecycle的角度上，for better development + better execution的）**
  - TFX的contribution有：
    - One machine learning platform for many learning tasks.
    - Continuous trainning
    - Easy-to-use configuration and tools
      - 特别是在Data Analysis, transformation, validation等等方便用户
    - Production-level reliability and scalability
      - Low latency, high scalability reliability and robustness
- **Towards a Unified Architecture for in-RDBMS Analytics→BISMARCK: 聚焦点为在关系型数据库上进行机器学习**
  - 为什么要这么做呢：数据库在数据结构、数据操作、安全性上有天然的优势（可以回来）
  - Contribution: 提供了一个Unified Architecture方便引入新的模型（不然每次都要develop from scratch）+ 为“bad” order of data提供了策略 + 让BISMARCK并行化计算的策略

- **TensorFlow: A system for large-scale machine learning**
  - 设计原则：primitive operators + deferred execution + common abstraction for heterogeneous accelerators




  - 两个比较重要的组件：
    - Distributed master: translate user requests into execution across a set of tasks.(partition, cache, applies standard optimizations)
    - Dataflow executor: handle requests and schedule execution.
- 分布式学习：
  - 四种parallelism: Model Parallelism, Data Parallelism, Pipeline Parallelism, Operator Level Parallelism
    - Data parallelism partitions the input data across workers, model parallelism partiiton operators across workers.
- **Parameter server system:** *Scaling Distributed Machine Learning with the parameter Server.* （我认为的是一种实现分布式学习的一种架构方式，有点类似shared memory的感觉）
  - Brief Architecture: Both data and workloads are distributed over worker nodes, whithe the server nodes manintain globally shared parameters.

    - 实现过程中的其中两个challenges：
        - Accessing the parameters require an enormous amount of network bandwidth
        - fault tolerance (在云环境下machines can can unreliable and jobs can be preempted)
    - 一些技术细节的名词
        - server node, server manager
        - worker group(一个worker group一个app) worker node, task scheduler.
        - Asynchronous tasks
- Pipeline paralism: **PipeDream: Generalized Pipeline Parallelism for DNN Trainning.**
    - from intra → inter + intra
    - 难点：DNN的pipeline是双向的shcedule forward和backward pass.
    - The scheduling algotihm: 1F1B-RR
    - Pipeline的优势：communicate less不用broadcast, 而只用点对点交流 + overlap communication
    - 我理解的一些技术
        - work partition：在单机上先做一次profilling
        - work scheduling: 1F1B-RR
        - effective learning: weight stashing（存下来各个版本的w）
- **对SGD的parallel:** Adaptive Communication Stategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD.
- **对prediction serving的优化：Pretzel**: Opening the Black Box of Machine Learning Prediction Serving System
    - 让prediction更加透明，这样可以更好的进行Pipeline的优化
    - Two phases:
        - Off-line Phases: 生成Pipeline stages和plan

        - On-line pahses: the scheduler is in charge of binding physical stages to shared execution units.

---

**Week 9: The first lecture about “ML for system”**

*Resource Central: Understanding and predicting workloads for improved Resource Management in Large Cloud Platforms* （感觉这个跟要去面的很相关诶！）

- 关键概念：**Resource Central(RC), a system that collects VM telemetry, learns these behaviors offline, and provides predictions online to various resource managers via a general client-side library. → Resource management 提高**
- Abstact:（这个Abstact写得特别好）:
    - **首先，引入了一些有关VM的characterization(这个感觉是feature的过程)**: first introduce an extensive characterization of Microsoft Azure’s VM workload, including distributions of the VM’s lifetime, deplyment size, and resource consumption.
    - **接着展示了VM的行为在整个生命周期上不会发生很大的变化**(这也是为什么我们可以trainning)
    - **然后，介绍Resource Central (RC)：此处参考上面的关键概念**
    - **最后利用它学习出来的结果来修改VM的scheduler**
- Introduction
    - Motivation:
        - Cloud computing 成为一个很大的热点，研究员有很大的压力去研究good performance, avaiability, and reliability of cloud resource management.
        - 原有的方法缺少对characteristic的描述，offline profiling在VM运行的时候通常unavailable，Online trainning在VM运行的时候很困难.
        - **有清晰，深度的characteristics的话，会让资源管理更加高效**
        - There is a need for software that can produce such predictions and enable the providers’ resource management systems → 所以我们称这个技术为prediction-based resource management.
    - The paper’s work:

        - 介绍了关于Azure's的characterization，包括：distributions of the VM's size, lifetime, resourse consumption, utilization pattern, and deployment size.
        - Our characterization shows that many types of VM behavior are consistent over multiple lifetimes, when observed from the perspective of each cloud customer. VM的行为在整个生命周期变化不会很大
        - 基于以上两点，开发出了RC，并descirbe the modification to Azure's VM scheduler, 它放在client side, 所以即使联网失效也可以available
    - Related work
        - Could VM workload characterization
            - 以往的研究并没有对public provider(third party)的特性描述
        - Predicting cloud workload
            - RC's prediction result derive from a real cloud platform, its workload and machine learning framework.
        - Prediction-based scheduling
            - 以往的工作比较多是online的，但这样it retains contended server resource and may produce traffic bursts.
    - VM characteristic （跳过）
    - Example RC use-cases
        - Smart VM scheduling
        - Smart cluster selection
        - Smart power oversubscription and capping
        - Scheduling server maintenance
        - Recommending CM and deployment sizes
    - Design Principles
        - RC需要independent, 并且off the critical performance and availability paths of the system
        - 在可维护性上，它应该尽量简单并且rely on existing well-supported infrastructures

        - 尽量少修改使用它的系统，并且为用户提供一个简单的接口
    - 这门课使用的模型（重要！！）：
        - Random Forest
        - Extreme Gradient Boosting Tree
        - FFT

---

## Class in Spring(放到在b站上上完network的网课再看吧)

**Lec: AI Application in Network Congestion Control**

---

## 叶片虫害的论文：

*Citrus disease detection and classification using end-to-end anchor-based deep learning model.*

- The detection process involves major steps like:
    - pre-processing
    - feature extraction
    - classification
    - segmentation
- Proposed methods:
    - Data:
        - The data is from Kaggle: 136 Blackspot, 130 canker, 163 greening, 46 health
    - Data prepare:
        - First, all images are converted to grayscale images
            - 因为在preprocessing的过程中发现images have different intensity level.
                - Histogram equalization technique(直方图均匀化) is performed to bring all the images into the same intensity range.
        - Convert the grayscale images to binary images using a thresholding function.

        - 画出Bounding Box
        - Data augumentation such as rotation to oversample the minority classes.
    - Network structure
        - **The network structure consists of four components:**
            - Feature extractor
            - Region Proposal Network(RPN)
            - Region of Interest(ROI) pooling
            - classifier
        - Feature Extractor
            - 提取出来的特征被用在Region Proposal Network和classifier中
            - 特征提取用的式ResNet101(残差神经网络：流入下下层的从F(x)变为了F(x) + x, 减少了梯度衰减)
                - 关于梯度衰减和梯度爆炸的小知识：原因是因为反向传播，这导致求得的导数会以指数增长的速度下降/上升
            - 这个模型用了很多个残差块，each with 3*3的filter with stride = 2。最后接了个平均池化池，以及一个全连接层FC (在tf中是tf.keras.layers.Flatten(), tf.keras.layers.Dense(512, activation='relu'))
                - 关于池化池的小知识：将一些局部的特征聚合
                - 关于全连接层的小知识：是做分类用的，因为聚合了所有提取出来的局部特征。
        - Region proposal network (RPN)
            - 提取出可能包含目标的潜在锚
            - 通过很多个n*n的卷积层，加上两个siblings 1*1 convolutional layers. 然后给objectness 评分: limited the number of proposed anchors to 300 with an itersection over Union of above 0.5
        - ROI (regoin of interest) Pooling:
            - 选出objectness在前50的ROIs成为potential target region
        - classifier

        - potential target region被输入到classifier里，然后用从feature extractor中提出的features来进行分类。
    - 学习过程
        - 采用了迁移学习
            - weights are transferred from a pre-trained model (the pretrained model on the COCO dataset)
        - 整个学习分为了两个阶段
            - 第一个阶段：只有feature extractor被训练几个周期，其他保持frozen，这样训练35个epochs，
            - 第二个阶段：trained together，370个epochs
    - Model’s loss function:
        - define separated losses and combine them to generate the model’s final score
        - 用RPN和Classifier的loss来加权(一般用的是交叉熵cross-entropy)

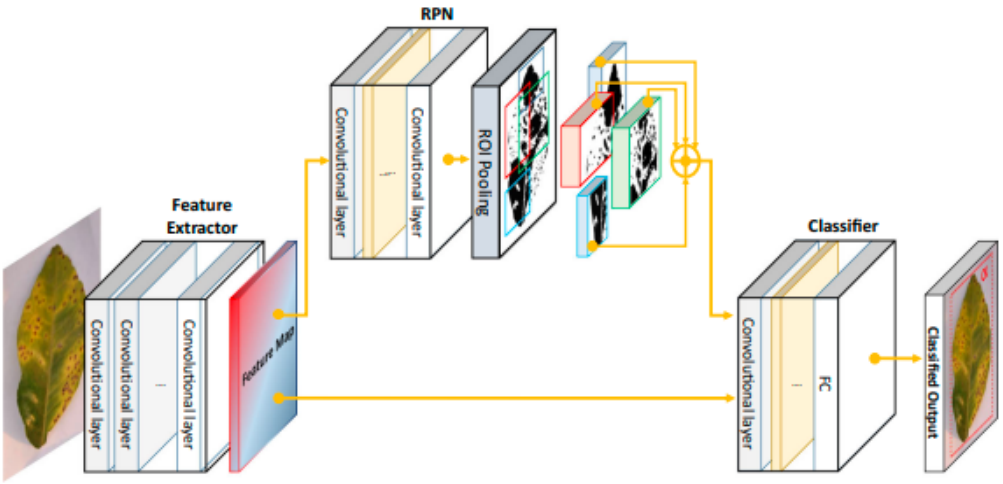



**Fig. 3** Block diagram of the proposed model structure

- Validation
    - 5-fold cross-validation strategy

### *2.2 K-fold Cross Validation*

另外一种折中的办法叫做K折交叉验证，和LOOCV的不同在于，我们每次的测试集将不再只包含一个数据，而是多个，具体数目将根据K的选取决定。比如，如果K=5，那么我们利用五折交叉验证的步骤就是：

1.将所有数据集分成5份

2.不重复地每次取其中一份做测试集，用其他四份做训练集训练模型，之后计算该模型在测试集上的 $MSE_i$

3.将5次的 $MSE_i$ 取平均得到最后的MSE

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i.$$

- 这个模型能outperform others的原因很大一部分在于RPN，因为这样classifer 可以更专注于manageable number of target like candiates.

---

## Apply AI to network

***Pensieve Neural Adaptive Video Streaming with Pensieve***

- 现有的：ABR算法（自适性算法）based on simplified or inaccurate models. 缺点：太fixed了
- **Pensieve：a system that generates ABR algorithms using reinforment learning (RL)**
  - 训练是基于从用户数据播放器（video player）采集来的数据 — 是仿真，这里不能模拟真实的场景缺点
- 要解决的问题：HTTP-based video streaming traffic.

Introduction:

- 现有的ABR算法是基于估计的网络吞吐量，和buffer size
  - Challenges:
    - fluctuate network condition

        - 如果要提升用户体验的话，需要balance各种各样的东西: video quality(highest average bitrate), minimizing rebuffering events, video quality smoothness(avoid constant bitrate fluctuations)
- Pensieve maps “raw” observations(吞吐量，playback buffer occupancy，video chunk sizes) to the bitrate decision.
- ABR server: which video clients query to **get the bitrate to use for the next chunk**.

---

## OpenNetLab

OpenNetLab开放网络平台通过构建分布式节点收集不同区域、形态、场景的网络数据。现阶段，微软亚洲研究院计划将与合作高校一起在亚洲范围内建设40多个分布式异构节点。每个节点将由服务器、笔记本电脑、智能设备等组件构成，同时提供有线宽带、无线局域网和4G/5G移动网络的接入能力，数据存储、集成、分享的标准化接口，以及适用于不同网络环境的AI模型运行、训练和验证工具，帮助研究人员专注于网络AI算法和模型。今后，这些节点将在平台用户同意的情况下，实时收集网络状态、数据包跟踪等非隐私/非敏感数据，为各类网络AI模型的训练及验证提供支持。

- 网络研究生态系统
- 实地训练和验证网络模型

讲座：

**Real-time Communication:**

- 很多和网络模块有关系：视频传输的帧率，速率...非常非常复杂
- chanllenges：
    - 全场景的测试（不同的网络环境）
    - 用户行为不一样，不是同一个协议，云商给网络模块的创新有很大的
    - complex network bottleneck
- Needs:
    - needs heterogenous devices
    - Needs real applications
    - Needs distributed nodes&heterogenous network

- Hightlight - Data centric for networking - related AI 可以在上面找数据集训练，也可以用上面的网络环境来测试

- OpenNetlab can provide TCP congestion benchmarks between nodes across the world after the end of support of pantheon
- R3NET RTC软件中 UKF（波值估计算法）的算法 —> 不够，需要启发式算法
- 问题：
  - 可以同边缘计算相结合吗？ —> 计划给这些结点增加machine learning的能力，来提供边缘计算的能力，增加边缘计算场景
  - 帮助从网络多场景中收集数据
  - 是否提供经典的网络算法？ —> 经典的拥塞控制？
  -

基本知识

做事方式 —>

兴趣

沟通能力

confident 事实求是