Designing Machine Learning Systems Reading Notes

Some other blogs by the author: https://huyenchip.com/blog/

Chapter 1: Overview of Machine learning Systems

- ML algorithm is just a small part of an ML system in production.

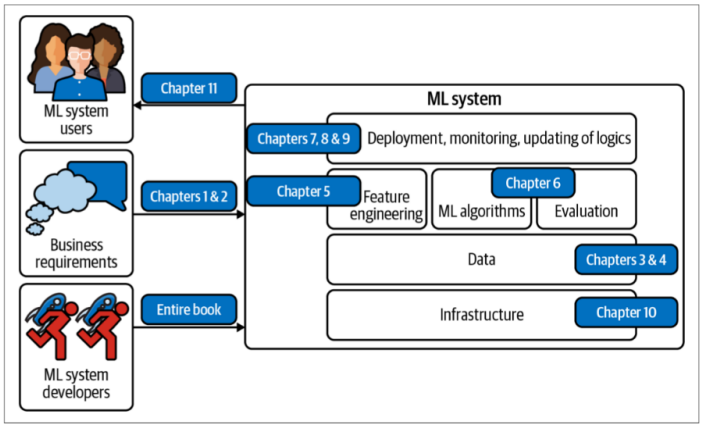


Figure 1-1. Different components of an ML system. "ML algorithms" is usually what people think of when they say machine learning, but it's only a small part of the entire system.

- When to use ML:
  - ML is an approach to (1) learn (2) complex patterns from (3) existing data and use these patterns to make (4) predictions on (5) unseen data.

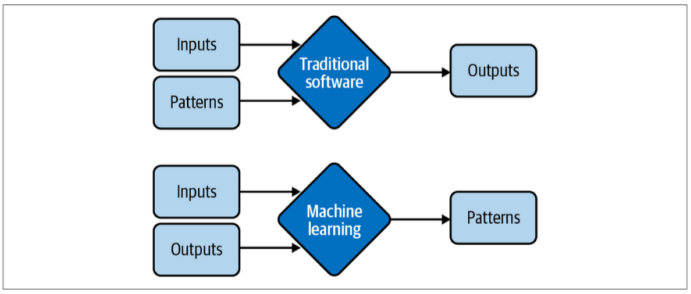


Figure 1-2. Instead of requiring hand-specified patterns to calculate outputs, ML solutions learn patterns from inputs and outputs

- ML in research vs in production
  - Requirements:
    - In research people only care about model performance and achieve SOA result on benchmark datasets.

        - In production, different stakeholders are involved and have different(often conflicting) requirements. e.g. for app to recommend restaurants:
            - ML engineer: build a recommendation system, want more complex model and more data to achieve better model performance.
            - Sales team: want a model to recommend more expensive restaurants to user since they bring in more service fees.
            - Product team: increase in latency leads to drop in orders through the service, so want a model can return result in less than 100ms.
            - ML platform team: don't want too many model updates and prioritize on platform improving.
            - Manager: maximize the margin and cut cost.
        - hard to optimize single metric in ML model to meet real business requirements:
            - return restaurant that user are most likely to click on?
            - return restaurant that will have most service fee?
        - need to understand all those different requirements and how strict they are, then make trade-off.
            - within 100ms might be a hard requirements.
            - for some systems, small improvement in model performance can result in huge boost in revenue, but for some other systems, it might be not noticeable to users.
        - e.g. In many ML competitions like Kaggle, one of the most popular technique is ensemble learning. but it's too complex and expensive to use in prod.
- Computational priorities:
    - model development vs model deployment and maintenance
    - throughput is more important for training, latency is more important for inference. with modern distributed systems, we can do batch queries, relationship between latency and throughput becomes more complex. larger batch size can improve throughput, but may increase latency.
    - In production, we also need to take much more care to the high percentile outliers. e.g. customers with the slowest requests are often those who have the most data on their accounts since they have made many purchases before, and are the most valuable customers.
- Data:
    - research usually just use public dataset which is already cleaned and well formatted.
    - in production, need to use data collected from users, which is noisy, unstructured, constantly shifting, and most likely biased.
- Fairness:
    - research usually just care about SOA accuracy, but not much on fairness metrics.
    - in production, small bias embodied in model can hurt millions of people in a second, and especially on minority groups.
- Interpretability

Table 1-1. Key differences between ML in research and ML in production

	Research	Production
Requirements	State-of-the-art model performance on benchmark datasets	Different stakeholders have different requirements
Computational priority	Fast training, high throughput	Fast inference, low latency
Data	Static[a]	Constantly shifting
Fairness	Often not a focus	Must be considered
Interpretability	Often not a focus	Must be considered

[a] A subfield of research focuses on continual learning: developing models to work with changing data distributions. We'll cover continual learning in Chapter 9.

- ML system vs Traditional software
    - In traditional systems, code and data are separate, and we want to keep things as modular and separate as possible. Usually we only need to test/debug the code.
    - ML systems are part code and part data and part artifacts created from these two. need to test/debug both code and the data used.

Chapter 2: Introduction to Machine Learning Systems Design

BUSINESS AND ML OBJECTIVES

- Business objectives: The ultimate goal of any project in most companies is to increase profits, either directly or indirectly. Directly: increase conversion rate, cut cost; indirectly: increase customer satisfaction, increase time spent on app.
- ML objectives: model accuracy.

It's important to map the ML metrics to your business metrics to make it meaningful.

- e.g. For a bank customer service support department, real business goal is to address the slow customer support issue. but it's not an ML problem.
    - after investigation, discover the bottleneck here is to routing requests to the right place among all the departments in the bank.
    - then can turn it into a classification problem to get input from user request, and predict the department it should go to.

REQUIREMENTS FOR ML SYSTEMS

- Reliability
    - System should continue to perform the correct function at the desired level under adversity.
    - "correctness" might be hard to determine for ML systems. How do you tell an Ads recommendation is "correct" or "wrong"? Unexpected recommendation may result in higher conversion rate.
    - In traditional systems, if something goes wrong we will usually get a warning or runtime error. But for ML systems, usually it will just fail silently. Even each individual user is not aware of it.
- Scalability
    - There are multiple ways an ML system can grow:
        - Model complexity/size

        - Traffic volume.
        - Number of models
    - Handling growth isn't just resource scaling, but also model management, monitoring/retraining, ...
- Maintainability
- Adaptability
    - adapt to the shifting data distribution and business requirements.

ITERATIVE PROCESS

Developing ML system is an iterative, never-ending process.

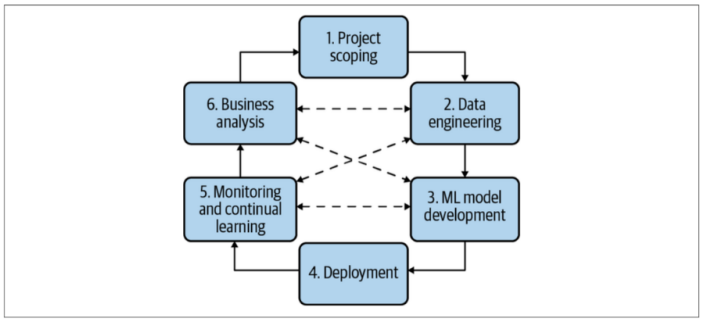


Figure 2-2. The process of developing an ML system looks more like a cycle with a lot of back and forth between steps

TYPES OF ML TASKS

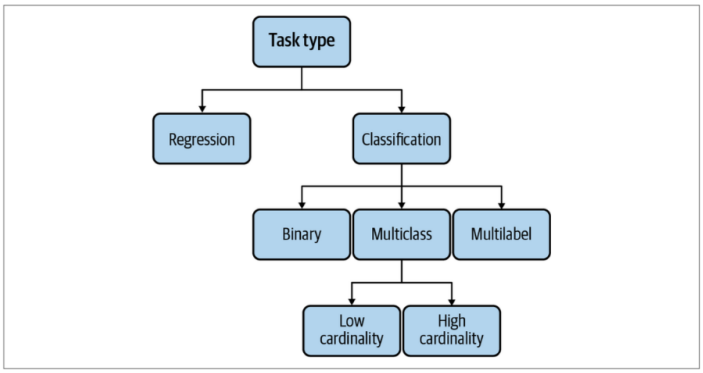


Figure 2-3. Common task types in ML

- Regression vs Classification
    - Regression and classification can be used to solve the same problem:

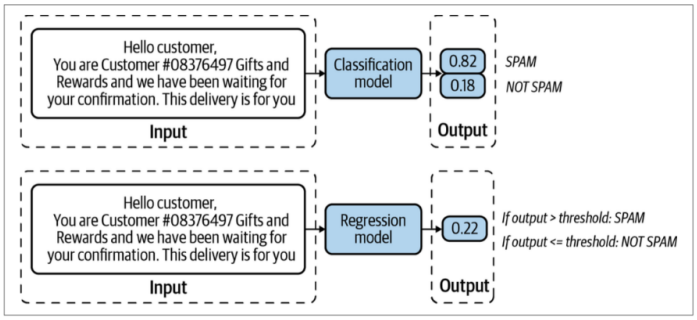


Figure 2-4. The email classification task can also be framed as a regression task

    - main difference between classification and regression is on the label used for training.
- binary vs multi-class classification
    - if number of class is large, may consider hierarchical classification.
- multi-label classification
    - One record can belong to multiple classes. e.g. classify an article into: tech, entertainment, finance, politics.
        - One way is to use a multi-class classifier, so the output of the model for each record is a vector of probabilities corresponding to each category: [0.45, 0.2, 0.02, 0.33], and the label is also a vector which can contains multiple classes: [1, 0, 0, 0], [0, 1, 1, 0], ...
            - cannot just pick the highest topic since now we may have multiple categories and we don't know how many it is. may set a threshold for a valid output
        - The other way is to have four binary classifiers to classify on each topics.
- multiple ways to frame a problem
    - for app recommendation:
        - can use multi-class classification:

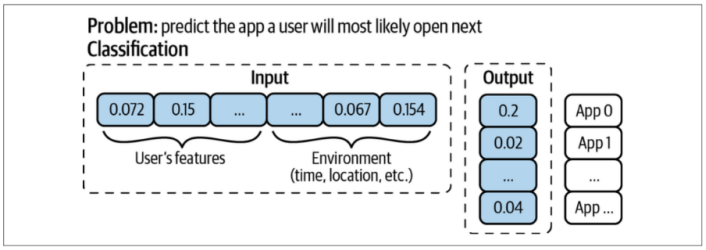


Figure 2-5. Given the problem of predicting the app a user will most likely open next, you can frame it as a classification problem. The input is the user's features and environment's features. The output is a distribution over all apps on the phone.

            - but every time there is a new app added. the dimension of model will be changed, and need to retrain.
        - So a better approach would be build a regression model to rate an app for a given user, and then rank the scores for all apps.

OBJECTIVE FUNCTIONS(LOSS FUNCTION)

- Regression: average absolute error
- Classification: cross entropy. for binary case, it's also called logistic loss.

[TODO] a blog to explain on different loss functions: https://towardsdatascience.com/understanding-sigmoid-logistic-softmax-functions-and-cross-entropy-loss-log-loss-dbbbe0a17efb#179d.

Decoupling objective
Problem in real world usually need multiple steps/components, could be tricky to optimize for multiple objectives. so better try to decouple them

- e.g. for recommendation systems, purely maximizing user engagements may lead to ethical issues like bias on prefer to recommend extreme posts. So also need another objective to minimize spread of extreme views and misinformation.
- these two objectives: maximize engagement and minimize extreme post are conflict to some extent.
- so we can express the total loss as: loss= a * quality_loss + b * engagement_loss, and tune the value of a and b to adjust the trade-off
- if have a big single model for this, will need to retrain the model every time tune the value of a,b
- so better to have two separate model to predict post quality and post engagement, and then combine together to get total loss.

MIND VS DATA

ML algorithm vs data
...

Chapter 3: Data Engineering Fundamentals

DATA SOURCES

- First-party data: data collected by your own company on your own customers
- Second-party data: data collected by another company on their own customers
- Third-party data: data collected by another company from public who aren't their customers

Smartphone has unique advertiser ID:

- IDFA(Apple's identifier for advertiser), AAID(Android advertising ID)
- data from all apps,websites,services on each phone can be collected to generate history for each advertiser ID.
- new privacy policy has reduced significantly the amount of third-party data available, so force many companies to focus more on first-party data.

DATA FORMATS

Format	Binary/Text	Human-readable	Example use cases
JSON	Text	Yes	Everywhere
CSV	*Text*	*Yes*	*Everywhere*
Parquet	*Binary*	*No*	*Hadoop, Amazon Redshift*
Avro	Binary primary	No	Hadoop
Protobuf	Binary primary	No	Google, TensorFlow (TFRecord)
Pickle	Binary	No	Python, PyTorch serialization

- JSON: ...
- Row-major vs column major
    - column-major is more flexible and efficient for read
    - row-major is more efficient for write
    - pandas DataFrame is built around the columnar format, ndarray in Numpy uses row-major format
- Text vs binary format: ...

DATA MODEL

- Relational model:
    - usually require to normalize the data to spread information across multiple relations: star schema
    - use declarative languages instead of imperative, so it's up to the database to how to execute it in backend and can leverage to query optimizer to optimize the physical plan.
- NoSQL:
    - document model: ...
    - graph model: ...

Structured vs unstructures:

Table 3-5. The key differences between structured and unstructured data

Structured data	Unstructured data
Schema clearly defined	Data doesn't have to follow a schema
Easy to search and analyze	Fast arrival
Can only handle data with a specific schema	Can handle data from any source
Schema changes will cause a lot of troubles	No need to worry about schema changes (yet), as the worry is shifted to the downstream applications that use this data
Stored in data warehouses	Stored in data lakes

DATA STORAGE ENGINE AND PROCESSING

- OLTP:
    - write heavy, often row-major
- OLAP:
    - read heavy, often column-major

The term "OLAP.OLTP" have become outdated and the separation is being closed.
For traditional OLAP/OLTP, the storage and processing are tightly coupled, which results in the same data being stored in multiple dbs. Whereas the recent trend is to decouple the storage from the processing system.

MODES OF DATAFLOW

- through DB
- through services: request-driven
- through real-time transport/msg broker: event-driven, works better for data-heavy systems

BATCH PROCESSING VS STREAM PROCESSING

...

Chapter 4: Training Data

SAMPLING

- Nonprobability sampling: ...
- Probability based sampling:
    - simple random sampling
    - stratified sampling: divide into groups and random sample from each group
    - weighted sampling
    - Reservoir sampling: random sampling k elements from unbounded streaming data, ensure each sample has same probability to be selected.
        - put first k elements into reservoir
        - for each incoming nth element, generate a random number i between 1~n
        - if 1<= i <= k, replace the ith element in reservoir, else skip.
        - nth element will has k/n probability of being in the reservoir

LABELING

- Hand label: slow, hard to track quality
- Natural label: e.g. ad click, stock price
    - can have feedback loop based on user actions, need to carefully choose the right window size for the loop, short window can capture labels faster but may prematurely label a recommendation as bad before it's clicked on.

Handling the lack of labels

- weak supervision
    - use heuristics to generate labels. e.g. use pattern match or other labeling functions(LF) created by domain experts
    - one heuristic may be not accurate so usually can combine multiple rules, denoise and reweight them to get final generated label
    - e.g. X-ray img classification task. spending few hours of writing LFs to generate labels for weak supervised learning which takes one year of hand labeling for fully supervised learning:

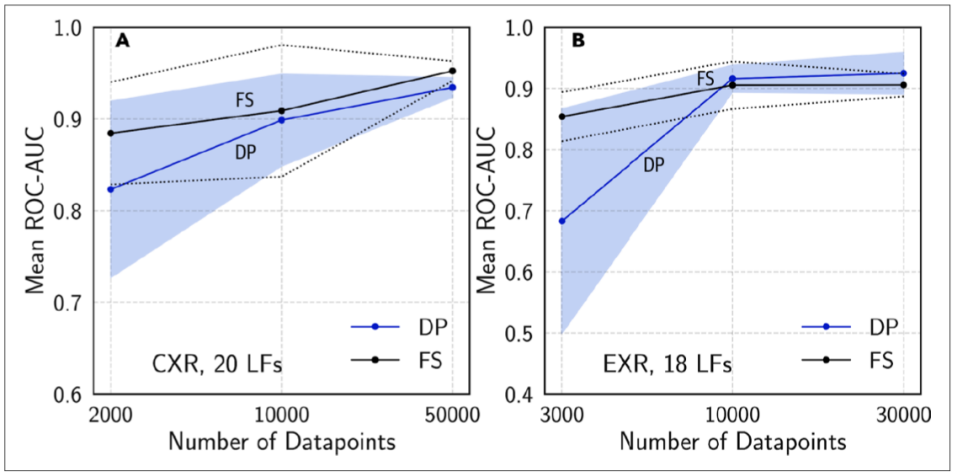


Figure 4-5. Comparison of the performance of a model trained on fully supervised labels (FS) and a model trained with programmatic labels (DP) on CXR and EXR tasks. Source: Dunnmon et al.[15]

- Question you may ask: if heuristics work well to predict labels, why do we need an ML model for this?
  - usually LF still cannot beat ML models on performance, weak supervised learning is just a way to utilize those un-labeled data as a complementary step to further boost the performance of the supervised learning model.
  - even for fully weak supervised learning, usually LFs doesn't cover all data samples so cannot generalize well, therefore we need to train an ML model based on it to better generalize to all cases.

- semi-supervision
  - leverage structural assumptions to generate new labels based on a small set of initial labels.
    - one classic semi supervision method is self-training: training a model on a small set of labeled data, then use it to generate label for more unlabeled data, and train again, use updated model to generate labels, train again, ...
    - another method assumes that data samples that share similar characteristics share the same labels. e.g. contain the same key words, or compare similarity with clustering method.
    - perturbation-based method: assumes introducing a small perturbation to the training input doesn't change its label. e.e.g add white noise to img, small random values to embeddings, ...
- transfer learning
  - reuse a pretrained model as the starting point. e.g. zero/few-shot learning
- active learning
  - instead of randomly labeling all data, can choose samples that are most helpful to the model accuracy based on some metrics or heuristics. e.g. samples that the model output has low confidence, or has disagreement among multiple models.

Table 4-2. Summaries of four techniques for handling the lack of hand-labeled data

Method	How	Ground truths required?
Weak supervision	Leverages (often noisy) heuristics to generate labels	No, but a small number of labels are recommended to guide the development of heuristics
Semi-supervision	Leverages structural assumptions to generate labels	Yes, a small number of initial labels as seeds to generate more labels
Transfer learning	Leverages models pretrained on another task for your new task	No for zero-shot learning Yes for fine-tuning, though the number of ground truths required is often much smaller than what would be needed if you train the model from scratch
Active learning	Labels data samples that are most useful to your model	Yes

CLASS IMBALANCE

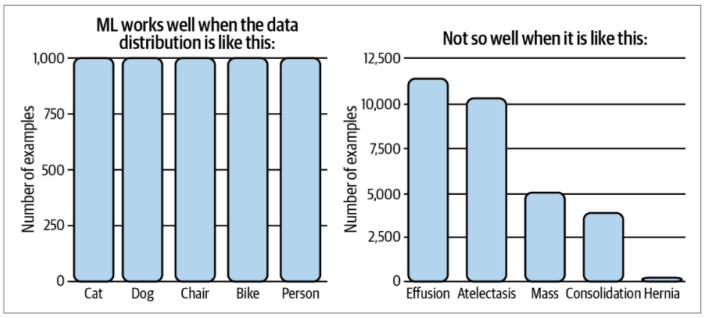


Figure 4-8. ML works well in situations where the classes are balanced. Source: Adapted from an image by Andrew Ng[26]

- insufficient signal for the model to learn about the minority classes and becomes a few-shot learning problem
- easy to stuck in nonoptimal solution by exploiting simple heuristic instead learning the underlying pattern
- usually the rare events are often more interesting/dangerous than regular events. e.g. ad click/fraud content are rare events but that's what we actually care the most.

Handling class imbalance

- larger and deeper network can performs well on imbalanced data, since they are complex enough to learn the imbalance pattern. but this could be challenging.
- other approaches:
  - choose the right metrics. e.g. for binary classification:
    - *Table 4-6. Definitions of True Positive, False Positive, False Negative, and True Negative in a binary classification task*

	Predicted Positive	Predicted Negative
Positive label	True Positive (hit)	False Negative (type II error, miss)
Negative label	False Positive (type I error, false alarm)	True Negative (correct rejection)

$$\text{Precision} = \text{True Positive} / (\text{True Positive} + \text{False Positive})$$

$$\text{Recall} = \text{True Positive} / (\text{True Positive} + \text{False Negative})$$

$$F1 = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$$

    - output of binary classification is a probability. usually we can consider value > 0.5 as true and else false. but it's tunable. larger threshold value can increase true positive rate(recall), but will reduce false positive rate. can plot true positive rate against false positive rate with different threshold values, know as ROC

curve(receiver operating characteristics).

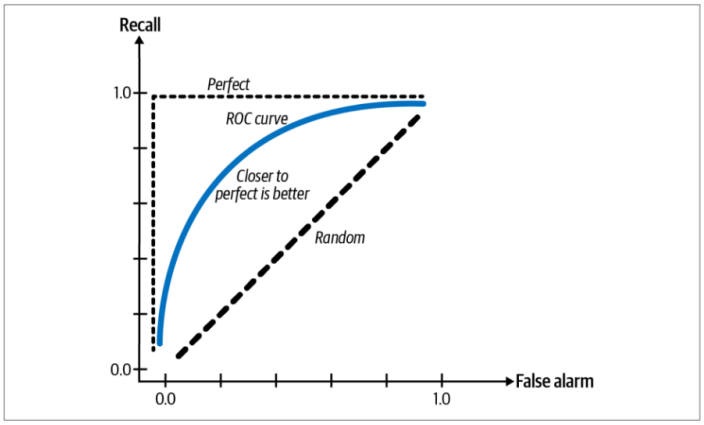
- area under the ROC curve(AUC). for perfect model AUC should have max value(1)
- 

Figure 4-9. ROC curve

- data-level method: resampling
  - have risk to overfit to resampled distribution
  - don't evaluate model on resampled data
  - also can have two-phase learning: train on resampled data, then fine tune on original data
- algorithm-level methods:
  - cost-sensitive learning: can have a cost matrix to define what's the loss weight for each different output classes(which class is classified as which class)
    - *Table 4-8. Example of a cost matrix*

	Actual NEGATIVE	Actual POSITIVE
Predicted NEGATIVE	$C(0, 0) = C_{00}$	$C(1, 0) = C_{10}$
Predicted POSITIVE	$C(0, 1) = C_{01}$	$C(1, 1) = C_{11}$

  - class-balanced loss: give more weights for loss function on samples in the minority group
  - ensemble can also help

DATA AUGMENTATION

family of techniques to increase the amount of training data when there are only limited data available. also can make model more robust to noise and even adversarial attacks

- simple label-preserving transformation: e.g. rotate/crop the img
- perturbation: add noise randomly or by a search strategy(adversarial augmentation)
- data synthesis: e.g. use a template to generate massive sentences: "Find me a [CUISINE] restaurant within [NUMBER] miles of [LOCATION]". or more advanced method like using a generative model.

Chapter 5: Feature Engineering

LEARNED FEATURE VS ENGINEERED FEATURE

the promise of deep learning is that we don't need to handcraft features, and the model should be able to learn the right way

to extract features from raw input. But currently we are still far from this point.

COMMON FEATURE ENGINEERING OPERATIONS

- handling missing values
  - three types on missing values:
    - missing not at random
      - value missing related to the value itself. e.g. people with higher income tend to not disclose their income
    - missing at random
      - value missing due to another observed variable. e.g. people of certain gender tend to not disclose their age.
    - missing completely at random
      - no pattern
  - ways to handle:
    - deletion
      - column deletion: drop the feature.
      - row deletion: drop individual sample(only works when it's missing completely at random), may create bias in the model
    - imputation: fill in with certain value
      - should avoid fill in some actual possible values
      - may injecting your own bias and add noise
- scaling
  - different features may have different value range, better to scale them to a similar range
  - also feature value may have a skewed distribution, can us log transformation to mitigate.
- encoding categorical features
  - usually need to define an id/label for each category, like ad_id, page_type.
  - can generate a hashed value for each category as the index. also can specify the hash space to fix the number of encoded values.
    - can choose large space to reduce the collision rate or use a locality-sensitive hashing function so collision samples are also very similar.
    - in general, experiments shows hash collision doesn't hurt the model performance too much.
- feature crossing
  - capture nonlinear relationship between variables
  - may blow up the feature space. e.g. both feature A and B have 100 possible values, A*B will have 10000possible values. so will need more data to learn all those possible values, otherwise may overfitting
- discrete and continuous positional embedding
  - mainly used for transformer models. embedding value can be learnt from training or predefined a fixed value.

DATA LEAKAGE

phenomenon when a form of the label "leaks" into the set of features used for making predictions, but this same information is not available during inference for future data.

- e.g. train a model to detect cancer from chest x-ray img. the training data contains x-ray img when people were lying down and standing up. since people scanned while lying down were more likely to be serious ill, so the model learned to predict based from people's position instead of the actual infection component.
- e.g. model picking up the text font that certain hospital used to label the scans

common causes for data leakage

- splitting training and validation data set for time-correlated data randomly instead of by time
    - e.g. for model predicting stock price. similar stocks tend to change price together, if most tech stocks goes down on a day, large likely the rest of tech stock will also go down. so the right way is to use first 6 days data to train and eval on 7th day's data, otherwise you're leaking future information into the model by including prices from the seventh day in the training data
    - to prevent future information from leaking into the training process and allowing model to cheat during eval, need to split the data by time instead randomly
- scaling/filling missing data with global statistics before splitting
    - scaling/filling missing data sometimes need some global statistics of the data(e.g. avg, variance,...), so if you scale before split the training and eval set, will leak those global statistics information to the model.
- have duplicated sample in training and eval set
    - e.g. 3.3% and 10% of the imgs from the test set of the CIFAR-10 and CIFAR-100 public dataset have duplicates in training set...
- group leakage
    - a group of strongly correlated sampled are divided into different splits. e.g. X-Ray img from same person taken at different time

ENGINEERING GOOD FEATURES

adding more features doesn't always mean better model perf:

- will increase the risk of data leakage
- may cause overfitting
- can increase memory usage
- increase inference latency

two factors to consider when evaluating a feature

- feature importance
- feature generalization
    - feature coverage
    - feature value distribution

Chapter 6: Model Development and Offline Evaluation

MODEL DEVELOPMENT AND TRAINING

Model selection: ...

Ensembles:

- bagging(bootstrap aggregation): instead of training one model on entire dataset, sample the data to create multiple different datasets, and train a model on each of them.
    - improve training stability and accuracy
    - reduce variance and helps to avoid overfitting
- boosting:
    - a family of iterative algorithms that convert weak learner to strong one. Each learner is trained on same dataset, but samples are weighted differently so next version focus more on the examples that previous version misclassified. Final strong classifier is a weighted combination of all versions.
    - e.g. GBDT(gradient boosting decision tree)

Experiment tracking and versioning: also need to do data versioning besides code version, ...

Distributed training:

- Gradient checkpointing: method used on a single machine for reducing the memory footprint when training deep neural networks, at the cost of having a small increase in computation time: https://arxiv.org/abs/1604.06174v2
- Data parallelism: split data on multiple machines, train models on all of them and accumulate gradients. Key problem is how to accurately and effectively accumulate gradients from different machines.
    - synchronous SGD(stochastic gradient descent)
        - workers need to wait for each other
    - asynchronous SGD
        - gradient updates on different machines may have consistency issues
        - in theory can converge in the end but requires more steps than synchronous.
        - in practice, gradient updates tend to be sparse for large networks, so each gradient update only modify small fractions and less likely have conflict. therefore can have similar performance as synchronous SGD.

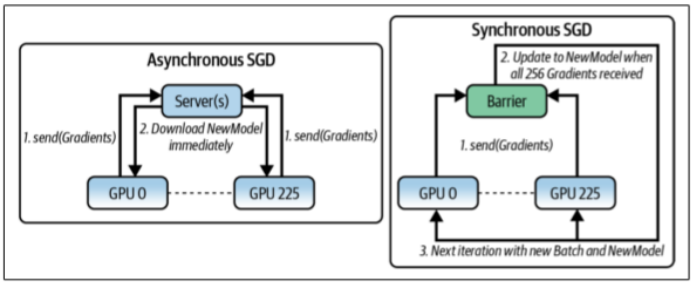


Figure 6-6. Synchronous SGD versus asynchronous SGD for data parallelism. Source: Adapted from an image by Jim Dowling[17]

    - another issue for data parallelism is we're effectively training with a super large batch size(1000 machine with batch size 100 => effective batch size 1M), then will have much less training step(gradient iteration times) so may not enough to reach the optimal point. can tune up the learning rate to update gradient at larger scale for each step, but may cause diminishing return issue.
- Model parallelism
    - different components of your model are trained on different machines
    - there might be dependencies between components so different machines may not purely run in parallel. can use pipeline parallelism by breaking each input data into multiple mini-batch:
    -

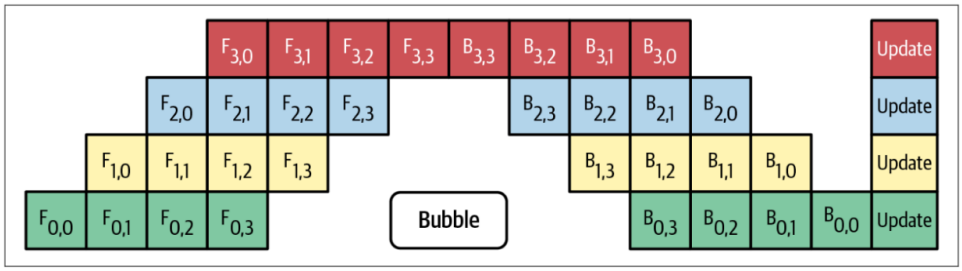


Figure 6-8. Pipeline parallelism for a neural network on four machines; each machine runs both the forward pass (F) and the backward pass (B) for one component of the neural network. Source: Adapted from an image by Huang et al.[22]

- model parallelism and data parallelism aren't mutually exclusive and can be used together depending on the design.

AutoML:
Use more compute resources to replace ML expertise.

- soft AutoML: hyperparameter tuning
- hard AutoML: architecture search and learner optimizer

MODEL OFFLINE EVALUATION

Baseline: random baseline, simple heuristic, ...
Evaluation methods: metrics other than accuracy

- perturbation test
- invariance test
- calibration: match the predicted result distribution to the real event distribution. e.g. for movie recommendation, if the user watches romance movies 80% of the time and comedy 20% of the time, then the recommended movie should consist of 80% of romance and 20% comedy instead of 100% romance even it's the more preferred topic.

Chapter 7: Model Deployment and Prediction Service

Model compression

- low-rank factorization: replace high dimensional tensors with lower dimensional tensors.
  - e.g. compact convolution filters:
    - ■

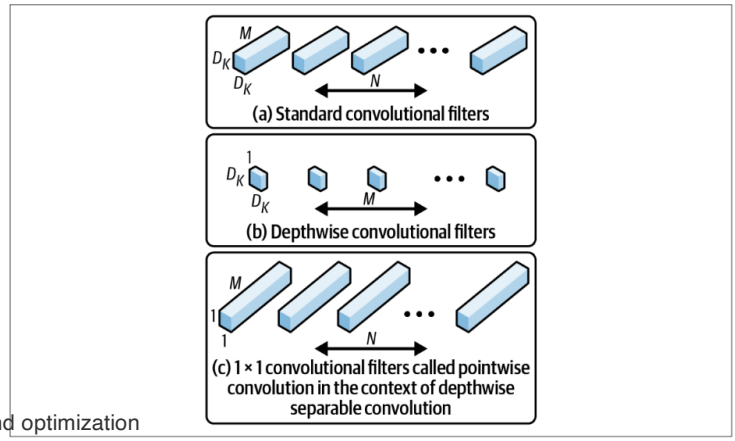


Figure 7-9. Compact convolutional filters in MobileNets. The standard convolutional filters in (a) are replaced by depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter. Source: Adapted from an image by Howard et al.

Model compiling and optimization

- different backend have their own memory layout and low level implementation of primitive operations. so can create middleware(IR) to bridge and decouple high level ML framework and the low level backends:
  -
- knowle
- pruning
- quantiz
  - u
    r
  - c

ıe industry,

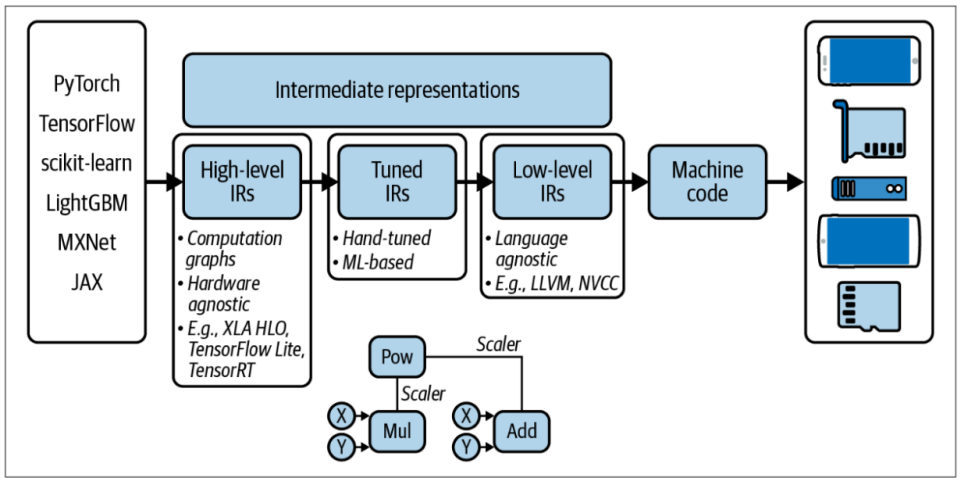


Figure 7-12. A series of high- and low-level IRs between the original model code to machine code that can run on a given hardware backend

- also have opportunities to optimized the compute graph on IR, both locally and gloablly.
  - local optimization:
    - vectorization
    - parallelization
    - loop tiling: change the loop order to leverage hardware's memory layout and cache.
    - operator fusion: fuse multiple operators into onw to avoid redundant memory access
      - e.g. for simple for loop operator:
        -

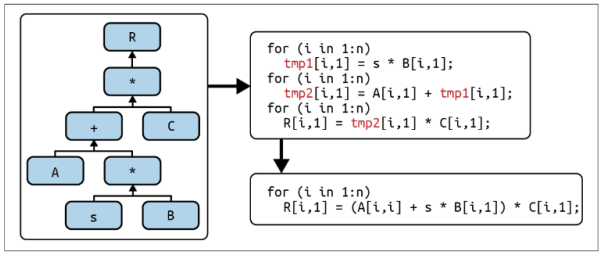


Figure 7-13. An example of an operator fusion. Source: Adapted from an image by Matthias Boehm[47]

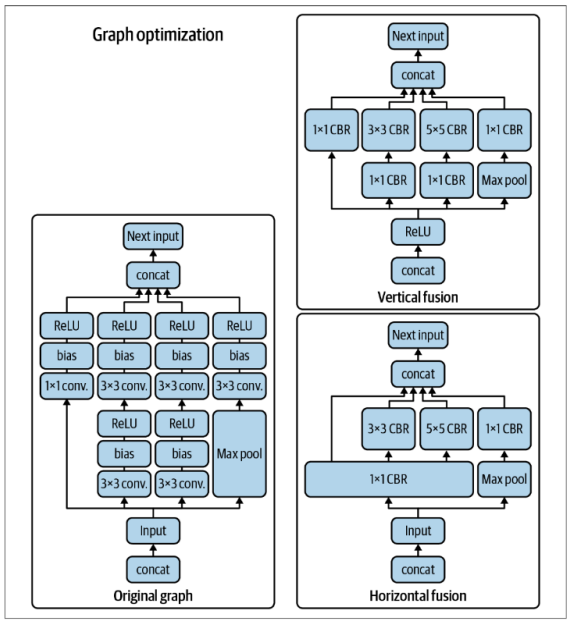
- e.g. for neural network:
  - 

Figure 7-14. Vertical and horizontal fusion of the computation graph of a convolution neural network. Source: Adapted from an image by TensorRT team[68]

Chapter 8: Data Distribution Shifts and Monitoring

Deploying a model isn't the end of the process. also need to keep monitoring its performance to detect any regressions as well as deploy updates to fix the issue'.

- Software specific failures: ...
- ML specific failures:
  - production data differing from training data, so model performs well in training but bad in inference
  - real world isn't stationary and keeps drifting suddenly, gradually or seasonally.
  - extreme edge cases cause catastrophic consequences, e.g. self-driving car
  - degenerate feedback loops: prediction itself can influence the feedback, and therefore influence next iteration of the model.
    - e.g. for song recommendation, user tends to click on the first song in the recommended list, so the feedback will make system to rank the first song even higher.
    - e.g. model to select candidate from resumes. only the selected candidates will be hired in the end, so the bias in the model will be amplified.
    - so need to monitor the output popularity diversity, average coverage of long-tail items, ... to detect degenerate feedback loop
    - to address:

- use randomization
- use positional features

Monitoring and observability

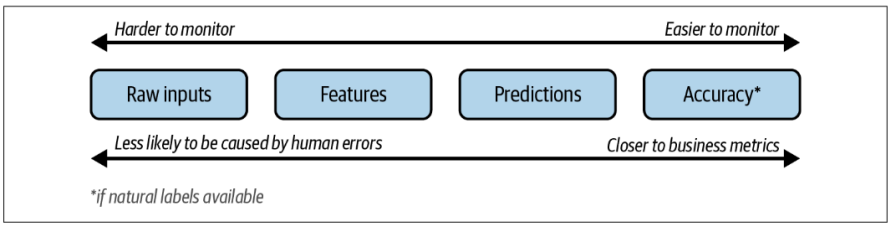


Figure 8-5. The more transformations an artifact has gone through, the more likely its changes are to be caused by errors in one of those transformations

Chapter 9: Continual Learning and Test in Production

CONTINUAL LEARNING

continuously update model to adapt to data drifting.

- stateless retraining: from scratch
- stateful retraining: incremental fine tune with new data

challenges:

- fresh data access challenge: need a realtime infra, also need to join events to get label
- evaluation challenge: continual learning makes the model more susceptible to coordinated manipulation or adversarial attack.

TEST IN PRODUCTION

- A/B test, random route traffic to each model
- Interleaving experiments, expose a user to both models' recommendation and see which one user will click on.
- Bandits:
    - multi-arm bandit algorithm: balance between exploration and exploitation
        - e.g. multiple slot machines in casino, want to find which one gives the highest payout(exploration) while maximizing the total payout(exploitation). so a simplest bandit algorithm here is to choose the current max machine 80% of the time and explore new machines 20% of the time.
        - well studied in academia, bandits require much less data to determine which model is the best compared with purely random A/B test, and also reduce the opportunity cost during the experiment.
        - much harder to implement, need to compute and keep track of models' payoff, therefore are not widely used in the industry.
    - contextual bandits: can also use bandit for the recommendation from one model
        - e.g. for video recommendation, cannot recommend user's favorite topic all the time, also need to show user some other topics and collect feedback to see if user has other interested topics. known as partial/bandit

feedback problem. so the contextual bandits can help to balance between showing user the items they like and showing items you want feedback on.

- similar as reinforcement learning problem: not optimize for one single step , need to take a series of actions before seeing the rewards.

Chapter 10: Infrastructure and Tooling for MLOps

...

Chapter 11: The Human Side of Machine Learning

...