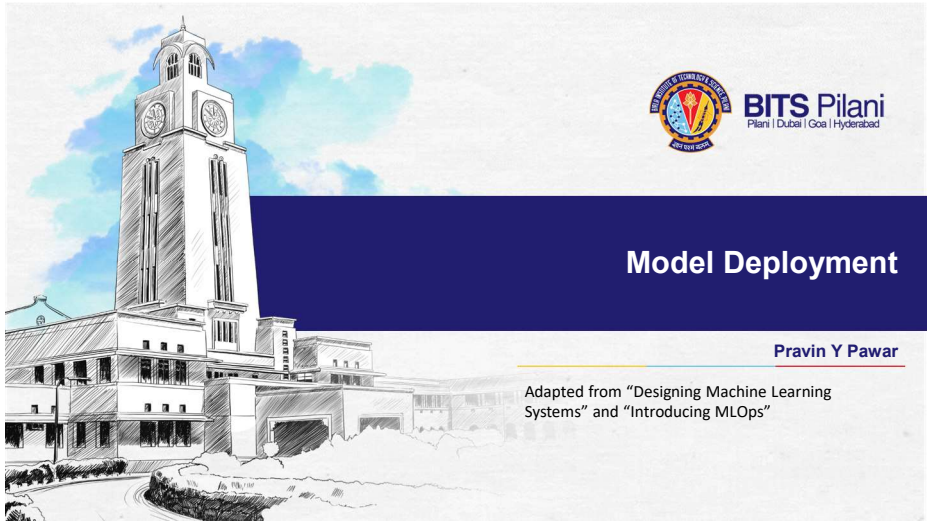


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model Deployment

Pravin Y Pawar

Adapted from "Designing Machine Learning Systems" and "Introducing MLOps"

Model Deployment

- "Deploy" is a loose term that generally means making model running and accessible
- To be deployed, model will have to leave the development environment
  - During model development, model usually runs in a development environment
  - can be deployed to a staging environment for testing
  - can be deployed to a production environment to be used by end users
- Production is a broad spectrum
  - For some teams, production means generating nice plots in notebooks to show to the business team
  - For other teams, production means keeping your models up and running for millions of users a day
- If your work is in the first scenario, production environment is similar to the development environment
- If your work is closer to the second scenario, welcome to word of model deployment!

Deployment is easy!

If you ignore all hard parts

- If want to deploy a model for friends to play with, all needs to be done is to
  - wrap predict function in a POST request endpoint using Flask or FastAPI,
  - put the dependencies this predict function needs to run in a container
  - push model and its associated container to a cloud service like AWS or GCP to expose the endpoint
- Can use this exposed endpoint for downstream applications:
  - e.g., when an application receives a prediction request from a user,
  - this request is sent to the exposed endpoint, which returns a prediction
- If familiar with the necessary tools, can have a functional deployment in an hour!

Machine Learning Deployment Myths

- Deploying an ML model can be very different from deploying a traditional software program
- This difference might cause people who have never deployed a model before
  - to either dread the process
  - or underestimate how much time and effort it will take
- Common myths about the deployment process
  - Myth 1: You Only Deploy One or Two ML Models at a Time
  - Myth 2: If We Don't Do Anything, Model Performance Remains the Same
  - Myth 3: You Won't Need to Update Your Models as Much
  - Myth 4: Most ML Engineers Don't Need to Worry About Scale

Deployment : Reality

- The hard parts include
  - making model available to millions of users
  - with a latency of milliseconds and 99% uptime,
  - setting up the infrastructure
    - so that the right person can be immediately notified when something goes wrong,
    - figuring out what went wrong
    - seamlessly deploying the updates to fix what's wrong
- In many companies,
  - the responsibility of deploying models falls into the hands of the same people who developed those models
- In many other companies, once
  - a model is ready to be deployed, it will be exported and handed off to another team to deploy it
- However, this separation of responsibilities can cause high overhead communications across teams and make it slow to update model.
  - It also can make it hard to debug should something go wrong.

Productionalization and Deployment

- A key component of MLOps
  - presents an entirely different set of technical challenges than developing the model
  - domain of the software engineer and the DevOps team
- Organizational challenges in managing the information exchange between the data scientists and these teams must not be underestimated!
  - without effective collaboration between the teams, delays or failures to deploy are inevitable

Model Deployment Types and Contents

what exactly is going into production, and what does a model consist of?

- Commonly two types of model deployment:
- Model-as-a-service, or live-scoring model
  - Typically the model is deployed into a simple framework to provide a REST API endpoint
  - (the means from which the API can access the resources it needs to perform the task)
  - that responds to requests in real time
- Embedded model
  - Here the model is packaged into an application, which is then published
  - A common example is an application that provides batch-scoring of requests
- What to-be-deployed models consist of depends, of course, on the technology chosen,
  - but typically they comprise a set of code (commonly Python, R, or Java) and data artifacts
    - can have version dependencies on runtimes and packages that need to match in the production environment
    - because the use of different versions may cause model predictions to differ

Model Deployment : Dependency Management

Model Export

- Can export the model to a portable format such as PMML, PFA, ONNX, or POJO
  - improves model portability between systems and simplify deployment
  - helps in reducing dependencies on the production environment
- Come at a cost
  - each format supports a limited range of algorithms,
  - sometimes the portable models behave in subtly different ways than the original
- Whether or not to use a portable format is a choice to be made based on a thorough understanding of the technological and business context.

Model Deployment: Dependency Management(2)

Containerization

- Containerization is an increasingly popular solution
  - to the headaches of dependencies when deploying ML models
- Container technologies such as Docker are lightweight alternatives to virtual machines,
  - allowing applications to be deployed in independent, self-contained environments,
  - matching the exact requirements of each model
- Also enable
  - new models to be seamlessly deployed using the blue-green deployment technique
  - scaling of compute resources for models
- Orchestrating many containers is the role of technologies such as Kubernetes
  - can be used both in the cloud and on-premise.

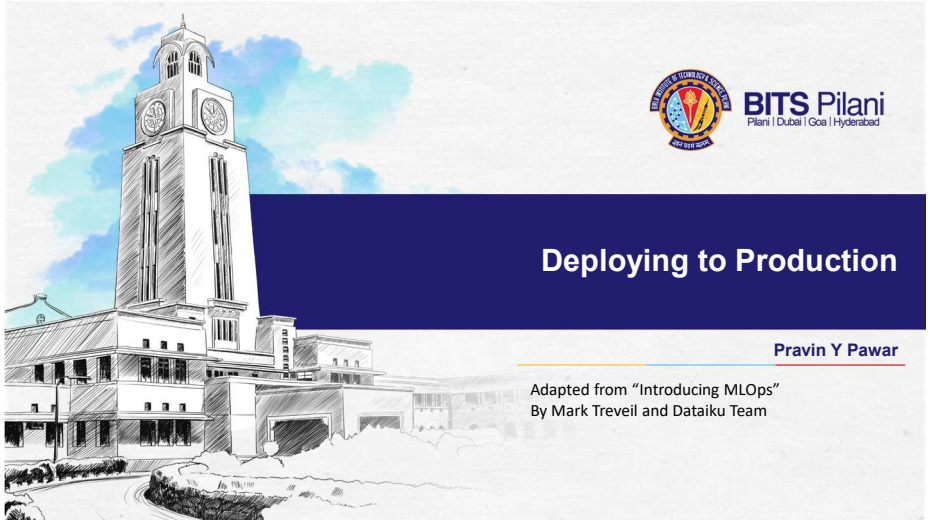
Model Deployment Requirements

What needs to be addressed – when moving from development to the production

- In customer-facing, mission-critical use cases, a more robust CI/CD pipeline is required
- Typically involves:
  1) 1Ensuring all coding, documentation and sign-off standards have been met
  2) Re-creating the model in something approaching the production environment
  3) Revalidating the model accuracy
  4) Performing explainability checks
  5) Ensuring all governance requirements have been met
  6) 6. Checking the quality of any data artifacts
  7) Testing resource usage under load
  8) Embedding into a more complex application, including integration tests

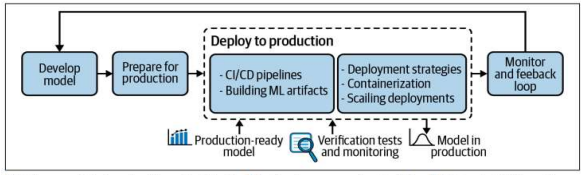
Model Deployment Requirements(2)

- One thing is for sure is needed:
  - rapid, automated deployment is always preferred to labor-intensive processes
- In heavily regulated industries (e.g., finance and pharmaceuticals),
  - governance and regulatory checks will be extensive
  - are likely to involve manual intervention
- The desire in MLOps, just as in DevOps, is to automate the CI/CD pipeline as far as possible
  - speeds up the deployment process
  - enables more extensive regression testing
  - reduces the likelihood of errors in the deployment



Deploying to Production

- Business leaders view rapid deployment of systems into production as key to maximizing business value
  - only true if deployment can be done smoothly and at low risk
- Lest dive into the concepts and considerations when deploying machine learning models to production
  - that impact and drive—the way MLOps deployment processes are built



Deployment to production highlighted in the larger context of the ML project life cycle

CI/CD Pipelines

A common acronym for continuous integration and continuous delivery (or put more simply, deployment)

- Forms a modern philosophy of agile software development and a set of practices and tools
  - to release applications more often and faster, while also better controlling quality and risk
- Ideas are decades old and already used to various extents by software engineers,
  - different people and organizations use certain terms in very different ways
- Essential to keep in mind that
  - these concepts should be tools to serve the purpose of delivering quality fast
  - first step is always to identify the specific risks present at the organization
- CI/CD methodology should be adapted based on the needs of the team and the nature of the business.

CI/CD for ML

- CI/CD concept apply just as well to machine learning systems
  - are a critical part of MLOps strategy
- An example of such pipeline could be:
  1. Build the model
     a. Build the model artifacts
     b. Send the artifacts to long-term storage
     c. Run basic checks (smoke tests/sanity checks)
     d. Generate fairness and explainability reports
  2. Deploy to a test environment
     a. Run tests to validate ML performance, computational performance
     b. Validate manually
  3. Deploy to production environment
     a. Deploy the model as canary
     b. Fully deploy the model

CI/CD for ML(2)

- Many scenarios are possible, depend on the application,
  - the risks from which the system should be protected
  - and the way the organization chooses to operate
- Generally, an incremental approach to building a CI/CD pipeline is preferred:
  - a simple or even naïve workflow on which a team can iterate
  - often much better than starting with complex infrastructure from scratch
- A starting project does not have the infrastructure requirements of a tech giant
  - can be hard to know up front which challenges deployments will present
- There are common tools and best practices,
  - but there is no one-size-fits-all CI/CD methodology
  - means the best path forward is starting from a simple (but fully functional) CI/CD workflow
  - then introducing additional or more sophisticated steps along the way as quality or scaling challenges appear

Building ML Artifacts

- The goal of a continuous integration pipeline is
  - to avoid unnecessary effort in merging the work from several contributors
  - also to detect bugs or development conflicts as soon as possible
- The very first step is using centralized version control systems
  - unfortunately, working for weeks on code stored only on a laptop is still quite common)
- The most common version control system is Git, an open source software
  - majority of software engineers across the world already use Git,
  - increasingly being adopted in scientific computing and data science
- Git allows for
  - maintaining a clear history of changes,
  - safe rollback to a previous version of the code,
  - multiple contributors to work on their own branches of the project before merging to the main branch, etc
- Git is appropriate for code, but not designed
  - to store other types of assets common in data science workflows, such as large binary files (for example, trained model weights),
  - or to version the data itself

ML Artifact

- Once code and data is in a centralized repository,
  - a testable and deployable bundle of the project must be built
  - are usually called artifacts in the context of CI/CD
- Each of the following elements needs to be bundled into an artifact
  - that goes through a testing pipeline and is made available for deployment to production:
    - Code for the model and its preprocessing
    - Hyperparameters and configuration
    - Training and validation data
    - Trained model in its runnable form
    - An environment including libraries with specific versions, environment variables, etc.
    - Documentation
    - Code and data for testing scenarios

The Testing Pipeline

- Testing pipeline can validate a wide variety of properties of the model contained in the artifact
  - good tests should make it as easy as possible to diagnose the source issue when they fail
- Automating tests as much as possible is essential and is a key component of efficient MLOps
- A lack of automation or speed wastes time,
  - also discourages the development team from testing and deploying often,
  - which can delay the discovery of bugs or design choices that make it impossible to deploy to production

Deployment concepts

- Integration
  - Process of merging a contribution to a central repository and performing more or less complex tests
  - typically merging a Git feature branch to the main branch
- Delivery
  - Same as used in the continuous delivery (CD) part of CI/CD,
  - Process of building a fully packaged and validated version of the model ready to be deployed to production
- Deployment
  - Process of running a new model version on a target infrastructure
  - Fully automated deployment is not always practical or desirable
- Release
  - In principle, release is yet another step, directing production workload to model
  - deploying a model version (even to the production infrastructure) does not necessarily mean that the production workload is directed to the new version
    - multiple versions of a model can run at the same time on the production infrastructure

Categories of Model Inferences

Two ways to approach model deployment

- Batch scoring,
  - where whole datasets are processed using a model, such as in daily scheduled jobs
- Real-time scoring,
  - where one or a small number of records are scored,
  - such as when an ad is displayed on a website and a user session is scored by models to decide what to display
- In some systems, scoring on one record is technically identical to requesting a batch of one!
- In both cases, multiple instances of the model can be deployed
  - to increase throughput and potentially lower latency

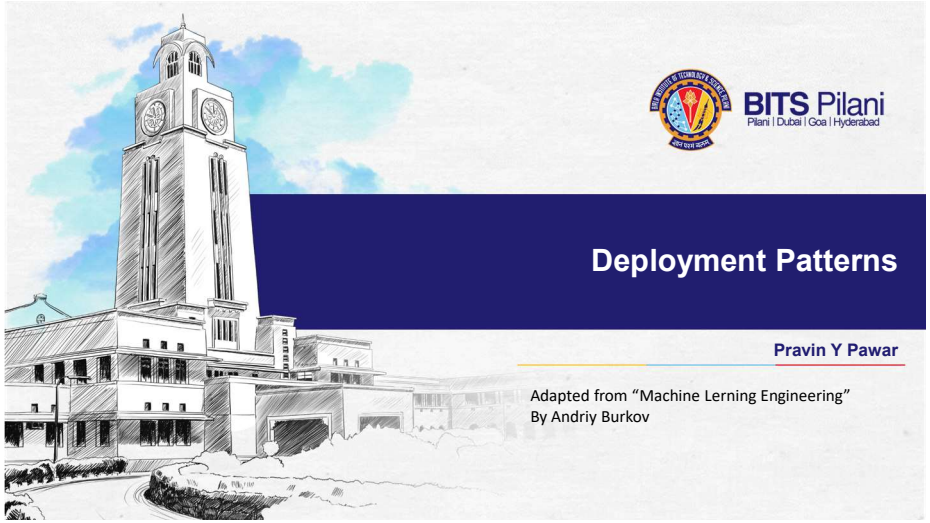
Considerations When Sending Models to Production

- When sending a new model version to production, first consideration is often to avoid downtime,
  - in particular for real-time scoring
- Blue-green or red-black— deployment
  - basic idea is that rather than shutting down the system, upgrading it, and then putting it back online,
  - a new system can be set up next to the stable one
  - and when it's functional, the workload can be directed to the newly deployed version
  - and if it remains healthy, the old one is shut down)
- Canary deployment
  - idea is that the stable version of the model is kept in production,
  - but a certain percentage of the workload is redirected to the new model, and results are monitored
  - usually implemented for real-time scoring, but a version of it could also be considered for batch

Maintenance in Production

Once a model is released, it must be maintained

- At a high level, there are three maintenance measures:
  - Resource monitoring
  - Health check
  - ML metrics monitoring
- Resource monitoring
  - Just as for any application running on a server, collecting IT metrics such as CPU, memory, disk, or network usage
  - can be useful to detect and troubleshoot issues
- Health check
  - Need to check if the model is indeed online and to analyze its latency
  - simply queries the model at a fixed interval (on the order of one minute) and logs the results
- ML metrics monitoring
  - about analyzing the accuracy of the model and comparing it to another version or detecting when it is going stale
  - may require heavy computation, this is typically lower frequency
- Finally, when a malfunction is detected, a rollback to a previous version may be necessary
  - critical to have the rollback procedure ready and as automated as possible;
  - testing it regularly can make sure it is indeed functional



Model Deployment Patterns

- Once the model has been built and thoroughly tested, it can be deployed
  - means to make it available for accepting queries generated by the users of the production system
- Once the production system accepts the query, the latter is transformed into a feature vector
  - The feature vector is then sent to the model as input for scoring
  - The result of the scoring then is returned to the user
- A trained model can be deployed in various ways
  - can be deployed on a server, or on a user's device
  - can be deployed for all users at once, or to a small fraction of users
- A model can be deployed following several patterns:
  - statically, as a part of an installable software package,
  - dynamically on the user's device,
  - dynamically on a server, or
  - via model streaming

Static Deployment

- Very similar to traditional software deployment
  - prepare an installable binary of the entire software
  - model is packaged as a resource available at the runtime
- Depending on the operating system and the runtime environment
  - Objects of both the model and the feature extractor can be packaged as a
    - part of a dynamic-link library (DLL on Windows),
    - Shared Objects (*.so files on Linux),
    - or be serialized and saved in the standard resource location for virtual machine-based systems,
    - such as Java and .Net.

Static Deployment(2)

Pros and Cons

- Many advantages:
  - software has direct access to the model, so the execution time is fast for the user
  - user data doesn't have to be uploaded to the server at the time of prediction
    - saves time and preserves privacy
  - model can be called when the user is offline
  - software vendor doesn't have to care about keeping the model operational
    - becomes the user's responsibility
- Several drawbacks:
  - The separation of concerns between the machine learning code and the application code isn't always obvious
    - makes it harder to upgrade the model without also having to upgrade the entire application
  - If the model has certain computational requirements for scoring (such as access to an accelerator or a GPU)
    - may add complexity and confusion as to where the static deployment can or cannot be used

Dynamic Deployment on User's Device

- Similar to a static deployment, in the sense the user runs a part of the system as a software application on their device
- Difference is that in dynamic deployment, the model is not part of the binary code of the application
- Achieves better separation of concerns!
  - Pushing model updates is done without updating the whole application running on the user's device
- Dynamic deployment can be achieved in several ways:
  - by deploying model parameters,
  - by deploying a serialized object, and
  - by deploying to the browser

Deployment of Model Parameters

- In this deployment scenario, the model file only contains the learned parameters
  - user's device has installed a runtime environment for the model
- Some machine learning packages, like TensorFlow,
  - have a lightweight version that can run on mobile devices
- Alternatively, frameworks such as Apple's Core ML allow running models on Apple devices
  - created using popular packages, including scikit-learn, Keras, and XGBoost

Deployment of a Serialized Object

- Model file is a serialized object that the application would deserialize
- The advantage is that don't need to have a runtime environment for model on the user's device
  - all needed dependencies will be deserialized with the object of the model
- An evident drawback is that an update might be quite "heavy,"
  - which is a problem if your software system has millions of users

Deploying to Browser

- Most modern devices have access to a browser, either desktop or mobile
- Some machine learning frameworks, such as TensorFlow.js,
  - have versions that allow to train and run a model in a browser, by using JavaScript as a runtime
- Even possible to train a TensorFlow model in Python,
  - then deploy it to, and run it in the browser's JavaScript runtime environment
  - if a GPU (graphics processing unit) is available on the client's device, Tensorflow.js can leverage it

Dynamic Deployment on User's Device(2)

Advantages and Drawbacks

- Advantages
  - Calls to the model will be fast for the user
  - Reduces the impact on the organization's servers, as most computations are performed on the user's device
- If the model is deployed to the browser,
  - advantage is organization's infrastructure only needs to serve a web page that includes the model's parameters
  - A downside is bandwidth cost and application startup time might increase.
    - users must download the model's parameters each time they start the web application
    - as opposed to doing it only once when they install an application

Dynamic Deployment on User's Device(3)

Advantages and Drawbacks

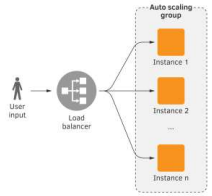
- Monitoring
  - Deploying to a user's device makes it difficult to monitor the model performance
- Model updates
  - A serialized object can be quite voluminous
  - Some users may be offline during the update, or even turn off all future updates
  - may end up with different users using very different model versions
  - becomes difficult to upgrade the server-side part of the application
- Third-party analyses
  - Deploying models on the user's device means that the model easily becomes available for third-party analyses
  - may try to reverse-engineer the model to reproduce its behavior
  - may search for weaknesses by providing various inputs and observing the output
  - may adapt their data so the model predicts what they want

Dynamic Deployment on a Server

- Because of the complications with other approaches, and problems with performance monitoring,
- Most frequent deployment pattern is to place the model on a server (or servers),
  - make it available as
    - a Representational State Transfer application programming interface (REST API) in the form of a web service,
    - Google's Remote Procedure Call (gRPC) service
- Four common practices
  - Deployment on a Virtual Machine
  - Deployment in a Container
  - Serverless Deployment
  - Model Streaming

Deployment on a Virtual Machine(VM)

- In a typical web service architecture deployed in a cloud environment
  - predictions are served in response to canonically-formatted HTTP requests
- A web service running on a virtual machine
  - receives a user request containing the input data,
  - calls the machine learning system on that input data
  - then transforms the output of the machine learning system into the output JSON or XML string
- To cope with high load, several identical VMs are running in parallel
  - A load balancer dispatches the incoming requests to a specific virtual machine
  - VMs can be added and closed manually, or be a part of an autoscaling group that launches
  - VMs can be terminated virtual machines based on their usage



: Deploying a machine learning model as a web service on a virtual machine.

Deployment on a Virtual Machine(VM)2

- In Python, a REST API web service is usually implemented
  - using a web application framework such as Flask or FastAPI
- TensorFlow, a popular framework used to train deep models,
  - comes with TensorFlow Serving, a built-in gRPC service
- Advantage: Architecture of the software system is conceptually simple: a typical web or gRPC service
- Downsides
  - Need to maintain servers (physical or virtual)
  - If virtualization is used, there is an additional computational overhead due to virtualization and running multiple OS
  - Network latency, which can be a serious issue, depending on how fast you need to process scoring results
  - has a relatively higher cost, compared to deployment in a container, or a serverless deployment

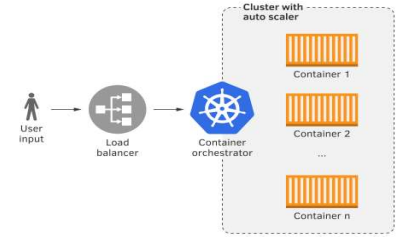
Deployment in a Container

- A more modern alternative to a virtual-machine-based deployment
  - considered more resource-efficient and flexible than with virtual machines
- A container is similar to a virtual machine
  - in the sense that it is also an isolated runtime environment with its own filesystem, CPU, memory, and process space
- The main difference, however, is that all containers are running on the same virtual or physical machine
  - share the operating system, while each virtual machine runs its own instance of the operating system
- Deployment Process
  - The machine learning system and the web service are installed inside a container
    - Usually, a container is a Docker container, but there are alternatives
  - Then a container-orchestration system is used to run the containers on a cluster of physical or virtual servers
    - A typical choice of a container-orchestration system for running on-premises or in a cloud platform, is Kubernetes
    - Some cloud platforms provide both their own container-orchestration engine, such as AWS Fargate and Google Kubernetes Engine

Deployment in a Container(2)

Organization

- Virtual or physical machines are organized into a cluster,
  - whose resources are managed by the container orchestrator
- New virtual or physical machines can be manually added to the cluster, or closed
- If your software is deployed in a cloud environment,
  - a cluster autoscaler can launch (and add to the cluster) or terminate virtual machines
  - based on the usage of the cluster.



Deploying a model as a web service in a container running on a cluster.

Deployment in a Container(3)

- Advantage
  - More resource-efficient as compared to the deployment on a virtual machine
  - Allows the possibility to automatically scale with scoring requests
  - Allows us to scale-to-zero- reduced down to zero replicas when idle and brought back up if there is a request to serve
    - the resource consumption is low compared to always running services
    - leads to less power consumption and saves cost of cloud resources
- Drawback
  - Containerized deployment is generally seen as more complicated, and requires expertise

Serverless Deployment

- Several cloud services providers, including Amazon, Google, and Microsoft, offers serverless computing
  - Lambda-functions on Amazon Web Services, and Functions on Microsoft Azure and Google Cloud Platform
- The serverless deployment consists of preparing a zip archive
  - with all the code needed to run the machine learning system (model, feature extractor, and scoring code)
  - must contain a file with a specific name that contains a specific function
  - is uploaded to the cloud platform and registered under a unique name
- The cloud platform provides an API to submit inputs to the serverless function
  - specifies its name, provides the payload, and yields the outputs
- The cloud platform takes care of
  - deploying the code and the model on an adequate computational resource,
  - executing the code,
  - routing the output back to the client

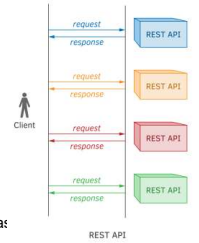
Serverless Deployment(2)

Advantages and Limitations

- Advantages to relying on serverless deployment
  - don't have to provision resources such as servers or virtual machines
  - don't have to install dependencies, maintain, or upgrade the system
  - highly scalable and can easily and effortlessly support thousands of requests per second
  - support both synchronous and asynchronous modes of operation
  - cost-efficient: only pay for compute-time
  - simplifies canary deployment, or canarying
  - Rollbacks are also very simple in the serverless deployment because it is easy to switch back to the previous version of the function
- Limitations
  - Restrictions by the cloud service provider
    - the function's execution time, zip file size, and amount of RAM available on the runtime
  - Zip file size limit can be a challenge - A typical model requires multiple heavyweight dependencies
  - Unavailability of GPU access can be a significant limitation for deploying deep models

Model Serving – REST API Revisited

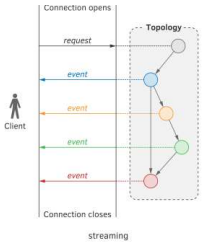
- In complex systems, there can be many models applied to the same input
  - a model can input a prediction from another model
- For example, the input may be a news article
  - One model can predict the topic of the article
  - Another model can extract named entities
  - Third model can generate a summarization of the article, and so on
- According to the REST API deployment pattern, need one REST API per model
  - client would call one API by sending a news article as a part of the request - get the topic as response
  - client calls another API by sending a news article, and gets the named entities as response; etc.

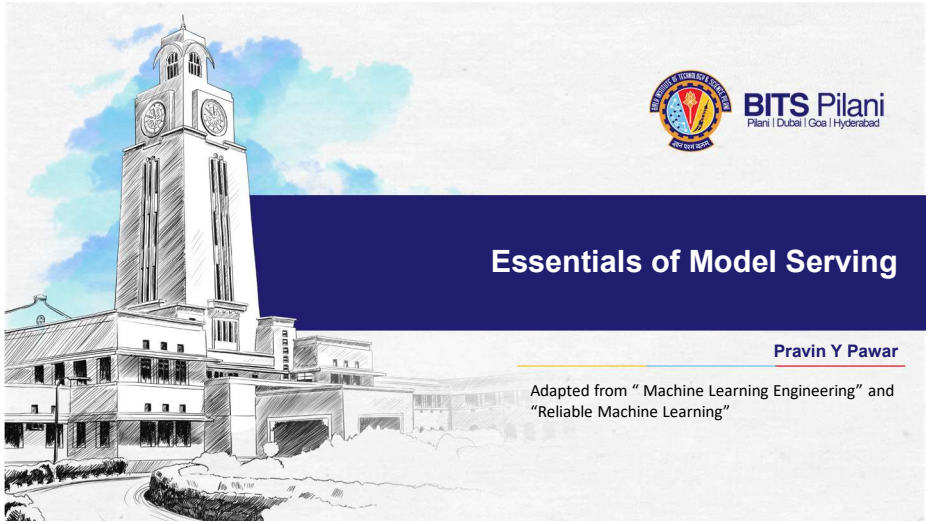


Model Streaming

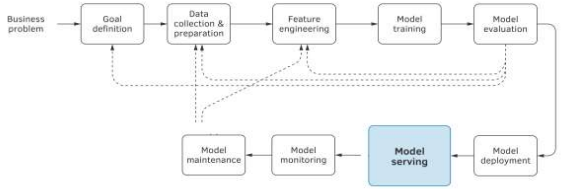
Can be seen as an inverse to the REST API

- Streaming works differently
  - All models, as well as the code needed to run them, are registered within a stream-processing engine (SPE)
  - Apache Storm, Apache Spark, and Apache Flink
  - Or, they are packaged as an application based on a stream-processing library (SPL),
  - such as Apache Samza, Apache Kafka Streams, and Akka Streams
- Based on notion of data processing topology
  - Input data flows in as an infinite stream of data elements sent by the client
  - Following a predefined topology, each data element in the stream undergoes a transformation in the nodes of the topology
  - Transformed, the flow continues to other nodes.
- In a stream-processing application, nodes transform their input in some way,
  - then either,
    - send the output to other nodes, or
    - send the output to the client, or
    - persist the output to the database or a filesystem.





Model Serving in ML Lifecycle



Machine learning project life cycle.

Model Serving

- Model built in training phase, needs to be taken into the world so that it can start predicting!
- Aka Model Serving
- Process of creating a structure to ensure system can ask the model to make predictions on new examples, and return those predictions to the people or systems that need them

Properties of the Model Serving Runtime

- The model serving runtime is the environment in which the model is applied to the input data
  - The runtime properties are dictated by the model deployment pattern
- However, an effective runtime will have several additional properties such as
  - Security and Correctness
  - Ease of Deployment
  - Guarantees of Model Validity
  - Ease of Recovery
  - Avoidance of Training/Serving Skew
  - Avoidance of Hidden Feedback Loops

Properties of the Model Serving Runtime(2)

Security and Correctness

- The runtime is responsible for authenticating the user's identity, and authorizing their requests.
- Things to check are:
  - whether a specific user has authorized access to the models they want to run,
  - whether the names and the values of parameters passed correspond to the model's specification,
  - whether those parameters and their values are currently available to the user.

Properties of the Model Serving Runtime(3)

Ease of Deployment and Recovery

- Ease of Deployment
  - The runtime must allow the model to be updated with minimal effort
    - ideally, without affecting the entire application
  - If the model was deployed as a web service on a physical server,
    - then a model update must be as simple as replacing one model file with another, and restarting the web service
  - If the model was deployed as a virtual machine instance or container,
    - then the instances or containers running the old version of the model should be replaceable by gradually stopping the running instances and starting new instances from a new image
    - The same principle applies to the orchestrated containers
- Ease of Recovery
  - An effective runtime allows easy recovery from errors by rolling back to previous versions
    - The recovery from an unsuccessful deployment should be produced in the same way, and with the same ease, as the deployment of an updated model
    - The only difference is that, instead of the new model, the previous working version will be deployed.

Properties of the Model Serving Runtime(4)

Avoidance of Training/Serving Skew

- When it concerns feature extraction, Strongly recommended to avoid using two different codebases,
  - one for training the model, and one for scoring in production
  - even a tiny difference between two versions of feature extractor code may lead to suboptimal or incorrect model performance
- The engineering team may reimplement the feature extractor code for production for many reasons
  - most common being data analyst's code is inefficient or incompatible with the production ecosystem
- Runtime should allow easy access to the feature extraction code for various needs,
  - including model retraining, ad-hoc model calls, and production.
- One way to implement it is by wrapping the feature extraction object into a separate web service
- If cannot avoid using two different codebases to generate features for training and production,
  - then the runtime should allow for the logging of feature values generated in the production environment
  - Those values should then be used as training values

Key questions for model serving

- Lot of ways to create structures around the model that support serving,
- Each with very different sets of trade-offs
- Useful to think through specific questions about needs of system
  - What will be the load to model?
  - What are the prediction latency needs of model?
  - Where does the model need to live?
  - What are the hardware needs for model?
  - How will the serving model be stored, loaded, versioned and updated?
  - What will feature pipeline for serving look like?

What will be the load to model?

- QPS (Queries Per second) – need to know in serving environment what is the level of traffic that model will be asked to handle – when queries are done on demand
  - Model serving predictions to millions of daily users may need handle thousands of QPS
  - Model runs audio recognizer on mobile device may run at a few QPS
  - Model predicting real estate prices might not be served on demand at all!
- Few basic strategies to handle large traffic loads
  - Replicate model across many machines and run these parallel – may be on cloud
  - Use more powerful hardware – accelerators like GPU or specialized chips
  - Tune computation cost of model itself by using fewer features or layers or parameters
  - Model cascades can be effective at cost reduction

What are the prediction latency needs of model?

- Prediction latency is time between the moment request is made and moment response is received
- Acceptable prediction latency can vary dramatically among applications
  - Is major determiner of serving architecture choices
- Taken together, latency and traffic load define overall computational needs of ML system
- If latency is too high, can be mitigated by
  - Using more powerful hardware
  - Making model less expensive to compute
- But creating a larger number of model replicas is not a solution for latency issue!

Where does the model need to live?

Model needs to be stored on physical device in specific location – home of model

- This choice has significant implications on overall serving architecture
- On a local machine
  - Not a practical solution – may be suitable for small batch predictions
  - Not recommended beyond small-scale prototyping or bespoke uses
- On servers owned or managed by our organization
  - Important when specific privacy or security concerns are in place
  - Right option if latency is hypercritical concern or if speciality hardware is needed to run models
  - Can limit flexibility in terms of scaling up/down, require special attention to monitoring
- In the cloud
  - Can allow easy scaling overall computation footprint up or down
  - Can be done by
    - running model servers on own virtual servers and controlling how many of them to be used
    - managed inference service
- On-device
  - Everything from mobile phones to smart watches, digital assistants, automobiles, printers etc.
  - Has strict constraints on model size, because of limited memory and power

What are the hardware needs for model?

- Range of computational hardware and chip options have emerged
  - Enabled dramatic improvements in serving efficiency for various model types
- Multicore CPUs
  - Suitable for non-deep methods, non deep matrix multiplications
- Hardware accelerators – commonly GPU
  - Choice of serving deep learning models as they involve dense matrix multiplications
  - But has drawbacks
    - Specialized hardware – need to invest or use a cloud service – costly options
    - Not suited for operations not involving large amounts of dense matrix calculations

How will the serving model be stored, loaded, versioned and updated?

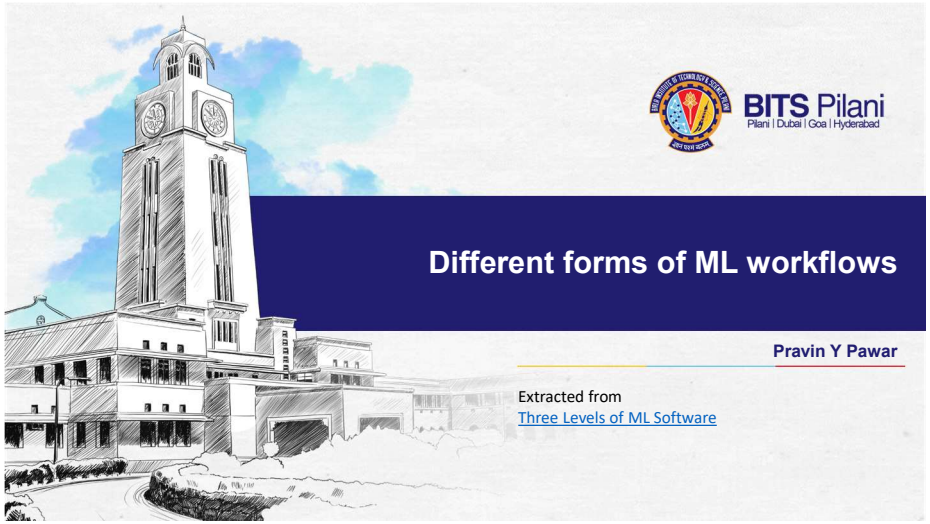
Model is physical object – has size and needs space

- Model serving in offline environment be stored on disk and loaded by specific binaries in batch jobs
  - Main requirement is disk space to store model and I/O capacity to load model from disk and RAM needed to load model into memory for use
- Model used in online serving needs to be stored in RAM in dedicated machine
  - For high-throughput services in latency critical settings, copies of these models likes to be stored and served in many replica machine in parallel
- Eventually these models needs to be updated by retraining
- means needs to swap version of model currently used in serving on a given machine with a new version
- Deciding exactly how many versions to be supported and at what capacity is important architectural choice
  - Requires balancing resourcing, system complexity and organizational requirements together

What will feature pipeline for serving look like?

Feature needs to be processed at serving time as well as at training time

- Any feature processing or other data manipulation that is done to data at training time will almost certainly needs to repeated for all examples sent to model at serving time
  - Computational requirements for this may be considerable
- Actual code used to turn incoming examples data to features of model may be different at serving time from the code used for similar tasks at training time
  - Main source of classic training-serving skew and bugs notoriously difficult to detect and debug
- Promise is in form of feature stores – handle both training and serving together in a single package
- Creating feature for model to use at serving time is key source of latency
  - Means serving feature pipeline is far from an afterthought
  - But is indeed often most production-critical part of the entire serving stack



Different forms of ML workflows

- Operating an ML model might assume several architectural styles
- Primarily Four architectural patterns which are classified along two dimensions:
  - ML Model Training
  - ML Model Prediction
- For sake of simplicity disregard the third dimension - ML Model Type
  - which denotes the type of machine learning algorithm such as
    - Supervised
    - Unsupervised
    - Semi-supervised
    - and Reinforcement Learning

Model Training Patterns

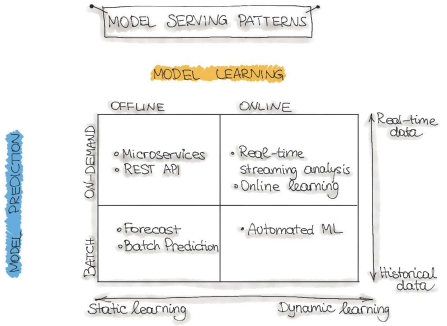
- Two ways to perform ML Model Training:
  - Offline learning
  - Online learning
- Offline learning (aka batch or static learning)
  - The model is trained on a set of already collected data
  - After deploying to the production environment, the ML model remains constant until it re-trained
  - Model will see a lot of real-live data and becomes stale - 'model decay' and should be carefully monitored
- Online learning (aka dynamic learning)
  - The model is regularly being re-trained as new data arrives, e.g. as data streams
  - Usually the case for ML systems that use time-series data, such as sensor, or stock trading data to accommodate the temporal effects in the ML model

Model Prediction Patterns

- Two modes to denote the mechanics of the ML model to makes predictions
  - Batch predictions
  - Real-time predictions
- Batch predictions
  - The deployed ML model makes a set of predictions based on historical input data
  - often sufficient for data that is not time-dependent, or when it is not critical to obtain real-time predictions as output
- Real-time predictions (aka on-demand predictions)
  - Predictions are generated in real-time using the input data that is available at the time of the request

ML architecture patterns

Forecast, Web-Service, Online Learning, and AutoML



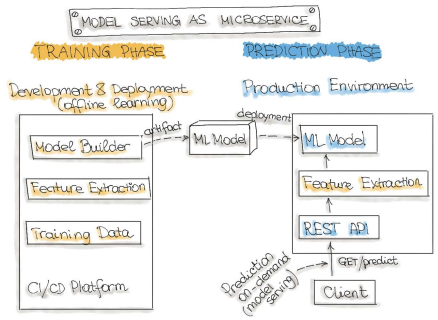
Forecast

- Widely spread in academic research or data science education (e.g., Kaggle or DataCamp)
  - used to experiment with ML algorithms and data as it is the easiest way to create a machine learning system
  - not very useful in an industry setting for production systems (e.g. mobile applications)
- Usually involves steps
  - take an available dataset
  - train the ML model
  - run this model on another (mostly historical) data
  - makes predictions

Web-Service (or Microservices)

Architecture for wrapping trained models as deployable services

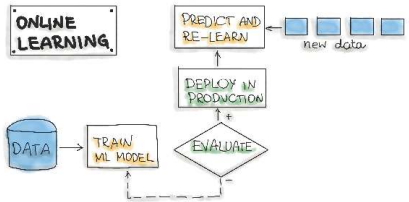
- The most commonly described deployment architecture for ML models
- The web service takes input data and outputs a prediction for the input data points
  - Model is trained offline on historical data, but it uses real-live data to make predictions
  - Model remains constant until it is re-trained and re-deployed into the production system.
- The difference from a forecast (batch predictions) is that
  - the ML model runs near real-time
  - handles a single record at a time instead of processing all the data at once



Online Learning (Real-time streaming analytics)

Most dynamic way to embed machine learning into a production system

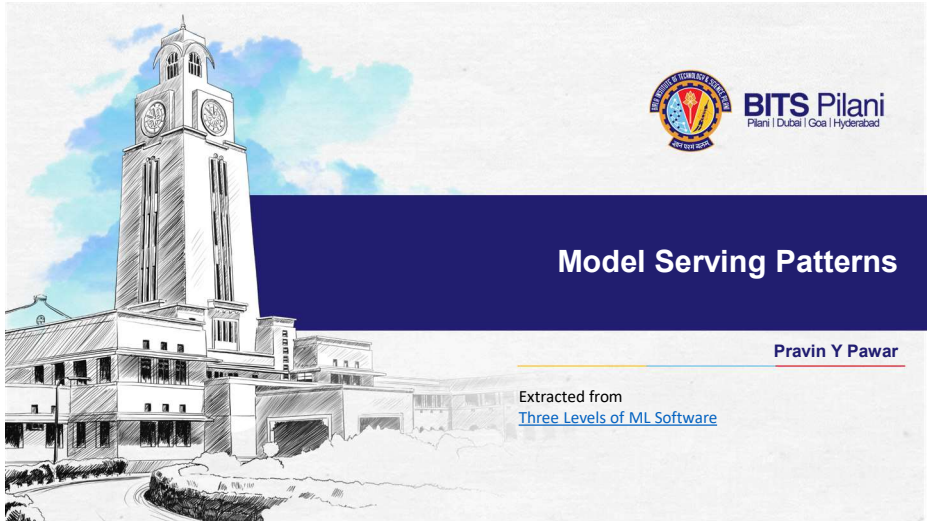
- Confusing name because the core learning or ML model training is usually not performed on the live system
  - call it incremental learning- term online learning is already established within the ML community
- In this type of ML workflow
  - ML learning algorithm is continuously receiving a data stream, either as single data points or in small groups called mini-batches
  - System learns about new data on the fly as it arrives, so the ML model is incrementally being re-trained with new data
  - Continually re-trained model is instantly available as a web service
  - Technically works well with the lambda architecture in big data systems
  - Model would typically run as a service on a Kubernetes cluster or similar
- A big difficulty with the online learning system in production is that
  - if bad data is entering the system, the ML model, as well as the whole system performance, will increasingly decline



AutoML

Sophisticated version of online learning

- AutoML is getting a lot of attention
  - considered the next advance for enterprise ML
  - promises training ML models with minimal effort and without machine learning expertise
  - User needs to provide data, and the AutoML system automatically selects an ML algorithm and configures the selected algorithm
  - very experimental way to implement ML workflows
  - usually provided by big cloud providers, such as Google or MS Azure
  - Instead of updating the model, need to execute an entire ML model training pipeline in production that results in new models on the fly



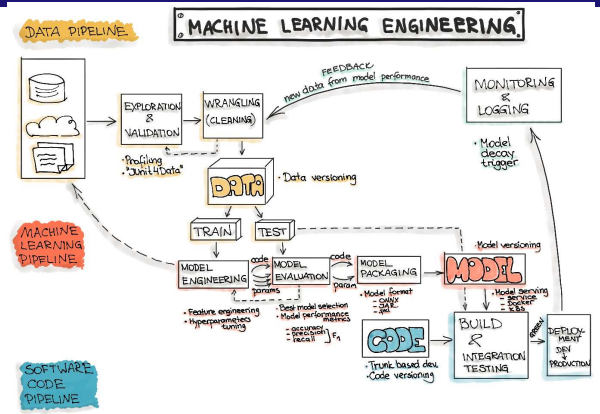
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model Serving Patterns

Pravin Y Pawar

Extracted from
Three Levels of ML Software

Machine Learning Workflow



Code: Deployment Pipelines

- The final stage of delivering an ML project includes the following three steps:
  - Model Serving
  - Model Performance Monitoring
  - Model Performance Logging
- Model Serving
  - The process of deploying the ML model in a production environment
- Model Performance Monitoring
  - The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation
  - interested in ML-specific signals, such as prediction deviation from previous model performance
  - signals might be used as triggers for model re-training
- Model Performance Logging
  - Every inference request results in a log-record.

Model Serving Patterns

- Three components should be considered when ML model is served in a production environment
  - The inference is the process of getting data to be ingested by a model to compute predictions
  - requires a model, an interpreter for the execution, and input data
- Deploying an ML system to a production environment includes two aspects,
  - First deploying the pipeline for automated retraining and ML model deployment
  - Second, providing the API for prediction on unseen data
- Model serving is a way to integrate the ML model in a software system
- Five patterns to put the ML model in production:
  - Model-as-Service
  - Model-as-Dependency
  - Precompute
  - Model-on-Demand
  - and Hybrid-Serving

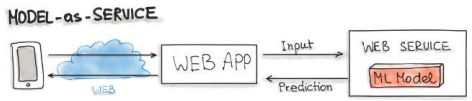
Model Serving Patterns(2)

Machine Learning Model Serving Taxonomy

Machine Learning Model Serving Taxonomy			
	ML Model		
Service & Versioning	Together with the consuming application	Independent from the consuming application	
Compile/ Runtime Availabilty	Build & runtime available	Available re-motely through REST API/RPC	Available at the runtime scope
Serving Patterns	"Model-as-Dependency"	"Model-as-Service"	"Precompute" and "Model on Demand"
	Hybrid Model Serving (Federated Learning)		

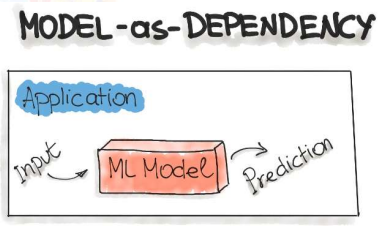
Model-as-Service

- A common pattern for wrapping an ML model as an independent service
  - can wrap the ML model and the interpreter within a dedicated web service
  - applications can request through a REST API or consume as a gRPC service
- Used for various ML workflows
  - Forecast
  - Web Service
  - Online Learning



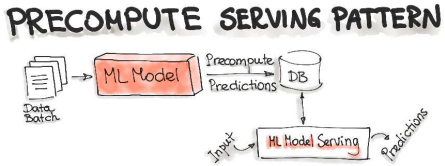
Model-as-Dependency

- Probably the most straightforward way to package an ML model
  - mostly used for implementing the Forecast pattern.
- A packaged ML model is considered as a dependency within the software application
  - For example, the application consumes the ML model like a conventional jar dependency by invoking the prediction method and passing the values
  - The return value of such method execution is some prediction that is performed by the previously trained ML model



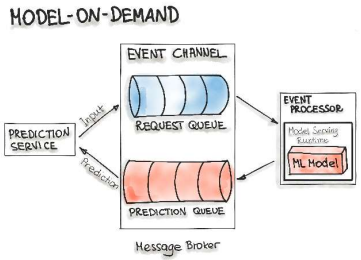
Precompute Serving Pattern

- Tightly related to the Forecast ML workflow
- Use an already trained ML model and precompute the predictions for the incoming batch of data
  - Resulting predictions are persisted in the database
  - For any input request, query the database to get the prediction result



Model-on-Demand

- Treats the ML model as a dependency that is available at runtime
  - contrary to the Model-as-Dependency pattern, has its own release cycle and is published independently
- Message-broker architecture is typically used for such on-demand model serving
  - contains two main types of architecture components:
    - a broker component
    - an event processor component
  - Broker component is the central part that contains the event channel that are utilised within the event flow
  - The event channels, which are enclosed in the broker component, are message queues
  - Message broker allows one process to write prediction-requests in a input queue
  - Event processor contains the model serving runtime and the ML model
    - connects to the broker, reads these requests in batch from the queue and sends them to the model to make the predictions
- The model serving process runs the prediction generation on the input data
  - writes the resulted predictions to the output queue
  - queued prediction results are pushed to the prediction service that initiated the prediction request

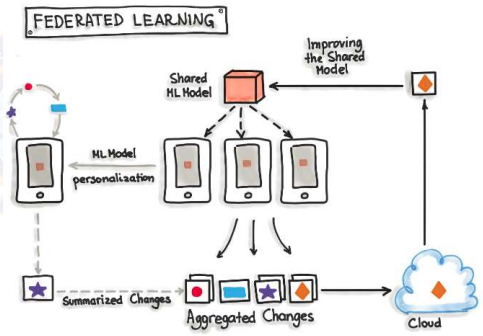


Hybrid-Serving (Federated Learning)

- Unique in the way it does, there is not only one model that predicts the outcome, but there are also lots of it
  - Exactly spoken there are as many models as users exist, in addition to the one that's held on a server
- Start with the unique model, the one on the server
  - Model on the server-side is trained only once with the real-world data
  - sets the initial model for each user
  - a relatively general trained model so it fits for the majority of users
- On the other side, there are the user-side models - really unique models
  - possible for the devices to train their own models due to hardware enhancements
  - devices will train their own highly specialized model for their own user
  - Once in a while, the devices send their already trained model data (not the personal data) to the server
- Then the server model will be adjusted
  - the actual trends of the whole user community will be covered by the model
  - this updated server model is set to be the new initial model that all devices are using

Hybrid-Serving (Federated Learning) 2

- Not having any downsides for the users, while the server model gets updated, this happens only when the device is idle, connected to WiFi and charging
  - testing is done on the devices, therefore the newly adopted model from the server is sent to the devices and tested for functionality
- Big benefit
  - data used for training and testing, which is highly personal, never leaves the devices while still capturing all data that is available
  - possible to train highly accurate models while not having to store tons of (probably personal) data in the cloud
- Constraints
  - mobile devices are less powerful
  - the training data is distributed across millions of devices and these are not always available for training
  - Exactly for this TensorFlow Federated (TFF) has been created - lightweight form of TensorFlow

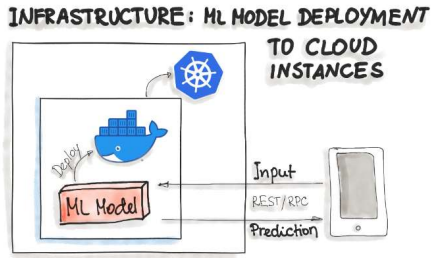


Deployment Strategies

- Common ways for wrapping trained models as deployable services, namely deploying ML models as
  - Docker Containers to Cloud Instances
  - Serverless Functions

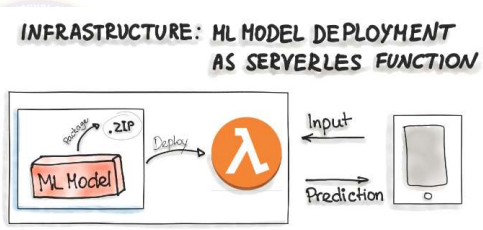
Deploying ML Models as Docker Containers

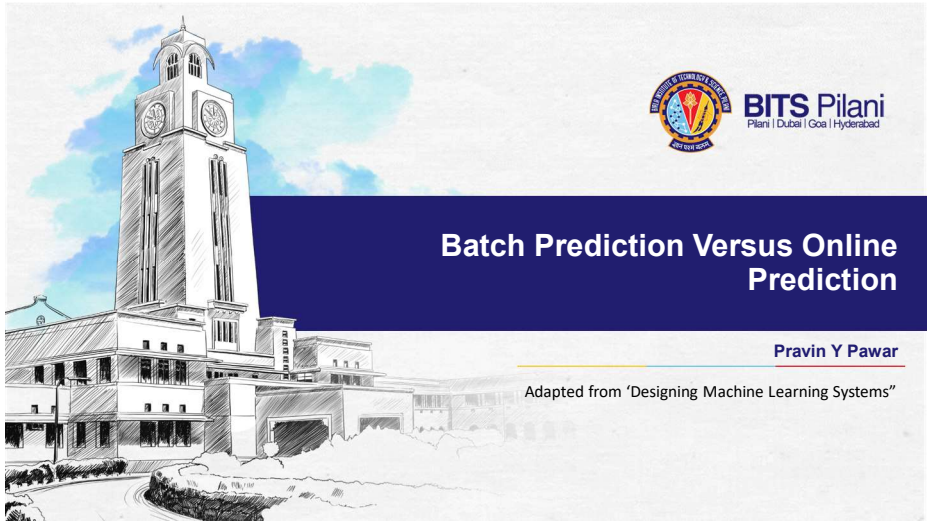
- No standard, open solution to ML model deployment!
- Containerization becomes the de-facto standard for delivery
  - as ML model inference being considered stateless, lightweight, and idempotent
  - means deploy a container that wraps an ML model inference code
- Docker is considered to be de-facto standard containerization technology
  - for on-premise, cloud, or hybrid deployments
- One ubiquitous way is to package the whole ML tech stack (dependencies) and the code for ML model prediction into a Docker container
  - Then Kubernetes or an alternative (e.g. AWS Fargate) does the orchestration
  - The ML model functionality, such as prediction, is then available through a REST API (e.g. implemented as Flask application)



Deploying ML Models as Serverless Functions

- Various cloud vendors already provide machine-learning platforms - can deploy model with their services
  - Amazon AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio, and IBM Watson Machine Learning
  - Commercial cloud services also provide containerization of ML models such as AWS Lambda and Google App Engine servlet host
- In order to deploy an ML model as a serverless function,
  - the application code and dependencies are packaged into .zip files, with a single entry point function
  - could be managed by major cloud providers such as Azure Functions, AWS Lambda, or Google Cloud Functions
  - However, attention should be paid to possible constraints of the deployed artifacts such as the size of the artifacts





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Batch Prediction Versus Online Prediction

Pravin Y Pawar

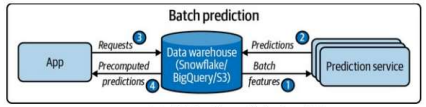
Adapted from 'Designing Machine Learning Systems"

Three main modes of prediction

- Batch prediction, which uses only batch features
- Online prediction that uses only batch features (e.g., precomputed embeddings)
  - Aka on-demand prediction
- Online prediction that uses both batch features and streaming features
  - aka streaming prediction

Batch prediction

- When predictions are generated periodically or whenever triggered
  - Predictions are stored somewhere, such as in SQL tables or an in-memory database, and retrieved as needed
- For example,
  - Netflix might generate movie recommendations for all of its users every four hours,
  - the precomputed recommendations are fetched and shown to users when they log on to Netflix
- AKA asynchronous prediction: predictions are generated asynchronously with requests

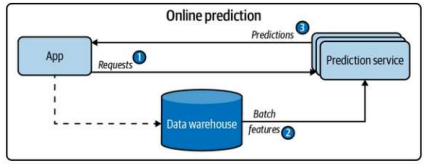


A simplified architecture for batch prediction

Online prediction

Using only batch features

- When predictions are generated and returned as soon as requests for these predictions arrive
  - For example, enter an English sentence into Google Translate and get back its French translation immediately
  - aka on-demand prediction
- Traditionally, when doing online prediction, requests are sent to the prediction service via RESTful APIs
  - aka synchronous prediction: predictions are generated in synchronization with requests



A simplified architecture for online prediction that uses only batch features

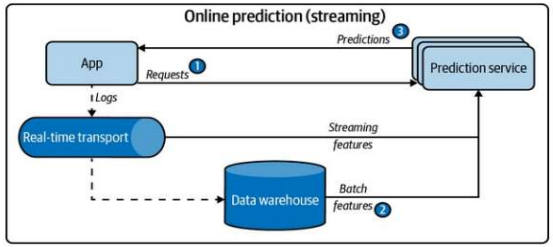
Online prediction

Using both batch and streaming features - "streaming prediction"

- Batch features
  - features computed from historical data, such as data in databases and data warehouses
  - for e.g. a restarents rating
- Streaming features
  - Features computed from streaming data — data in real-time transports
  - for e.g. estimation for delivery time
- In batch prediction, only batch features are used
- In online prediction, however, it's possible to use both batch features and streaming features.
  - For example, after a user puts in an order on DoorDash, they might need the following features to estimate the delivery time:
    - The mean preparation time of this restaurant in the past
    - In the last 10 minutes, how many other orders they have, and how many delivery people are available

Streaming prediction

A simplified architecture for online prediction that uses both streaming features and batch features



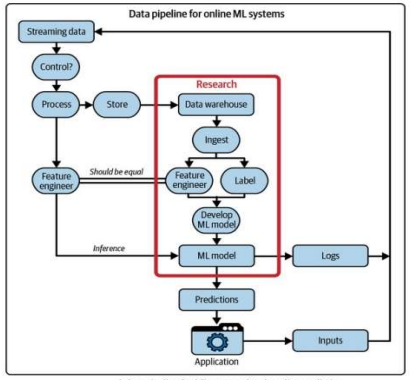
A simplified architecture for online prediction that uses both batch features and streaming features

Online vs Batch prediction

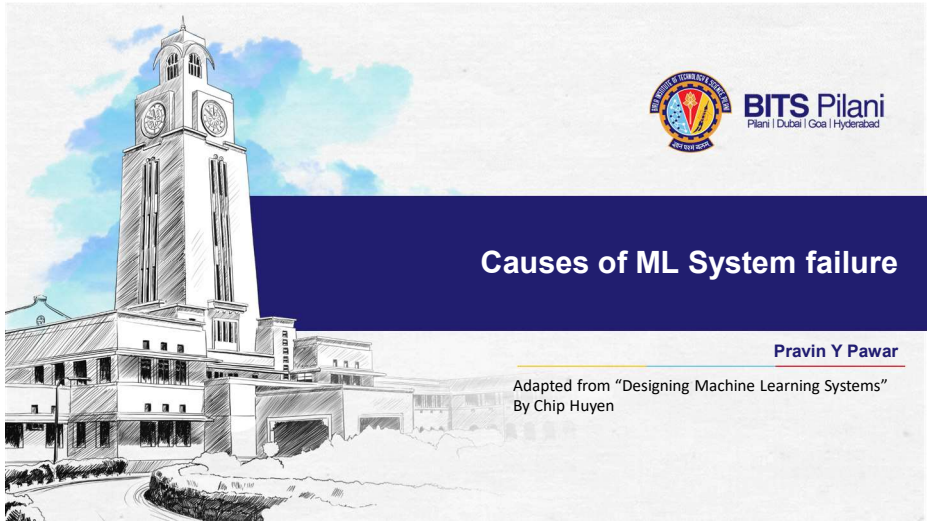
Some key differences between batch prediction and online prediction

	Batch prediction (asynchronous)	Online prediction (synchronous)
Frequency	Periodical, such as every four hours	As soon as requests come
Useful for	Processing accumulated data when you	When predictions are needed as soon as
	don't need immediate results (such as recommender systems)	a data sample is generated (such as fraud detection)
Optimized for	High throughput	Low latency

Unifying Batch Pipeline and Streaming Pipeline



A data pipeline for ML systems that do online prediction



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Causes of ML System failure

Pravin Y Pawar

Adapted from "Designing Machine Learning Systems"
By Chip Huyen

ML system failure

- A failure happens when one or more expectations of the system is violated
- In traditional software, mostly care about a system's operational expectations:
  - whether the system executes its logic within the expected operational metrics,
  - e.g., latency and throughput
- For an ML system, care about both its operational metrics and its ML performance metrics
- For example, consider an English-French machine translation system
  - Operational expectation might be that, given an English sentence, the system returns a French translation within a one-second latency
  - ML performance expectation is that the returned translation is an accurate translation of the original English sentence 99% of the time

ML system failure types

- Operational expectation violations are easier to detect,
  - as usually accompanied by an operational breakage
  - such as a timeout, a 404 error on a webpage, an out-of-memory error, or a segmentation fault
- However, ML performance expectation violations are harder to detect
  - as doing so requires measuring and monitoring the performance of ML models in production
- In the English-French machine translation system, detecting whether the returned translations are correct 99% of the time is difficult
  - if don't know what the correct translations are supposed to be
- To effectively detect and fix ML system failures in production,
  - it's useful to understand why a model, after proving to work well during development, would fail in production
- Two types of failures: software system failures and ML-specific failures

Software System Failures

failures that would have happened to non-ML systems

- Dependency failure
  - A software package or a codebase that system depends on breaks, which leads system to break
  - common when the dependency is maintained by a third party
- Deployment failure
  - failures caused by deployment errors,
  - such as when accidentally deploy the binaries of an older version of model instead of the current version,
  - or when systems don't have the right permissions to read or write certain files
- Hardware failures
  - When the hardware that is used to deploy model, such as CPUs or GPUs, doesn't behave the way it should
  - For example, the CPUs used might overheat and break down
- Downtime or crashing
  - If a component of system runs from a server somewhere, such as AWS or a hosted service, and that server is down, system will also be down

Software System Failures(2)

- Addressing software system failures requires not ML skills, but traditional software engineering skills!
- Because of the importance of traditional software engineering skills in deploying ML systems,
  - ML engineering is mostly engineering, not ML!
- The reasons for the prevalence of software system failures:
  - ML adoption in the industry is still nascent,
  - tooling around ML production is limited
  - and best practices are not yet well developed or standardized
- However, as tooling's and best practices for ML production mature,
  - there are reasons to believe that the proportion of software system failures will decrease
  - and the proportion of ML-specific failures will increase

ML-Specific Failures

failures specific to ML systems

- Examples include
  - data collection and processing problems,
  - poor hyper parameters,
  - changes in the training pipeline not correctly replicated in the inference pipeline and vice versa,
  - data distribution shifts that cause a model's performance to deteriorate over time,
  - edge cases,
  - and degenerate feedback loops
- Even though they account for a small portion of failures, they can be more dangerous than non-ML failures
  - as they're hard to detect and fix, and they can prevent ML systems from being used altogether

Production data differing from training data

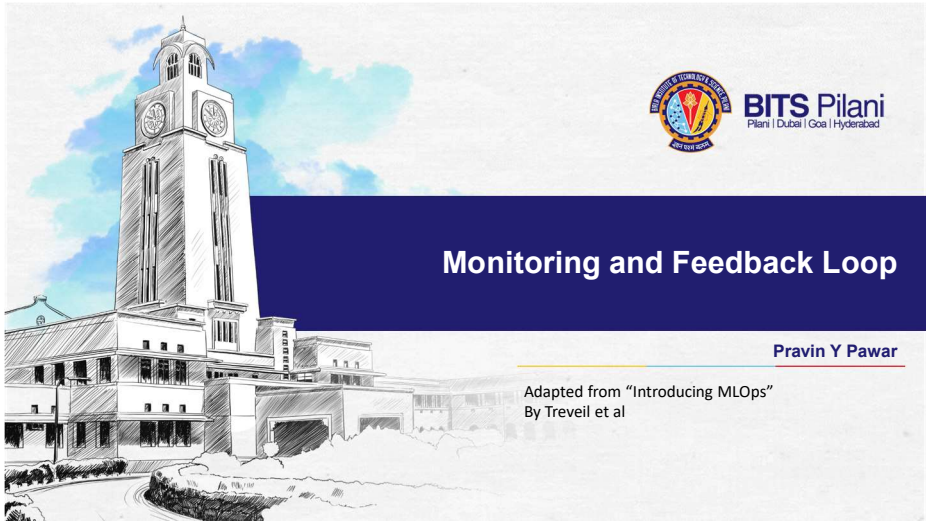
- ML model learns from the training data, means that the model learns the underlying distribution of the training data
  - with the goal of leveraging this learned distribution to generate accurate predictions for unseen data
  - when the model is able to generate accurate predictions for unseen data - model "generalizes to unseen data"
- The assumption
  - unseen data comes from a stationary distribution that is the same as the training data distribution
- This assumption is incorrect in most cases for two reasons!
- First, the underlying distribution of the real-world data is unlikely to be the same as the underlying distribution of the training data
  - Real-world data is multifaceted and, in many cases, virtually infinite,
  - whereas training data is finite and constrained by the time, compute, and human resources available during the dataset creation and processing
  - divergence leads to a common failure mode known as the train-serving skew: a model that does great in development but performs poorly when deployed
- Second, the real world isn't stationary. Things change. Data distributions shift.

Edge cases

- Edge cases are the data samples so extreme that they cause the model to make catastrophic mistakes
- An ML model that performs well on most cases but fails on a small number of cases
  - might not be usable if these failures cause catastrophic consequences
  - major self-driving car companies are focusing on making their systems work on edge cases
- Also true for any safety-critical application such as medical diagnosis, traffic control, e-discovery, etc.
- Can also be true for non-safety-critical applications
  - Imagine a customer service chatbot that gives reasonable responses to most of the requests,
  - but sometimes, it spits out outrageously racist or sexist content
  - will be a brand risk for any company that wants to use it, thus rendering it unusable

Degenerate feedback loops

- Feedback loop as the time it takes from when a prediction is shown until the time feedback on the prediction is provided
  - feedback can be used to
    - extract natural labels to evaluate the model's performance
    - train the next iteration of the model
- A degenerate feedback loop can happen when the predictions themselves influence the feedback,
  - which, in turn, influences the next iteration of the model
- A degenerate feedback loop is created when a system's outputs are used to generate the system's future inputs,
  - which, in turn, influence the system's future outputs
- Degenerate feedback loops are especially common in tasks with natural labels from users,
  - such as recommender systems and ads click-through-rate prediction
- Imagine building a system to recommend to users songs that they might like
  - songs that are ranked high by the system are shown first to users
  - because they are shown first, users click on them more,
  - which makes the system more confident that these recommendations are good



Model monitoring

Model monitoring is a crucial step in the ML model life cycle and a critical piece of MLOps

- When a machine learning model is deployed in production
  - it can start degrading in quality fast—and without warning—until it's too late
- Machine learning models need to be monitored at two levels:
- At the resource level, including ensuring the model is running correctly in the production environment.
  - Key questions include:
    - Is the system alive?
    - Are the CPU, RAM, network usage, and disk space as expected?
    - Are requests being processed at the expected rate?
- At the performance level, meaning monitoring the pertinence of the model over time
  - Key questions include:
    - Is the model still an accurate representation of the pattern of new incoming data?
    - Is it performing as well as it did during the design phase?

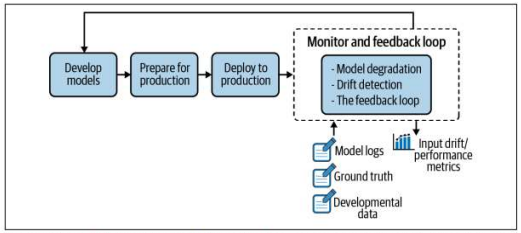
Model performance monitoring

- The first level is a traditional DevOps topic
- The latter is more complicated. Why?
  - Because how well a model performs is a reflection of the data used to train it
    - in particular, how representative that training data is of the live request data
- As the world is constantly changing,
  - a static model cannot catch up with new patterns that are emerging and evolving without a constant source of new data
- Model performance monitoring attempts to track this degradation,
  - At an appropriate time, it will also trigger the retraining of the model with more representative data

Model Retraining

- Critical for organizations to have a clear idea of deployed models' drift and accuracy
  - by setting up a process that allows for easy monitoring and notifications
- Ideal scenario would be a pipeline that automatically triggers checks for degradation of model performance
  - goal of notifications is not necessarily to kick off an automated process of retraining, validation, and deployment
  - but to alert the data scientist of the change; to diagnose the issue and evaluate the next course of action
- Critical that as part of MLOps and the ML model life cycle,
  - data scientists and their managers and the organization as a whole understand model degradation
- Every deployed model should come with monitoring metrics and corresponding warning thresholds
  - to detect meaningful business performance drops as quickly as possible

Monitoring and feedback loop



Monitoring and feedback loop highlighted in the larger context of the ML project life cycle

Model Degradation

- Once a machine learning model is trained and deployed in production
- Two approaches to monitor its performance degradation:
  - ground truth evaluation
  - input drift detection

Ground Truth Evaluation

Defined

- Ground truth retraining requires waiting for the label event
  - In a fraud detection model, the ground truth would be whether or not a specific transaction was actually fraudulent
  - For a recommendation engine, it would be whether or not the customer clicked on—or ultimately bought—one of the recommended products
- With the new ground truth collected,
  - next step is to compute the performance of model based on ground truth
  - compare it with registered metrics in the training phase
- When the difference surpasses a threshold, the model can be deemed as outdated, and it should be retrained
- The metrics to be monitored can be of two varieties:
  - Statistical metrics like accuracy, ROC AUC, log loss, etc.
    - domain agnostic
    - drawback is that the drop may be statistically significant without having any noticeable impact
    - cost of retraining and risk associated with a redeployment may be higher than expected benefits
  - Business metrics, like cost-benefit assessment such as the credit scoring
    - far more interesting because they ordinarily have a monetary value
    - enabling subject matter experts to better handle the cost-benefit trade-off of the retraining decision

Ground Truth Evaluation(2)

Challenges

- Ground truth is not always immediately, or even imminently, available
  - For some types of models, teams need to wait months (or longer) for ground truth labels to be available,
  - which can mean significant economic loss if the model is degrading quickly
  - Deploying a model for which the drift is faster than the lag is risky
- Ground truth and prediction are decoupled
  - To compute the performance of the deployed model on new data, it's necessary to be able to match ground truth with the corresponding observation
  - In many production environments, this is a challenging task because these two pieces of information are generated and stored in different systems and at different timestamps
- Ground truth is only partially available
  - In some situations, it is extremely expensive to retrieve the ground truth for all the observations,
  - which means choosing which samples to label and thus inadvertently introducing bias into the system

Input Drift

- Mathematically speaking, the samples of each dataset cannot be assumed to be drawn from the same distribution
  - i.e., they are not "identically distributed"
- Another mathematical property is necessary to ensure that ML algorithms perform as expected: independence
  - property is broken if samples are duplicated in the dataset or if it is possible to forecast the "next" sample given the previous one
- If train the algorithm on the first dataset and then deploy it on the second one
  - The resulting distribution shift is called a drift
- In wine quality prediction exercise,
- Called a feature drift
  - if the alcohol level is one of the features used by the ML model
  - or if the alcohol level is correlated with other features used by the model
- Called as a concept drift if it is not

Drift Detection in Practice

- To be able to react in a timely manner, model behavior should be monitored solely based on the feature values of the incoming data,
  - without waiting for the ground truth to be available
- The logic is that if the data distribution (e.g., mean, sd, correlations between features) diverges
  - between the training and testing phases on one side and the development phase on the other,
  - a strong signal that the model's performance won't be the same!
- Not the perfect mitigation measure, as retraining on the drifted dataset will not be an option,
  - but it can be part of mitigation measures (e.g., reverting to a simpler model, reweighting)

Causes of Data Drift

Two frequent root causes of data drift

- Sample selection bias
  - the training sample is not representative of the population
  - often stems from the data collection pipeline itself
  - For instance, building a model to assess the effectiveness of a discount program will be biased
    - if the best discounts are proposed for the best clients
- Non-stationary environment
  - where training data collected from the source population does not represent the target population
  - Often happens for time dependent tasks — such as forecasting with strong seasonality effects,
    - where learning a model over a given month won't generalize to another month

Input Drift Detection Techniques

Choice depends on the expected level of interpretability

- Two approaches
  - Univariate statistical tests
  - Domain classifier
- Should prefer univariate statistical tests
  - Organizations that need proven and explainable methods
- Domain classifier approach may be a good option
  - if complex drift involving several features simultaneously is expected
  - if the data scientists want to reuse what they already know and assuming the organization doesn't dread the black box effect

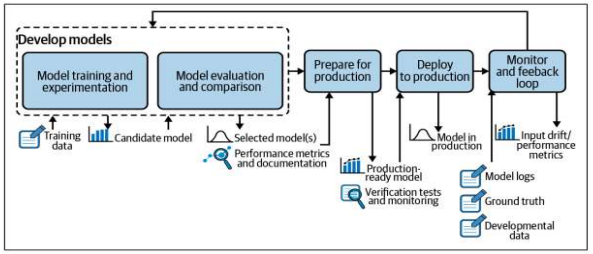
Univariate statistical tests

- Requires applying a statistical test on data from the source distribution and the target distribution for each feature
  - A warning will be raised when the results of those tests are significant
- Basic approaches rely on two tests:
  - For continuous features, the Kolmogorov-Smirnov test is a nonparametric hypothesis test that is used to check whether two samples come from the same distribution
  - For categorical features, the Chi-squared test is a practical choice that checks whether the observed frequencies for a categorical feature in the target datamatch the expected frequencies seen from the source data
- The main advantage of p-values is that they help detect drift as quickly as possible
- The main drawback is that they do not quantify the level of the effect
- If development datasets are very large, it is necessary to complement p-values with business-significant metrics
  - For example, on a sufficiently large dataset, the average age may have significantly drifted from a statistical perspective,
  - but if the drift is only a few months, this is probably an insignificant value for many business use cases

Domain classifier

- Data scientists train a model that tries to discriminate between the original dataset (input features and, optionally, predicted target) and the development dataset
  - stack the two datasets and train a classifier that aims at predicting the data's origin
- The performance of the model (its accuracy, for example) can then be considered as a metric for the drift level
  - If this model is successful in its task, and thus has a high drift score,
  - implies that the data used at training time and the new data can be distinguished
  - fair to say that the new data has drifted
- To gain more insights, in particular to identify the features that are responsible for the drift
  - one can use the feature importance of the trained model

Continuous delivery for end-to-end machine learning process



Continuous delivery for end-to-end machine learning process

The Feedback Loop

- All effective machine learning projects implement a form of data feedback loop
  - information from the production environment flows back to the model prototyping environment for further improvement
- Continuous delivery for end-to-end machine learning process
  - Data collected in the monitoring and feedback loop is sent to the model development phase
  - From there, the system analyzes whether the model is working as expected
  - If it is, no action is required
  - If the model's performance is degrading, an update will be triggered, either automatically or manually by the data scientist
- In practice, usually means either retraining the model with new labeled data or developing a new model with additional features

The Feedback Loop(2)

- Goal of retraining is to be able to capture the emerging patterns and make sure that the business is not negatively impacted
- This infrastructure is comprised of three main components, which are critical to robust MLOps capabilities:
  - A logging system that collects data from several production servers
  - A model evaluation store that does versioning and evaluation between different model versions
  - An online system that does model comparison on production environments,
    - either with the shadow scoring (champion/challenger) setup or with A/B testing

Logging

- Monitoring a live system means collecting and aggregating data about its states
- Nowadays, as production infrastructures are getting more and more complex,
  - with several models deployed simultaneously across several servers,
  - an effective logging system is more important than ever!
- Data from these environments needs to be centralized to be analyzed and monitored,
  - either automatically or manually
  - will enable continuous improvement of the ML system
- An event log of a machine learning system is a record with a timestamp and information such as
  - Model metadata - Identification of the model and the version
  - Model inputs - Feature values of new observations
  - Model outputs - Predictions made by the model that
  - System action - an action taken based on model prediction
  - Model explanation - an explanation of prediction (i.e., which features have the most influence on the prediction

Model Evaluation

- If model performance is degrading, after review, data scientists decide to improve the model by retraining it,
  - With several trained candidate models, the next step is to compare them with the deployed model
  - means evaluating all the models (the candidates as well as the deployed model) on the same dataset
- If one of the candidate models outperforms the deployed model, there are two ways to proceed:
  - either update the model on the production environment
  - or move to an online evaluation via a champion/challenger or A/B testing setup
- In a nutshell, this is the notion of model store
- A structure that allows data scientists to:
  - Compare multiple, newly trained model versions against existing deployed versions
  - Compare completely new models against versions of other models on labeled data
  - Track model performance over time

Model evaluation store

- Formally, the model evaluation store serves as a structure that centralizes the data related to model life cycle to allow comparisons
- Two main tasks of a model evaluation store are:
  - Versioning the evolution of a logical model through time
    - Each logged version of the logical model must come with all the essential information concerning its training phase, including:
      - The list of features used
      - The preprocessing techniques that are applied to each feature
      - The algorithm used, along with the chosen hyperparameters
      - The training dataset
      - The test dataset used to evaluate the trained model (this is necessary for the version comparison phase)
      - Evaluation metrics
  - Comparing the performance between different versions of a logical model

Online Evaluation

- Online evaluation of models in production is critical from a business perspective,
  - but can be challenging from a technical perspective
- Two main modes of online evaluation:
  - Champion/challenger (otherwise known as shadow testing)
    - the candidate model shadows the deployed model and scores the same live requests
  - A/B testing
    - the candidate model scores a portion of the live requests and the deployed model scores the others
- Both cases require ground truth
  - evaluation will necessarily take longer than the lag between prediction and ground truth obtention
- Whenever shadow testing is possible, it should be used over A/B testing
  - because it is far simpler to understand and set up, and it detects differences more quickly

The basics

- Monitoring, at the most basic level, provides data about how your systems are performing
  - data is made storable, accessible, and displayable in some reasonable way
- Observability is an attribute of software, meaning that when correctly written, the emitted monitoring data
  - usually extended or expanded in some way, with labeling or tagging
  - can be used to correctly infer behavior of system
- Obviously, monitoring is hugely important in and of itself,
  - but an offshoot of monitoring is absolutely crucial: alerting
  - A useful simplification is that when things go wrong, humans are alerted to fix them
  - defining the conditions for "things going wrong," and being able to reliably notify the responsible folks that

What Does It Look Like?

- To do monitoring,
  - must have a monitoring system
  - as well as systems to be monitored (called the target systems)
- Today, target systems
  - emit metrics a series, typically of numbers, with an identifying name
  - which are then collected by the monitoring system
  - and transformed in various ways,
    - often via aggregation (producing a sum or a rate across multiple instances or machines)
    - or decoration (adding, say, event details onto the same data)
  - aggregated metrics are used for system analysis, debugging, and the alerting

Example

Web Server

- Web server emits a metric of the total number of requests it received
  - The monitoring system will obtain these metrics, usually via push or pull,
    - which refers to whether the metrics get pulled from the target systems or get pushed from them
  - These metrics are then collated, stored,
  - and perhaps processed in some way, generally as a time series
- Different monitoring systems will make different choices about how to receive, store, process,
  - but the data is generally queryable and often there's a graphical way to plot the monitoring data
  - to take advantage of our visual comparison hardware (eyes, retinas, optic nerves, and so on) to figure out what's actually happening

Problems with ML Production Monitoring

- ML model development is still in its infancy!
  - tools are immature, conceptual frameworks are underdeveloped
  - discipline is in short supply, as everyone scrambles to get some kind of model—any kind of model!
- The pressure to ship is real and has real effects!
- In particular, model development,
  - which is inherently hard because it involves reconciling a wide array of conflicting concerns
  - gets harder because that urgency forces developers and data science folks to focus on those hard problems
  - ignore the wider picture
- That wider picture often involves questions around monitoring and observability!

Difficulties of Development Versus Serving

The first problem is that effectively simulating production in development is extremely hard

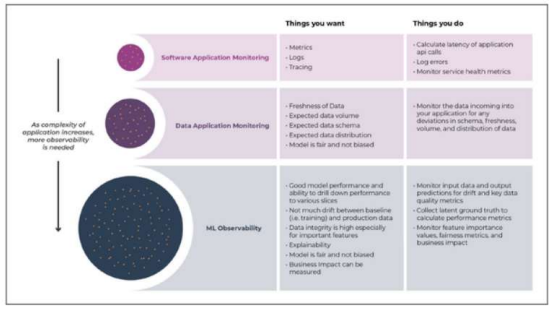
- Even if separate environments dedicated to that task (like test, staging, and so on.)
- Primarily Because of
- the wide variety of possible serving architectures
  - model pools, shared libraries, edge devices, etc., with the associated infrastructure running on)
- the invocation of the predictions
  - in development often invoke prediction methods directly or with a relatively small amount of code between developer and the model for velocity reasons
  - Running in production also generally means don't have the ability to manipulate input, logging level, processing, and so on
    - leading to huge difficulties in debugging, reproducing problematic configurations, etc.
- data in testing is not necessarily distributed like the data the model encounters in production
  - as always for ML, data distribution really matters

Difficulties of Development Versus Serving(2)

The second problem is about mature practices

- In conventional software delivery,
  - the industry has a good handle on work practices to improve throughput, reliability, and developmental velocity
  - most important is grouped concepts of continuous integration / continuous deployment (CI/CD), unit tests, small changes
- Unfortunately, today w\missing this equivalent of CI/CD for model development
  - not yet converged onto a good set of (telemetry-related, or otherwise) tools for model training and validation
- Expect this will improve over time as
  - existing tools (such as MLflow and Kubeflow) gain traction
  - vendors incorporate more of these concerns into their platforms
  - and the mindset of holistic, or whole-lifecycle monitoring gains more acceptance

Observability layers and system requirements

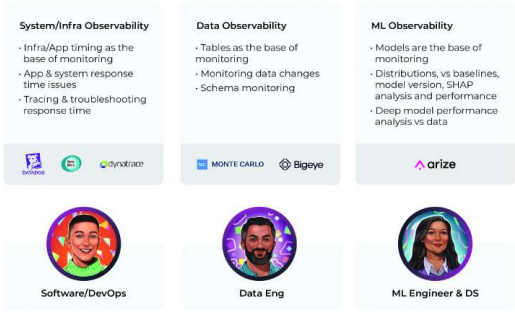


Observability layers and system requirements

Reasons for Continual ML Observability—in Production

- Observability data from models is absolutely fundamental to business
  - both tactical operations and strategic insights
- One example - connection between latency and online sales
  - In 2008, Amazon discovered that each additional 100 ms of latency lost 1% of sales,and also the converse
  - Similar results have been confirmed by Akamai Technologies, Google, Zalando, and others
- Without observability, there would be no way to have discovered this effect,
  - and certainly no way to know for sure that either making it better or worse!
- Ultimately, observability data is business outcome data
  - In the era of ML, this happily allows not just to detect and respond to outages,
  - but also to understand incredibly important things that are happening to business

ML observability in context

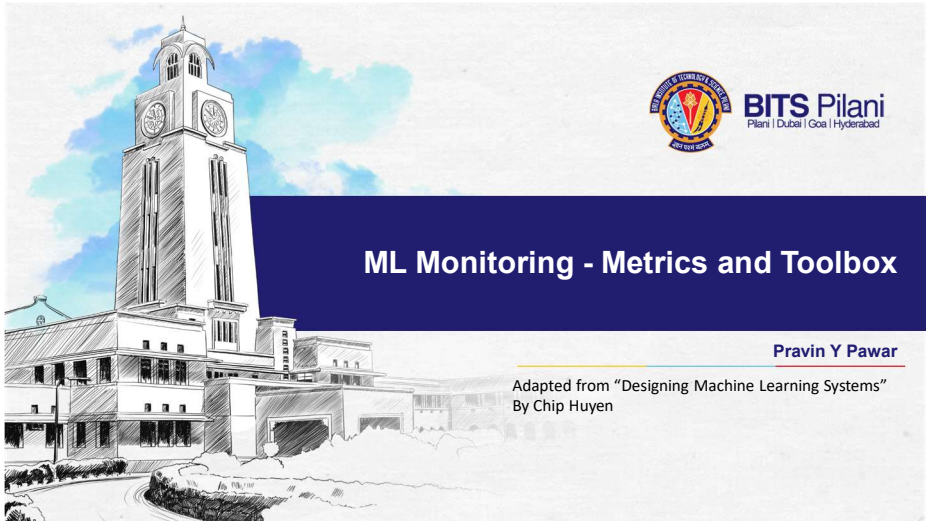


arize

Key Components of Observability

System vs Machine Learning

System Observability	ML Observability
Logs • Records of an event that happened within an application. • Typically not mutable by an event ID. • Searchable by tags and unstructured indexes.	**Inference Store** • Records of ML prediction events that are logged from the model. • Raw prediction events that hold granular context about the model's predictions. • Mutable by prediction ID and dataset.
Metrics • Measured values of system performance. • Metrics comprise a set of attributes (i.e. value, label, and timestamp) that convey information about SLAs, SLOs, and SLIs.	**Model Metrics** - Calculated metrics on the prediction events. - Provides ways to determine model health over time – this includes drift, performance, and data quality metrics. - Metrics can be monitored. - Metrics can be aggregate or slice-level.
Tracing • Provides context for the other components of observability (logs, metrics). • Follows the entire lifecycle of a request or action across distributed systems.	**ML Performance Tracing** - ML performance tracing is the methodology for pinpointing the source of a model performance problem. - Involves mapping back to the data that caused the problem. - Necessarily a distinct discipline because logs and metrics are rarely helpful for debugging model performance.

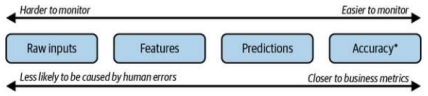


ML Monitoring

- As the industry realized that many things can go wrong with an ML system,
  - many companies started investing in monitoring and observability for their ML systems in production
- Monitoring refers to the act of tracking, measuring, and logging different metrics that can help us determine when something goes wrong
- Observability means setting up our system in a way that gives us visibility into system to help us investigate what went wrong
- Monitoring is all about metrics!
  - Because ML systems are software systems, the first class of metrics need to monitor are the operational metrics
  - designed to convey the health of systems
- Generally divided into three levels:
  - the network the system is run on,
  - the machine the system is run on,
  - and the application that the system runs
- Examples
  - latency; throughput; the number of prediction requests model receives in the last minute, hour, day;
  - the percentage of requests that return with a 2xx code;
  - CPU/GPU utilization; memory utilization; etc.

ML-Specific Metrics

- Within ML-specific metrics, there are generally four artifacts to monitor:
  - a model's accuracy-related metrics,
  - predictions,
  - features,
  - and raw inputs
- Artifacts generated at four different stages of an ML system pipeline
  - The deeper into the pipeline an artifact is, the more transformations it has gone through,
    - which makes a change in that artifact more likely to be caused by errors in one of those transformations
  - However, the more transformations an artifact has gone through, the more structured it's become
    - closer it is to the metrics actually care about, which makes it easier to monitor



Monitoring accuracy-related metrics

- Accuracy-related metrics are the most direct metrics to help you decide whether a model's performance has degraded
- If system receives any type of user feedback for the predictions it makes
  - click, hide, purchase, upvote, downvote, favorite, bookmark, share, etc.
  - should definitely log and track it
- Some feedback can be used to infer natural labels,
  - which can then be used to calculate model's accuracy-related metrics
- Feedback can be used to detect changes in ML model's performance
  - For example, when building a system to recommend to users what videos to watch next on YouTube,
  - want to track not only whether the users click on a recommended video (click-through rate),
  - but also the duration of time users spend on that video and whether they complete watching it (completion rate)
  - If, over time, the clickthrough rate remains the same but the completion rate drops, it might mean that recommender system is getting worse
- Possible to engineer system to collect users' feedback
  - For example, Google Translate has the option for users to upvote or downvote a translation

Monitoring predictions

- Prediction is the most common artifact to monitor
  - If it's a regression task, each prediction is a continuous value (e.g., the predicted price of a house)
  - If it's a classification task, each prediction is a discrete value corresponding to the predicted category
- Each prediction is usually just a number (low dimension), predictions are easy to visualize
  - summary statistics are straightforward to compute and interpret
- Can monitor predictions for distribution shifts as they are low dimensional
  - Easier to compute two-sample tests to detect whether the prediction distribution has shifted
  - Prediction distribution shifts are also a proxy for input distribution shifts
- Can also monitor predictions for anything odd happening
  - such as predicting an unusual number of False in a row

Monitoring features

- ML monitoring solutions in the industry focus on tracking changes in features, both
- the features that a model uses as inputs
  - the intermediate transformations from raw inputs into final features
- Feature monitoring is appealing because
  - compared to raw input data, features are well structured following a predefined schema
- The first step of feature monitoring is feature validation:
  - ensuring that features follow an expected schema
- Things can be check for a given feature:
  - If the min, max, or median values of a feature are within an acceptable range
  - If the values of a feature satisfy a regular expression format
  - If all the values of a feature belong to a predefined set
  - If the values of a feature are always greater than the values of another feature
- Many open source libraries that help in basic feature validation,
  - Two most common are Great Expectations and Deequ, which is by AWS

Monitoring features(2)

Four major concerns when doing feature monitoring

- A company might have hundreds of models in production, and each model uses hundreds, if not thousands, of features.
  - Even something as simple as computing summary statistics for all these features every hour can be expensive,
  - not only in terms of compute required but also memory used
  - Tracking, i.e., constantly computing, too many metrics can also slow down system
  - increase both the latency that users experience
- While tracking features is useful for debugging purposes, it's not very useful for detecting model performance degradation
  - In practice, an individual feature's minor changes might not harm the model's performance at all
  - Feature distributions shift all the time, and most of these changes are benign
  - If want to be alerted whenever a feature seems to have drifted, might soon be overwhelmed by alerts
  - realize that most of these alerts are false positives -"alert fatigue"
- Feature extraction is often done in multiple steps (such as filling missing values and standardization),
  - using multiple libraries (such as pandas, Spark),
  - on multiple services (such as BigQuery or Snowflake).
- Even if its detected a harmful change in a feature, it might be impossible to detect whether this change is
  - caused by a change in the underlying input distribution or whether it's caused by an error in one of the multiple processing steps
- The schema that features follow can change over time.
  - If don't have a way to version schemas and map each of features to its expected schema,
  - the cause of the reported alert might be due to the mismatched schema rather than a change in the data

Monitoring raw inputs

- A change in the features might be caused by problems in processing steps and not by changes in data
  - can monitor the raw inputs before they are processed
  - not be easier to monitor, as it can come from multiple sources in different formats, following multiple structures
- The way many ML workflows are set up today also makes it impossible for ML engineers to get direct access to raw input data,
  - as the raw input data is often managed by a data platform team who processes and moves the data to a location like a data warehouse,
- ML engineers can only query for data from that data warehouse where the data is already partially processed
- Monitoring raw inputs is often a responsibility of the data platform team, not the data science or ML team

Monitoring Toolbox

- Measuring, tracking, and interpreting metrics for complex systems is a nontrivial task
  - engineers rely on a set of tools to help them do so
- Common for the industry to herald metrics, logs, and traces as the three pillars of monitoring
- Seem to be generated from the perspective of people who develop monitoring systems:
  - traces are a form of logs and metrics can be computed from logs
- Focus on the set of tools from the perspective of users of the monitoring systems:
  - logs,
  - dashboards,
  - alerts

Monitoring Toolbox(2)

Logs

- Traditional software systems rely on logs to record events produced at runtime
  - An event is anything that can be of interest to the system developers,
  - either at the time the event happens or later for debugging and analysis purposes
- Examples of events
  - a container starts,
  - the amount of memory it takes,
  - when a function is called,
  - when that function finishes running,
  - the other functions that this function calls,
  - the input and output of that function,
  - log crashes, stack traces, error codes, and more.
- The number of logs can grow very large very quickly.
  - need to query your logs for the sequence of events that caused it, a process that can feel like searching for a needle in a haystack
- Because logs have grown so large and so difficult to manage,
  - there have been many tools developed to help companies manage and analyze logs
  - The log management market is estimated to be worth USD 2.3 billion in 2021, and it's expected to grow to USD 4.1 billion by 2026

Monitoring Toolbox(3)

Dashboards

- A picture is worth a thousand words!
  - A series of numbers might mean nothing, but visualizing them on a graph might reveal the relationships among these numbers
  - Dashboards to visualize metrics are critical for monitoring
- Another use of dashboards is to make monitoring accessible to non-engineers.
  - Monitoring isn't just for the developers of a system, but also for non-engineering stakeholders
    - including product managers and business developers
- Dashboard rot
  - Excessive metrics on a dashboard can also be counterproductive
    - important to pick the right metrics
    - abstract out lower-level metrics to compute higher-level signals that make better sense for specific tasks

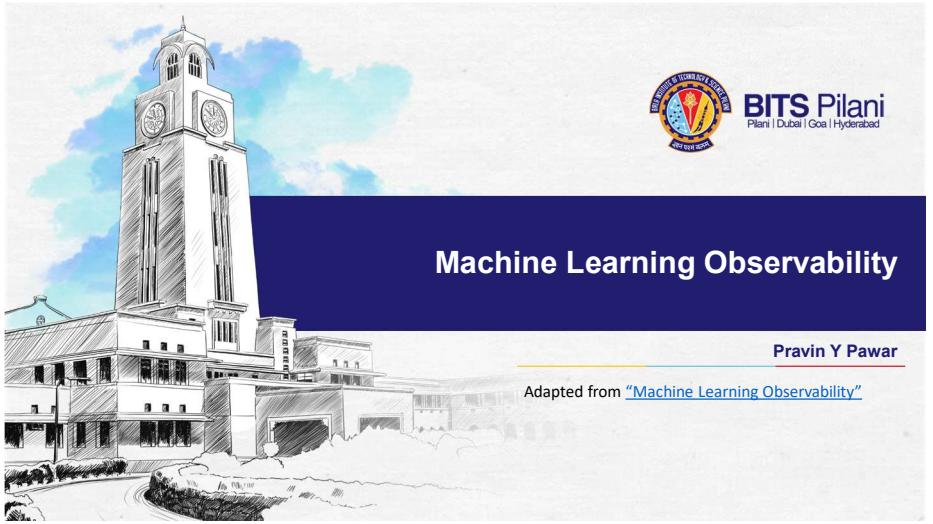
Monitoring Toolbox(4)

Alerts

- When monitoring system detects something suspicious, it's necessary to alert the right people about it

- An alert consists of the following three components:
- An alert policy
  - describes the condition for an alert
  - might want to create an alert when a metric breaches a threshold, optionally over a certain duration
- Notification channels
  - describe who is to be notified when the condition is met
- A description of the alert
  - helps the alerted person understand what's going on with a detailed description
  - often necessary to make the alert actionable
  - by providing mitigation instructions or a runbook, a compilation of routine procedures and operations that might help with handling the alert

- Alert fatigue is a real phenomenon!
  - can be demoralizing—nobody likes to be awakened in the middle of the night for something outside of their responsibilities
  - can be dangerous —being exposed to trivial alerts can desensitize people to critical alerts
  - important to set meaningful conditions so that only critical alerts are sent out

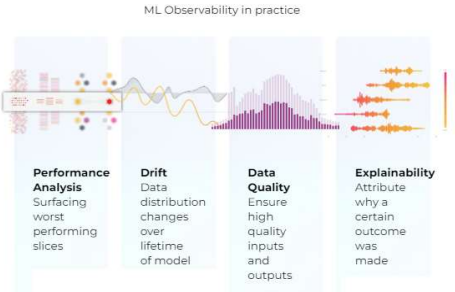


What is ML Observability?

- ML Observability is a tool used to monitor, troubleshoot, and explain machine learning models
  - as they move from research to production environments

- An effective observability tool should not only automatically surface issues,
  - but drill down to the root cause of your ML problems and act as a guardrail for models in production

4 Pillars of ML Observability



Performance Analysis

- ML observability enables fast actionable performance information on models deployed in production
- While performance analysis techniques vary on a case-by-case basis depending on model type and its use case in the real world,
  - common metrics include: Accuracy, Recall, F-1, MAE, RMSE, and Precision
- Performance analysis in an ML observability system ensures that performance has not degraded drastically from when it was trained or when it was initially promoted to production.

Drift

- ML observability encompasses drift to monitor for a change in distribution over time,
  - measured for model inputs, outputs, and actuals of a model
- Measure drift to identify if models have grown stale, have data quality issues, or if there are adversarial inputs in model
- Detecting drift in models will help protect models from performance degradation and allow to better understand how to begin resolution

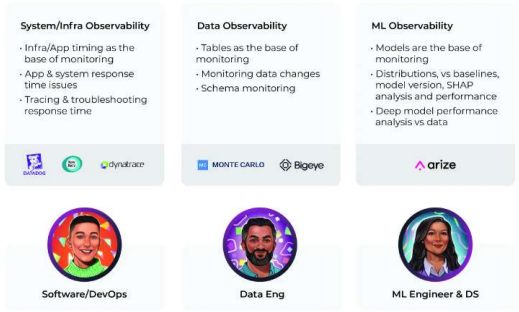
Data Quality

- Data quality checks in an ML observability system identify hard failures
  - within data pipelines between training and production
  - that can negatively impact a model's end performance
- Data quality includes
  - monitoring for cardinality shifts,
  - missing data,
  - data type mismatch,
  - out-of-range,
  - and more to better gauge model performance issues and ease RCA

Explainability

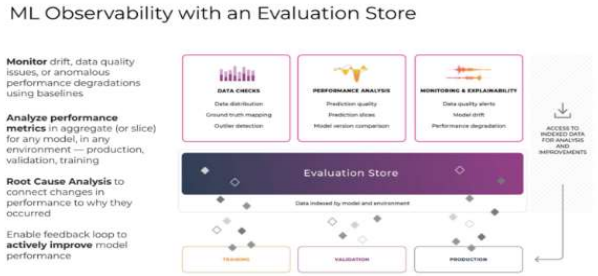
- Explainability in ML observability uncovers feature importance
  - across training, validation, and production environments
  - which provides the ability to introspect and understand why a model made a particular prediction
- Explainability is commonly achieved by
  - calculating metrics such as SHAP and LIME
  - to build confidence and continuously improve machine-learned models

ML observability in context



arize

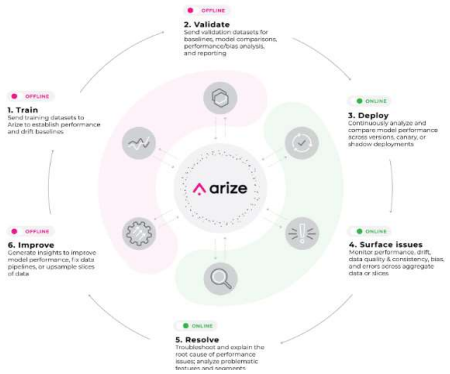
How Can I Achieve ML Observability?



arize

ML Observability with Arize

- ML Observability is the practice of obtaining a deep understanding into model's data and performance across its lifecycle
  - Observability doesn't just stop at surfacing a red or green light,
  - but enables ML practitioners to root cause/explain why a model is behaving a certain way in order to improve it



Continual Learning

- How do adapt models to data distribution shifts?
  - The answer is by continually updating our ML models
- Continual learning is largely an infrastructural problem
  - Setting up infrastructure allows to update models as frequently as required
- The goal of continual learning is to safely and efficiently automate the update
  - allows to design an ML system that is maintainable and adaptable to changing environments

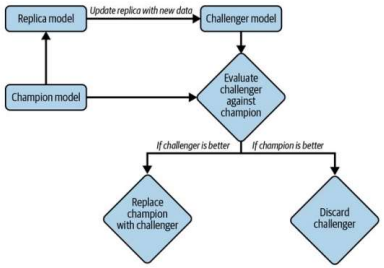
What is continual learning?

- Many people think of the training paradigm where a model updates itself with every incoming sample in production
  - people imagine updating models very frequently, such as every 5 or 10 minutes
- Very few companies actually do that!
  - makes model susceptible to catastrophic forgetting - the tendency of a neural network to completely and abruptly forget
    previously learned information upon learning new information
  - make training more expensive - most hardware backends today were designed for batch processing, so processing only
    one sample at a time causes a huge waste of compute power and is unable to exploit data parallelism
- Most companies don't need to update their models that frequently because of two reasons
  - First, they don't have enough traffic (i.e., enough new data) for that retraining schedule to make sense
  - Second, their models don't decay that fast
    - If changing retraining schedule from a week to a day gives no return and causes more overhead, there's no need to do it.

A simplification of how continual learning work in production

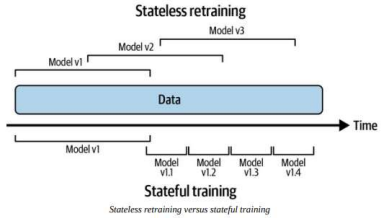
Champion-Challenger

- The updated model shouldn't be deployed until it's been evaluated
  - means that shouldn't make changes to the existing model directly
- Instead,
  - should create a replica of the existing model
  - update this replica on new data
  - only replace the existing model with the updated replica if
  - the updated replica proves to be better
- The existing model - champion model
- The updated replica - the challenger
- An oversimplification of the process for the sake of understanding!
  - In reality, a company might have multiple challengers at the same time



Stateless Retraining Versus Stateful Training

- Continual learning isn't about the retraining frequency,
  - but the manner in which the model is retrained
- Most companies do stateless retraining — the model is trained from scratch each time
- Some follows stateful training — the model continues training or new data
  - also known as fine-tuning or incremental learning
- Stateful training allows to update model with less data
- Training a model from scratch tends to require a lot more data
- One beautiful property that is often overlooked is that with stateful training,
  - it might be possible to avoid storing data altogether



Stateless retraining versus stateful training

Stateless Retraining Versus Stateful Training(2)

- The companies that have most successfully used stateful training
  - also occasionally train their model from scratch on a large amount of data to calibrate it
- Once infrastructure is set up to allow both stateless retraining and stateful training,
  - the training frequency is just a knob to twist
  - can update models once an hour, once a day, or whenever a distribution shift is detected
- Continual learning is about setting up infrastructure in a way that
  - allows a data scientist or ML engineer,
  - to update models whenever it is needed,
  - whether from scratch or fine-tuning,
  - to deploy this update quickly

Model iteration vs Data iteration

- Stateful training sounds cool, but how does this work if needs to add a new feature or another layer to model?
- Must differentiate two types of model updates:
  - Model iteration - A new feature is added to an existing model or the model architecture is changed
  - Data iteration - Model architecture and features remain the same, but refresh this model with new data.
- As of today, stateful training is mostly applied for data iteration!
- Changing model architecture or adding a new feature still requires training the resulting model from scratch
  - research shows that it might be possible to bypass training from scratch for model iteration
  - by using techniques such as knowledge transfer (Google, 2015) and model surgery (OpenAI, 2019)

Why Continual Learning?

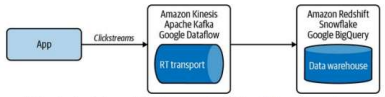
Why would you need the ability to update your models as fast as you want?

- The first use case of continual learning is to combat data distribution shifts,
  - especially when the shifts happen suddenly
- Another use case of continual learning is to adapt to rare events
- Can help overcome is the continuous cold start problem
- "Why continual learning?" should be rephrased as "why not continual learning?"
  - Continual learning is a superset of batch learning - allows to do everything traditional batch learning can do
  - If continual learning takes the same effort to set up and costs the same to do as batch learning,
    - there's no reason not to do continual learning!
- MLOps tooling for continual learning is maturing, which means, one day not too far in the future,
  - it might be as easy to set up continual learning as batch learning

Continual Learning Challenges

Fresh data access challenge - Data Sources

- If want to update model every hour, need new data every hour!
- Data Sources
  - Currently, many companies pull new training data from their data warehouses
    - The speed at which data can be pulled from data warehouses depends on the speed at which this data is deposited into data warehouses
    - The speed can be slow, especially if data comes from multiple sources
  - An alternative is to allow pull data before it's deposited into data warehouses,
    - e.g., directly from real-time transports such as Kafka and Kinesis that transport data from applications to data warehouses



Pulling data directly from real-time transports, before it's deposited into data warehouses, can allow you to access fresher data

Continual Learning Challenges(2)

Fresh data access challenge - Labelling

- If model needs labeled data to update, data will need to be labeled as well!
  - In many applications, the speed at which a model can be updated is bottlenecked by the speed at which data is labeled
- The best candidates for continual learning are tasks where can get natural labels with short feedback loops
  - dynamic pricing, estimating time of arrival, stock price prediction,
  - ads click-through prediction, and recommender systems
- If model's speed iteration is bottlenecked by labeling speed,
  - possible to speed up the labeling process by leveraging programmatic labeling tools like Snorkel to generate fast labels
  - possible to leverage crowdsourced labels to quickly annotate fresh data
- Given that tooling around streaming is still nascent,
  - architecting an efficient streaming-first infrastructure for accessing fresh data and extracting fast labels from real-time transports can be engineering-intensive and costly

Continual Learning Challenges(3)

Evaluation challenge

- ML systems make catastrophic failures in production,
  - from millions of minorities being unjustly denied loans,
  - to drivers who trust autopilot too much being involved in fatal crashes
- The biggest challenge is in making sure that this update is good enough to be deployed!
- The risks for catastrophic failures amplify with continual learning
  - More frequently models are updated, the more opportunities there are for updates to fail
  - Makes models more susceptible to coordinated manipulation and adversarial attack
- To avoid similar or worse incidents, it's crucial to thoroughly test each of model updates to ensure its performance and safety before deploying the updates to a wider audience.
- When designing the evaluation pipeline for continual learning, keep in mind that evaluation takes time,
  - which can be another bottleneck for model update frequency.

Continual Learning Challenges(4)

Algorithm challenge - "softer" challenge

- Challenge as it only affects certain algorithms and certain training frequencies.
  - only affects matrix-based and tree-based models that want to be updated very fast (e.g., hourly)
- Consider two different models:
  - a neural network and a matrix-based model, such as a collaborative filtering model
  - The collaborative filtering model uses a user-item matrix and a dimension reduction technique
- A neural network model
  - Can update model with a data batch of any size
  - Can even perform the update step with just one data sample
- Collaborative filtering model
  - For updating model, first need to use the entire dataset to build the user-item matrix before performing dimensionality reduction on it
  - if matrix is large, the dimensionality reduction step would be too slow and expensive to perform frequently
  - less suitable for learning with a partial dataset than the preceding neural network model

Four Stages of Continual Learning

- How to overcome these challenges and make continual learning happen?
- The move toward continual learning happens in four stages
  - Stage 1: Manual, stateless retraining
  - Stage 2: Automated retraining
  - Stage 3: Automated, stateful training
  - Stage 4: Continual learning

Four Stages of Continual Learning(2)

Stage 1: Manual, stateless retraining

- In the beginning, the ML team often focuses on developing ML models to solve as many business problems as possible
  - Because team is focusing on developing new models, updating existing models takes a backseat
  - No fixed frequency to update the models
- Update an existing model only when the following two conditions are met:
  - the model's performance has degraded to the point that it's doing more harm than good
  - team has time to update it
- The process of updating a model is manual and ad hoc
  - Someone, usually a data engineer, has to query the data warehouse for new data.
  - Someone else cleans this new data, extracts features from it, retrains that model from scratch on both the old and new data, and then exports the updated model into a binary format.
  - Someone else takes that binary format and deploys the updated model.
- A vast majority of companies outside the tech industry—
  - e.g., any company that adopted ML less than three years ago and doesn't have an ML platform team—are in this stage

Four Stages of Continual Learning(3)

Stage 2: Automated retraining

- After a few years, team has managed to deploy models to solve most of the obvious problems
  - Priority is no longer to develop new models, but to maintain and improve existing models
  - The ad hoc, manual process of updating models mentioned from the previous stage has grown into a pain point too big to be ignored
- Team decides to write a script to automatically execute all the retraining steps
  - run periodically using a batch process such as Spark
- Status
  - Most companies with somewhat mature ML infrastructure are in this stage
  - Some sophisticated companies run experiments to determine the optimal retraining frequency
  - For most companies in this stage, the retraining frequency is set based on gut feeling
    - e.g., "once a day seems about right" or "let's kick off the retraining process each night when we have idle compute"

Four Stages of Continual Learning(4)

Stage 2: Automated retraining - Requirements for setup

- If company has ML models in production,
  - likely that company already has most of the infrastructure pieces needed for automated retraining
- The feasibility of this stage revolves around the feasibility of writing a script to automate workflow and configure infrastructure to automatically:
  1. Pull data.
  2. Downsample or upsample this data if necessary.
  3. Extract features.
  4. Process and/or annotate labels to create training data.
  5. Kick off the training process.
  6. Evaluate the newly trained model.
  7. Deploy it.
- In general, the three major factors that will affect the feasibility of this script are:
  - scheduler,
  - data,
  - and model store

Four Stages of Continual Learning(4)

Stage 3: Automated, stateful training

- Need to reconfigure automatic updating script so that, when the model update is kicked off
  - it first locates the previous checkpoint and loads it into memory before continuing training on this checkpoint
- Requirements
  - The main thing needed at this stage is a way to track data and model lineage
- Imagine you first upload model version 1.0
  - is updated with new data to create model version 1.1, and so on to create model 1.2
  - Another model is uploaded and called model version 2.0
  - is updated with new data to create model version 2.1
  - After a while, you might have model version 3.32, model version 2.11, model version 1.64
- Might want to know
  - how these models evolve over time,
  - which model was used as its base model,
  - which data was used to update it so that you can reproduce and debug it
- Need model store that has this model lineage capacity

Four Stages of Continual Learning(5)

Stage 4: Continual learning

- At stage 3, models are still updated based on a fixed schedule set out by developers
  - Finding the optimal schedule isn't straightforward and can be situation-dependent
    - might want models to be automatically updated whenever data distributions shift and the model's performance plummets
  - The move from stage 3 to stage 4 is steep!
- Requirements
  - First need a mechanism to trigger model updates
    - Time-based - every five minutes
    - Performance-based - whenever model performance plummets
    - Volume-based - whenever the total amount of labeled data increases by 5%
    - Drift-based - whenever a major data distribution shift is detected
  - For this trigger mechanism to work, need a solid monitoring solution
    - Hard part is not to detect the changes, but to determine which of these changes matter
    - If monitoring solution gives a lot of false alerts, model will end up being updated much more frequently than it needs to be
  - Need a solid pipeline to continually evaluate model updates.
    - Writing a function to update models isn't much different from stage 3
    - Hard part is to ensure that the updated model is working properly

How Often to Update Your Models?

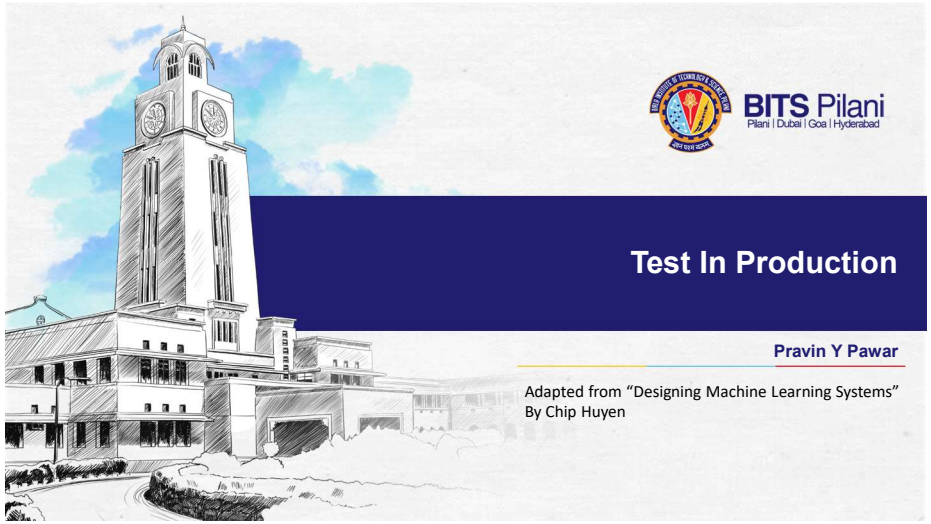
- Depends on how much gain model will get from being updated with fresh data
  - The more gain model can get from fresher data, the more frequently it should be retrained
- The question on how often to update your model is a difficult one to answer!
  - In the beginning, when infrastructure is nascent and the process of updating a model is manual and slow, the answer is: as often as you can.
  - As infrastructure matures and the process of updating a model is partially automated and can be done in a matter of hours, if not minutes,
    - the answer to this question is contingent on the answer to the following question:
    - "How much performance gain would I get from fresher data?"
- Points to be considered
  - Value of data freshness
  - Model iteration versus data iteration

Value of data freshness

- The question of how often to update a model becomes a lot easier
  - if knows how much the model performance will improve with updating
- For example,
  - If switch from retraining model every month to every week, how much performance gain can we get?
  - What if we switch to daily retraining?
- People keep saying that data distributions shift, so fresher data is better,
  - but how much better is fresher data?
    - One way to figure out the gain is by training model on the data from different time windows in the past and evaluating it on the data from today to see how the performance changes.

Model iteration versus data iteration

- Not all model updates are the same - can be model iteration or data iteration!
- Might wonder not only how often to update model, but also what kind of model updates to perform
  - In theory, can do both types of updates
  - In practice, should do both from time to time
- The more resources spend in one approach, the fewer resources can be spent in another
  - if iterating on data doesn't give much performance gain, then should spend resources on finding a better model
  - if finding a better model architecture requires 100X compute for training and gives 1% performance
  - whereas updating the same model on data from the last three hours requires only 1X compute and also gives 1% performance gain, will be better off iterating on data
- Currently no book can give the answer on which approach will work better for specific model on specific task
  - have to do experiments to find out.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Test In Production

Pravin Y Pawar

Adapted from "Designing Machine Learning Systems"
By Chip Huyen

Offline Evaluation

- An offline model evaluation happens when the model is being trained by the analyst
- The analyst tries out different
  - features,
  - models,
  - algorithms,
  - and hyperparameters
- Model training is guided in the right direction by
  - Tools like confusion matrix
  - Various performance metrics, such as precision, recall, and AUC
- Process
  - First, validation data is used to assess the chosen performance metric and compare models
  - Once the best model is identified, the test set is used, also in offline mode, to again assess the best model's performance
  - This final offline assessment guarantees post-deployment model performance

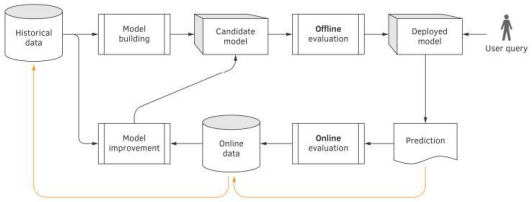
Offline Evaluation(2)

Two major test types for offline evaluation

- Test splits
  - are usually static and have to be static so that you have a trusted benchmark to compare multiple models
  - will be hard to compare the test results of two models if they are tested on different test sets
- Backtests
  - method of testing a predictive model on data from a specific period of time in the past
  - If model is updated to adapt to a new data distribution, it's not sufficient to evaluate this new model on test splits from the old distribution
  - one idea is to test model on the most recent data that you have access to - after updated model on the data from the last day, might want to test this model on the data from the last hour
- The question is whether backtests are sufficient to replace static test splits. Not quite!
  - If something went wrong with data pipeline and some data from the last hour is corrupted, evaluating model solely on this recent data isn't sufficient.
  - With backtests, should still evaluate model on a static test set that have been extensively studied and (mostly) trust as a form of sanity check.

Offline and Online Evaluation

- Historical data is first used to train a deployment candidate
  - Then it is evaluated offline
  - If the result is satisfactory, deployment candidate becomes the deployed model, and starts accepting user queries
- User queries and the model predictions are used for an online evaluation of the model
  - The online data is then used to improve the model
  - To close the loop, the online data is permanently copied to the offline data repository



The placement of offline and online model evaluations in a machine learning system.

Source: Machine Learning Engineering by Burkov

Why do we evaluate both offline and online?

- The offline model evaluation reflects how well the analyst succeeded
  - in finding the right features, learning algorithm, model, and values of hyperparameters
  - reflects how good the model is from an engineering standpoint
- Online evaluation, focuses on measuring business outcomes,
  - such as customer satisfaction, average online time, open rate, and click-through rate
  - may not be reflected in historical data, but it's what the business really cares about
- Offline evaluation doesn't allow us to test the model in some conditions that can be observed online,
  - such as connection and data loss, and call delays

Test in production

Aka Online Evaluation

- The only way to know whether a model will do well in production is to deploy it
  - led to one seemingly terrifying but necessary concept
- Techniques to help to evaluate models in production (mostly) safely
  - Shadow deployment
  - A/B testing
  - Canary analysis
  - Interleaving experiments
  - Bandits
- To sufficiently evaluate models, first need a mixture of offline evaluation and online evaluation!

Shadow Deployment

Might be the safest way to deploy model or any software update

- Works as follows
  1. Deploy the candidate model in parallel with the existing model
  2. For each incoming request, route it to both models to make predictions, but only serve the existing model's prediction to the user
  3. Log the predictions from the new model for analysis purposes.
- Only when found that the new model's predictions are satisfactory do replace the existing model with the new model!
- The risk of this new model doing something funky is low
  - as don't serve the new model's predictions to users until made sure that the model's predictions are satisfactory
- Always not favorable because it's expensive
  - doubles the number of predictions system has to generate, which generally means doubling inference compute cost

A/B Testing

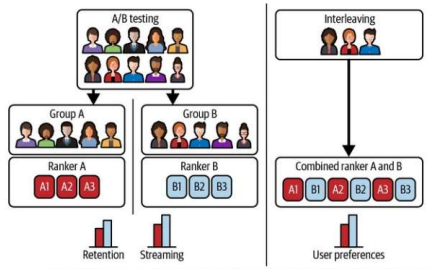
- A way to compare two variants of an object, typically by testing responses to these two variants, determining which of the two variants is more effective
  - the existing model as one variant, and the candidate model (the recently updated model) as another variant
- Works as follows:
  1. Deploy the candidate model alongside the existing model
  2. A percentage of traffic is routed to the new model for predictions; the rest is routed to the existing model for predictions.
  3. Monitor and analyze the predictions and user feedback, if any, from both models to determine whether the difference in the two models' performance is statistically significant.
- Two important things:
  - First, A/B testing consists of a randomized experiment: the traffic routed to each model has to be truly random.
    - If not, the test result will be invalid
  - Second, A/B test should be run on a sufficient number of samples to gain enough confidence about the outcome.
    - How to calculate the number of samples needed for an A/B test is a simple question with a very complicated answer.
- Often, in production, you don't have just one candidate but multiple candidate models
  - possible to do A/B testing with more than two variants, which means can have A/B/C testing or even A/B/C/D testing

Canary Release

- A technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users
  - before rolling it out to the entire infrastructure and making it available to everybody
- Works as follows:
  1. Deploy the candidate model alongside the existing model. The candidate model is called the canary
  2. A portion of the traffic is routed to the candidate model.
  3. If its performance is satisfactory, increase the traffic to the candidate model.
     If not, abort the canary and route all the traffic back to the existing model.
  1. Stop when either the canary serves all the traffic (the candidate model has replaced the existing model) or when the canary is aborted
- The candidate model's performance is measured against the existing model's performance according to the metrics you care about
  - If the candidate model's key metrics degrade significantly, the canary is aborted and all the traffic will be routed to the existing model
- Canary releases can be used to implement A/B testing due to the similarities in their setups
  - don't have to randomize the traffic to route to each model
  - first roll out the candidate model to a less critical market before rolling out to everybody

Interleaving Experiments

- Imagine two recommender systems, A and B, and want to evaluate which one is better
  - Each time, a model recommends 10 items users might like
- With A/B testing, divide users into two groups: one group is exposed to A and the other group is exposed to B.
  - Each user will be exposed to the recommendations made by one model
- What if instead of exposing a user to recommendations from a model, we expose that user to recommendations from both models and see which model's recommendations they will click on?



An illustration of interleaving versus A/B testing. Source: Adapted from an image by Parks

Interleaving Experiments(2)

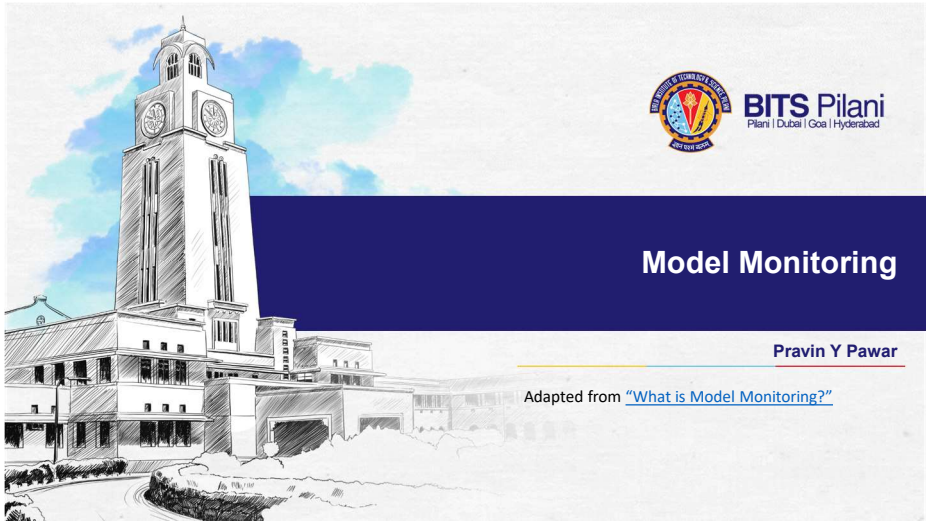
- In A/B testing, core metrics like retention and streaming are measured and compared between two groups
- In interleaving, two algorithms can be compared by measuring user preferences,
  - no guarantee that user preference will lead to better core metrics
- Netflix found that interleaving "reliably identifies the best algorithms with considerably smaller sample size compared to traditional A/B testing."
- When recommendations from multiple models are shown to users,
  - the position of a recommendation influences how likely a user will click on it
  - users are much more likely to click on the top recommendation than the bottom recommendation
- For interleaving to yield valid results, must ensure that at any given position,
  - a recommendation is equally likely to be generated by A or B.

Bandits

- Bandit algorithms originated in gambling!
- A slot machine is also known as a one-armed bandit
  - A casino has multiple slot machines with different payouts
  - don't know which slot machine gives the highest payout
  - Can experiment over time to find out which slot machine is the best while maximizing payout
- Multi-armed bandits are algorithms
  - allow you to balance between
    - exploitation (choosing the slot machine that has paid the most in the past)
    - and exploration (choosing other slot machines that may pay off even more)
- When multiple models needs to be evaluated, each model can be considered a slot machine whose payout (i.e., prediction accuracy) is unknown
- Bandits allow to determine how to route traffic to each model for prediction to determine the best model while maximizing prediction accuracy for users

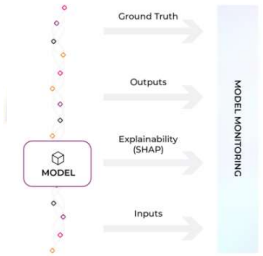
Bandits(2)

- Bandit is stateful: before routing a request to a model, need to calculate all models' current performance
- Requires three things:
  - Model must be able to make online predictions
  - Preferably short feedback loops
  - A mechanism to collect feedback, calculate and keep track of each model's performance, and route prediction requests to different models based on their current performance
- Bandits are well-studied in academia and have been shown to be a lot more data efficient than A/B testing
  - require less data to determine which model is the best
  - at the same time, reduce opportunity cost as they route traffic to the better model more quickly
  - But a lot more difficult to implement than A/B testing because it requires computing and keeping track of models' payoffs



What is Model Monitoring?

- A series of techniques used to detect and measure issues that arise with machine learning models
  - Once configured, monitors fire when a model metric crosses a threshold
- Areas of focus for monitoring include
  - model performance,
  - data quality,
  - drift detection
  - and embedding analysis



Why Is Model Monitoring Important?

- A lot can go wrong with a production model
  - navigating model issues can be challenging for even the most seasoned machine learning (ML) practitioner
- With model monitoring,
  - can immediately know when issues arise in machine learning models
  - better insights empower data scientists and ML engineers to pinpoint where to begin further analysis
- Model Monitoring Improves Business Outcomes
- Despite the multitude of problems a model can encounter in the real world,
  - over half of ML teams lack a reliable way to proactively surface when something is going wrong with a model in production
  - Many rely on
    - batch in-house solutions or dashboards that may not catch issues in time
    - tools that are not purpose-built for machine learning
- In an era where ML models relied on to increase profitability and even save lives, it's clear that better model monitoring is critical.

Model Performance Management

- Model performance indicates how model performs in production
  - Measure model performance with an evaluation metric,
  - which can be evaluated with daily or hourly checks
  - on metrics such as accuracy, recall, precision, F1, MAE, MAPE, and more
  - model type directs which performance metrics are applicable to model
- Performance monitor recommendations:
  - Performance monitors measure performance based on an evaluation metric
  - Can use performance metrics to compare model behavior between different environments
  - Use those insights to drill into the root cause of performance degradation
  - Important to look at the performance of models across various cohorts and slices of predictions

Drift Monitoring

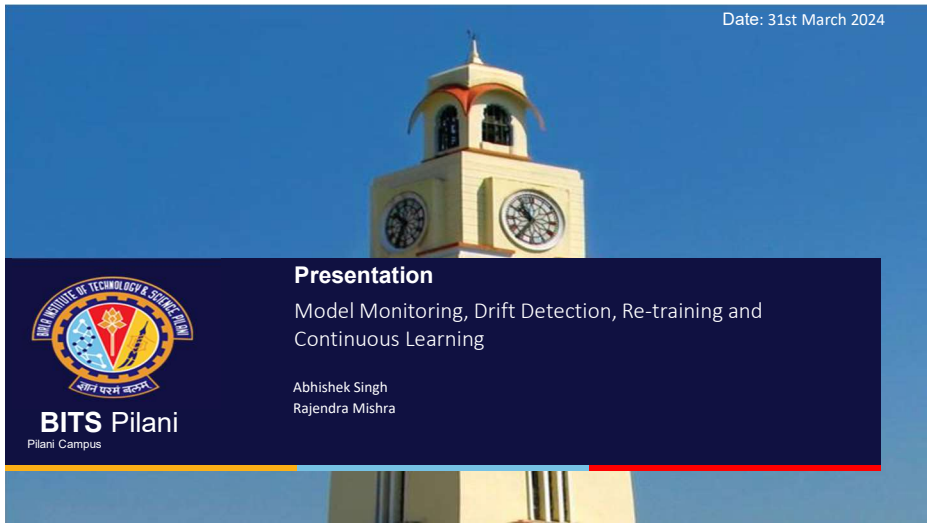
- Drift monitors measure distribution drift, which is the difference between two statistical distributions
- Models are trained with polished data to represent production environments
  - it's common for real-world production data to deviate from training parameters over time
  - need to measure drift to identify if models have grown stale, have data quality issues, or if there are adversarial inputs in model
- Drift monitor recommendations:
  - To detect drift over time, set baseline using training data to identify how model changes between features, predictions, and actuals
  - To detect short-term drift, set baseline using historical production data (i.e. two weeks).
  - With proactive monitoring, detecting drift should be easy with automatic alerts
  - Bulk-create monitors with multiple baselines, view feature performance at a glance, access a historical view of drift, and access the distribution view associated with drift metric

Data Quality Monitoring

- ML models rely on upstream data to train and make predictions
  - Data is commonly collected from multiple systems, vendors, or can be owned by another team,
  - making it difficult to ensure always have high-quality data
- Model health depends on high-quality data that powers model features!
  - Important to immediately surface data quality issues to identify how data quality maps to model's performance
- Data quality monitors help identify key data quality issues
  - such as cardinality shifts, data type mismatch, missing data, and more
- Data quality monitor recommendations:
  - Use data quality monitors to detect shifts in upstream data and alert underlying changes
  - Configure data quality monitors to detect data issues like change in cardinality and change in percent of missing values

Monitoring Unstructured Data

- Most companies building computer vision (CV) or natural language processing (NLP) models lack a window into their models
  - are performing in production, running the risk of models impacting earnings or acting in unfair ways
- Since deep learning models rely on human labeling teams,
  - identifying new patterns in production and troubleshooting performance can be tricky
- CV and NLP Model Monitoring Recommendations:
  - Visualizing and monitoring embeddings — vector representations of data where linear distances capture structure
  - By monitoring embeddings, ML teams can proactively identify when their unstructured data is drifting



Agenda

- Introduction
- Dashboard Tools
- Infrastructure monitoring
- ML Model Monitoring
- Model Training Monitoring
- Model Tuning Monitoring
- Model Serving resources monitoring
- Model Serving Monitoring
- Model re-training and deployment
- Model Re-training Policy
- Stateless Vs Stateful model re-training
- Four Stages of Continuous Learning
- Summary
- Python Packages available to detect drifts
- References
- Demo

Introduction

In machine learning system lifecycle, an important step begins, once we deploy our models to production.

Traditionally, with rule based, deterministic, software, the majority of the work occurs at the initial stage and once deployed, our system works as we've defined it.

But with machine learning, we haven't explicitly defined how something works but used data to architect a ***probabilistic solution***.

This approach is subject to **natural performance degradation over time**, as well as unintended behaviour**, since gradually the data exposed to the model will be different from what it has been trained on**. This isn't something we should be trying to avoid but rather **understand and mitigate as much as possible**.

In addition, there are **many aspects of the machine learning system that we need to monitor** to ensure that our machine learning use cases works as expected.

•Infrastructure monitoring

•ML Model Monitoring
1. Model Training Monitoring
2. Model Serving Monitoring

•Model Serving resources monitoring

Dashboard Tools

Dashboard tools play a crucial role in the MLOps lifecycle by providing insights into model performance, detecting potential issues, and ensuring that machine learning systems remain robust and efficient.

Prometheus

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. It is widely used for monitoring both machine learning models and infrastructure components, making it an essential tool for MLOps. Key features of Prometheus include:

•Flexible data model and powerful query language (PromQL), enabling users to define custom metrics and perform complex queries to gain insights into their systems
•Pull-based data collection, allowing Prometheus to scrape metrics from multiple sources, such as applications, databases, and other infrastructure components
•Built-in alerting system, enabling users to define custom alert rules and receive notifications when specific conditions are met
•Integration with popular visualization tools, such as Grafana, providing users with customizable dashboards and visualizations of their monitoring data

By incorporating Prometheus into their MLOps workflows, data scientists and engineers can effectively monitor their machine learning models and infrastructure components, ensuring the reliability and performance of their systems.

Dashboard Tools

Grafana

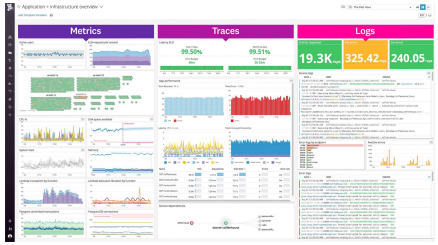
Grafana is an open-source analytics and monitoring platform that allows users to create, explore, and share dashboards and visualizations of their data. Grafana supports a wide range of data sources, including Prometheus, Elasticsearch, and InfluxDB, making it a versatile option for monitoring and performance management in MLOps. Key features of Grafana include:

•Customizable dashboards, allowing users to create and share visualizations of their monitoring data, such as model performance metrics and infrastructure metrics
•Flexible alerting system, enabling users to define custom alert rules and receive notifications through various channels, such as email, Slack, or PagerDuty
•Extensive plugin ecosystem, providing users with additional data sources, panels, and themes to customize their monitoring experience
•Integration with popular authentication and authorization systems, such as OAuth, LDAP, and GitHub, ensuring secure access to monitoring data

By using Grafana in their MLOps workflows, data scientists and engineers can gain valuable insights into their models' performance and infrastructure, identify potential issues, and optimize their systems for better results

Infrastructure **monitoring**

The first step to ensure that our model is performing well is to ensure that the actual system is up and running as it should. This can include metrics specific to service requests such as latency, throughput, error rates, etc. as well as infrastructure utilization such as CPU/GPU utilization, memory, etc. Fortunately, most MLOps providers and even CaaS layers will provide this insight into our system's health through a dashboard. In the event we don't, we can easily use Grafana, Datadog, etc. to ingest system performance metrics from logs to create a customized dashboard and set alerts. (we need to rely on the underlying platform for this)



ML **Model Monitoring**

Just monitoring the system's health won't be enough to capture the underlying issues with our model.

So we need various levels of model monitoring.

We need to **monitor training and tuning metrics**, **monitoring the model serving resources** e.g. throughput & latency, and finally, **model monitoring for issues like data drift, model decay** to take a decision to trigger re-training of the model in case the prediction drifts from ground truth beyond a threshold.

Model Training Monitoring

The first level of metrics to monitor and log involves the model training and tuning performance. These would be quantitative evaluation metrics that we used during model training and evaluation e.g. accuracy, precision, f1, etc. There are different metrics to be evaluated in case of regression or classification cases.

Regression Metrics
The most common metrics for evaluating predictions on regression machine learning problems:

•Mean Absolute Error.
•Mean Squared Error.
•R^2.
•Training parameters

Classification Metrics
Classification problems are perhaps the most common type of machine learning problem and as such there are a lots of metrics that can be used to evaluate predictions for these problems:

•Classification Accuracy.
•Logarithmic Loss.
•Area Under ROC Curve.
•Confusion Matrix.
•Classification Report.
•Training parameters

Model Tuning Monitoring

Once we've done model training and evaluation, it's possible that we want to see if you can further improve our training in any way. We can do this by **tuning hyper-parameters**. There were a few parameters we implicitly assumed when we did our training, and now is a good time to go back and test those assumptions and try other values. the following parameters can be monitored and logged:

- Fine-tuning parameters
- Metric related to model accuracy

In machine learning, the terms **model parameters** and **hyperparameters** are often used interchangeably. However, there is a subtle difference between the two.

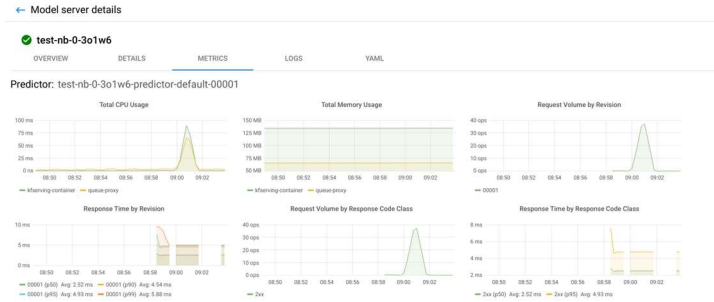
•**Model parameters** are the values that are learned by the model during training. They are typically represented by a vector of numbers. The number of model parameters depends on the complexity of the model. For example, a simple linear regression model may have only a few model parameters, while a complex deep learning model may have millions or even billions of model parameters.

•**Hyperparameters** are the values that are set before training the model. They control the learning process and the behavior of the model. For example, a hyperparameter might be the number of iterations to train the model, the learning rate, or the batch size. Hyperparameters can have a significant impact on the performance of the model.

In general, model parameters are learned from data, while hyperparameters are set by the user. However, there are some cases where model parameters can be set by the user, such as when using a pre-trained model.

Model Serving resources monitoring

An important part of monitoring system resources is monitoring the performance of model serving infrastructure. Once the model is trained and served, we are going to use the model serving for quiet sometime. So its important to ensure that the performance of model serving is as per expectation..



Model Serving Monitoring

There are 2 questions which we need to answer:

1. Does new incoming data reflect the same patterns as the data on which model was originally trained on ?
2. Is the model performing as well in development as it did in design phase ? If not, why ?

Machine Learning Model Drift

How well a model performs is the reflection of the data used to train it. If there is a significant change in distribution or composition of values of input variables or the target variables, it could lead to data drift which could cause the model to degrade. Model degradation can lead to inaccurate predictions and insights, therefore we want to carefully monitor model degradation. *To track model degradation there are 2 approaches to consider:*

1. ***Based on Ground Truth*** - Labelled data is compared to prediction
2. ***Based on Data Drift*** - Instead of waiting for the labelled data, the training data and recent data are compared statistically.

Model Serving Monitoring

Model Monitoring based on Ground Truth vs Data Drift

The ground truth is the correct answer that the model was asked to solve -- when we know the ground truth for all the predictions a model has made, we can judge with certainty how well the model is performing.

Ground truth monitoring requires waiting for the label event and then computing the performance of the model based on these ground truth observations. When the ground truth is available quickly it can the best solution to monitor model degradation, however obtaining ground truth can be slow and costly.

If our use cases requires rapid feedback, or if the ground truth is not available or hard to compute, input drift evaluation may be the way to go. The basis for input drift, is that the model is only going to predict accurately , if the data it was trained on is an accurate reflection of the real world, so if a comparison of the recent requests to a deployed model against the training data , shows distinct differences, then there is a strong likelihood that the model performance may be compromised.

Unlike for ground truth evaluation, the data required for input drift evaluation already exists, so there is no need to wait for any other information.

Note: ***If we wait to catch the model decay based on the ground-truth performance, it may have already caused significant damage to downstream business pipelines that are dependent on it. We need to employ more fine-grained monitoring to identify the sources of model drift prior to actual performance degradation.***

Model Serving Monitoring

Types of Drift

We need to first understand the different types of issues that can cause our model's performance to decay (model drift). The best way to do this is to look at all the moving pieces of what we're trying to model and how each one can experience drift.

Drift Type	Description
Feature/Input Drift → (X)	Input feature(s) distributions deviate. Also known as ***Covariate Shift.***
Label/Target Drift → (y)	Label/Target distribution deviates. Also known as ***Prior Probability Shift.***
Prediction Drift → P(y)	Model prediction distribution deviates. Also known as ***Model Drift***
Concept Drift → P(y|X)	External factors cause labels to evolve. Also known as ***Task Drift***

Model Serving Monitoring

Drift types and actions to take:

Feature/Data Drift	• Investigate feature generation process • Retrain using new data
Label/Target Drift	• Investigate label generation process • Retrain using new data
Prediction Drift	• Investigate Model training process • Access business impact of changes in prediction
Concept Drift	• Investigate additional feature engineering • Consider alternative approach/solution • Retrain/tune using new data

Model re-training and deployment

•It's usually a good idea to establish policies around when we're going to retrain our model.

•There's no right or wrong answer here, so it will depend on what works in our particular situation.

•we could simply choose to retrain our model whenever it seems to be necessary.

•That includes situations where we detect a drift, but it also includes situations where we always retrain the models according to a schedule, or on demand.

•***In practice, this is what many companies do, because it's simple to understand and in many domains, it works fairly well.***

•It can, however, incur higher training and data gathering costs unnecessarily.

•If we can automate the process of detecting the conditions which require model retraining, that will be ideal.

•That includes being able to detect model performance degradation or when we detect significant data drift, and triggering retraining.

•In both cases, in order to automate retraining, we should have data gathered and labelled automatically and only retrain when sufficient data is available.

•We need to have continuous training, integration and deployment setup to make the process fully automated.

Model Re-training Policy

Policy	Intuition
On Demand	Re-train the model on need basis (e.g. when ACTS detects that the deployed model is old, it triggers re-training and deployment)
On Schedule	Re-train at a fixed internal on a daily, weekly or monthly basis.
On Data Drift	We find significant changes in the input data and trigger model retraining.
On Model Performance Degradation	Prediction Drift based on ground truth, trigger model retraining.
On New data availability	When new data is available on ad-hoc basis and there is no trend. Re-train model when new labelled data is collected and available in source database.

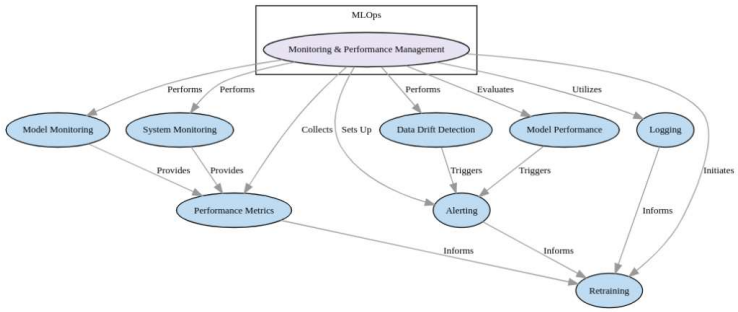
Stateless Vs Stateful model re-training

Policy	Intuition
Stateless	• New model is retrained from scratch each time.
Stateful	• The existing model continues to be trained on new data • Stateful re-training is also known as fine-tuning or incremental learning

Four Stages of Continuous Learning

Stage	Description
Stage 1: On-Demand, Stateless retraining	This is the first stage of continuous learning. In this stage we update the models on demand. In this stage the models are updated on need basis or as deemed suitable.
Stage 2: Automated, Stateless retraining	In this stage we write automated and recurring pipelines which rely mostly on schedule to trigger re-training. Most companies with somewhat mature ML infrastructure are in this stage.
Stage 3: Automated, Stateful retraining	In this stage we update existing model which continues to be trained on new data. Stateful re-training is also known as fine-tuning or incremental learning.
Stage 4: Continual learning	Till Stage 3 our models are still updated based on a fixed schedule. This might not be optimal always as quite a few models might not show any performance degradation and hence the stage 3 might lead to higher compute costs. On the flip side some of our models could decay/degrade much faster and require a much faster and automated retraining. In stage 4 our re-training trigger can be combination of one or more factors e.g. data drift, model performance degradation (beyond threshold) or arrival of new data.

Summary



Python Packages available to detect drifts

Some of the open-source packages available to detect drift are:

Evidently AI: Open-Source Tool To Analyze Data Drift. Evidently is an open-source Python library for data scientists and ML engineers. It helps evaluate, test, and monitor the performance of ML models from validation to production.

drift_detection: This package contains some developmental tools to detect and compare statistical differences between 2 structurally similar pandas dataframes. The intended purpose is to detect data drift — where the statistical properties of an input variable change over time. It provides a class DataDriftDetector which takes in 2 pandas dataframes and provides a few useful methods to compare and analyze the differences between the 2 datasets.

skmultiflow: One of the key features of skmultiflow is its ability to handle concept drift, which is the change in the underlying distribution of the data over time. It includes algorithms for detecting concept drift and adapting the machine learning model in real-time to account for the changes in the data. This makes it an ideal tool for building machine learning models that can adapt and continue to perform well as the data changes over time.

Deepchecks: Deepchecks Open Source is a python library for data scientists and ML engineers. The package includes extensive test suites for machine learning models and data, built in a way that's flexible, extendable, and editable.

NannyML: NannyML is an open-source Python library that helps you monitor and improve the performance of your machine learning models

References

- An overview of unsupervised drift detection methods
- Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift
- Monitoring and explainability of models in production
- Detecting and Correcting for Label Shift with Black Box Predictors
- Outlier and anomaly pattern detection on data streams

Thank You !

Demo

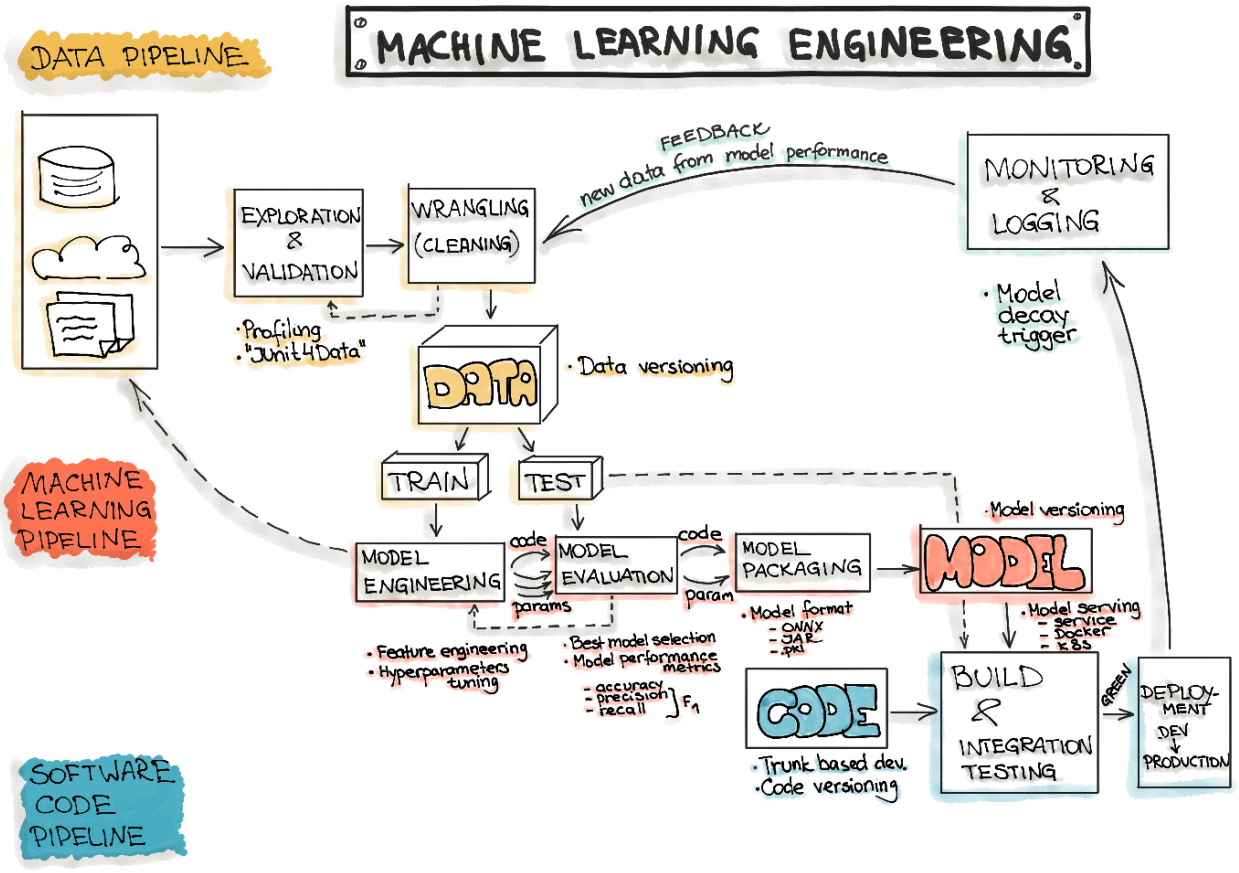
MLOps | Syllabus for Midsem exam

Dear Students,

The syllabus for midsem exam will consist of all the topics that we will cover till 16th January, broadly around

- Three levels of ML software
- ML life-cycle and System Architecture
- Challenges with ML lifecycle
- Motivation and Drivers for MLOps
- Peoples of MLOps
- Key MLOps features and maturity models
- AllTheOps: DataOps, ModelOps, AIOps
- MLOps life-cycle, process and capabilities
- Infrastructure: Storage and Compute
- Dev / Production Env, Runtimes
- ML Platforms
- Landscape of MLOps Tools / Platforms
- Experimentation
- Model Versioning
- Model Metadata
- Model Registry
- Model Packaging
- Model File formats
- Serialization
- Containerization

1.1 Three Levels of ML Software



Data

- Data Ingestion
  - ✓ Collecting data by using various frameworks and formats, such as Spark, HDFS, CSV, etc
  - ✓ Might also include synthetic data generation or data enrichment.

- Data Exploration
  - ✓ Includes data profiling to obtain information about the content and structure of the data
  - ✓ The output of this step is a set of metadata, such as max, min, avg of values

- Data Validation
  - ✓ Operations are user-defined error detection functions, which scan the dataset in order to spot some errors

- Data Wrangling
  - ✓ The process of re-formatting particular attributes and correcting errors in data, such as missing values imputation

- Data Labeling
  - ✓ The operation of the Data Engineering pipeline, where each data point is assigned to a specific category

- Data Splitting
  - ✓ Splitting the data into training, validation, and test datasets to be used during the core machine learning stages to produce the ML model

Model

- Model Training
  - ✓ The process of applying the machine learning algorithm on training data to train an ML model
  - ✓ includes feature engineering and the hyperparameter tuning for the model training activity

- Model Evaluation
  - ✓ Validating the trained model to ensure it meets original codified objectives before serving the ML model in production to the end-user

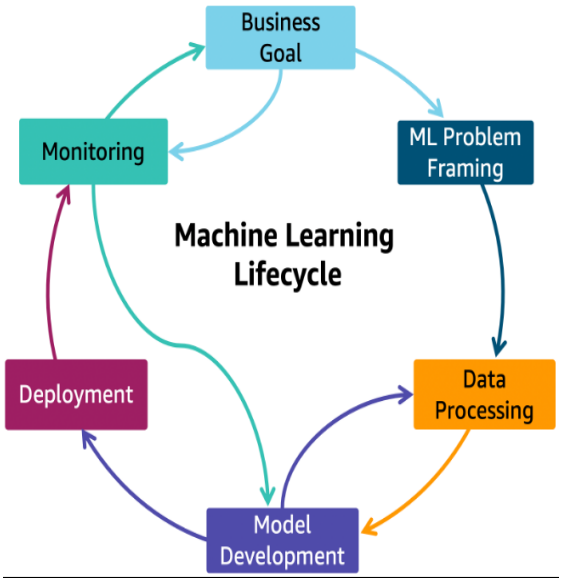
- Model Testing
  - ✓ Performing the final “Model Acceptance Test” by using the hold backtest dataset

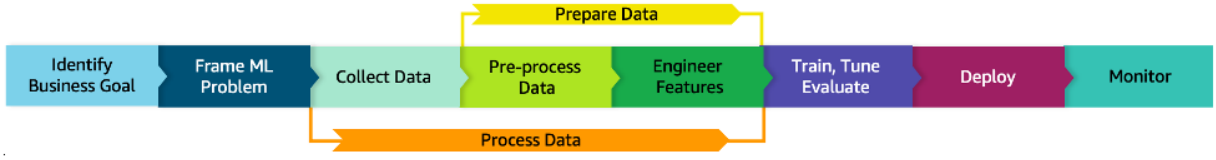
- Model Packaging
  - ✓ The process of exporting the final ML model into a specific format (e.g. PMML, PFA, or ONNX)
  - ✓ which describes the model, in order to be consumed by the business application

Code

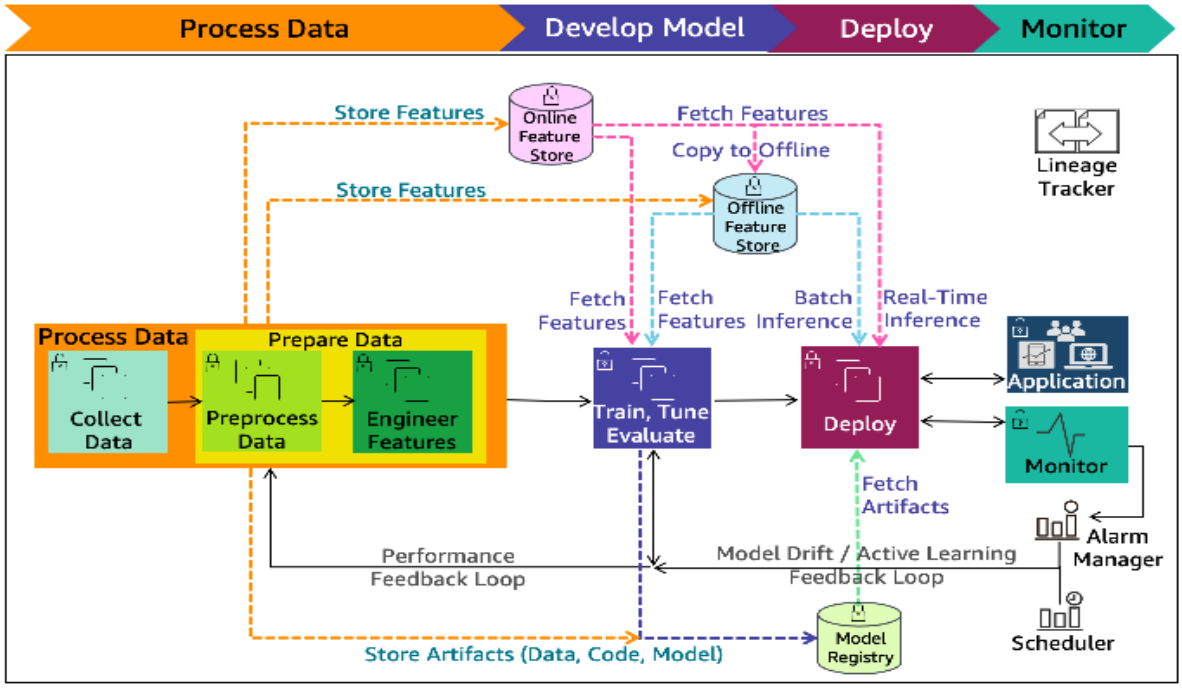
- Conventional Software Development
- ML Model Deployment

1.2 ML Lifecycle and System Architecture





ML lifecycle with data processing sub-phases included



ML lifecycle with detailed phases and expanded components

Business Goal:

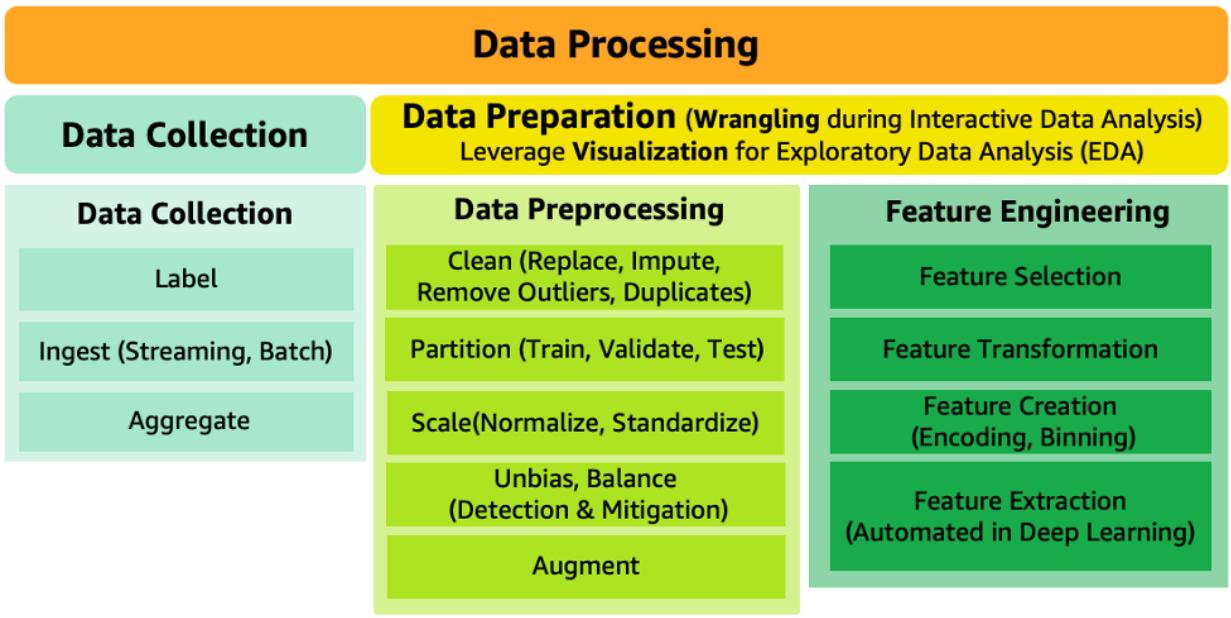
- ✓ should have a clear idea of the problem, and the business value to be gained by solving that problem
- ✓ must be able to measure business value against specific business objectives and success criteria

ML Problem Framing:

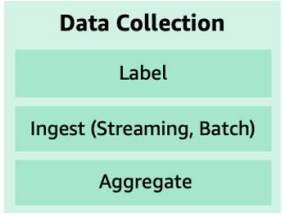
- ✓ the business problem is framed as a machine learning problem
- ✓ what is observed and what should be predicted (known as a label or target variable)
- ✓ Determining what to predict and how performance and error metrics must be optimized is a key step in this phase

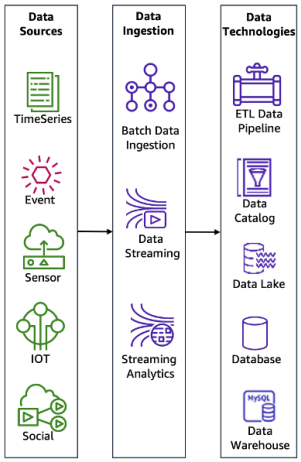
Data Processing:

- ✓ Training an accurate ML model requires data processing to convert data into a usable format
- ✓ includes collecting data, preparing data
- ✓ and feature engineering that is the process of creating, transforming, extracting, and selecting variables from data

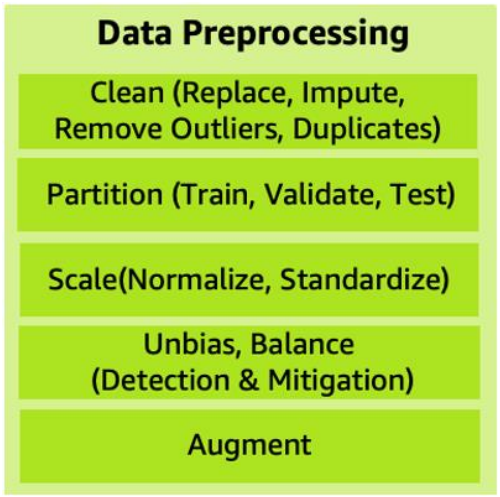


Data Collection

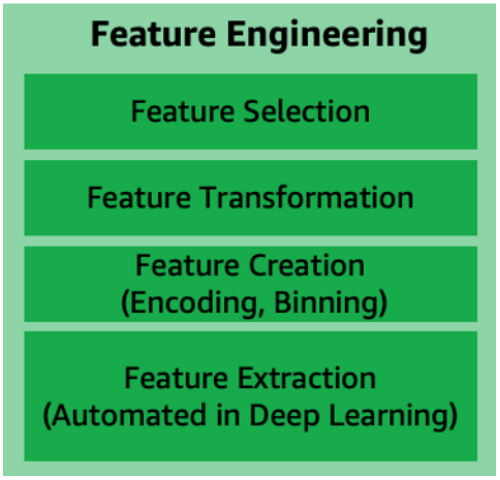




Data Preprocessing

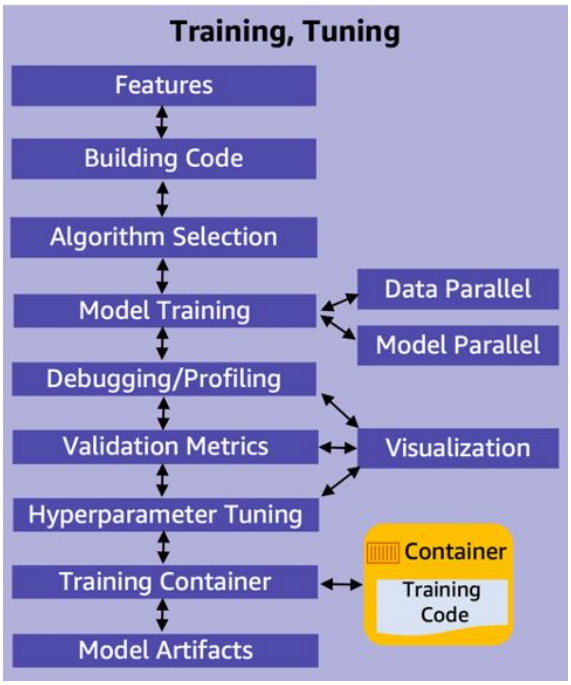


Feature Engineering



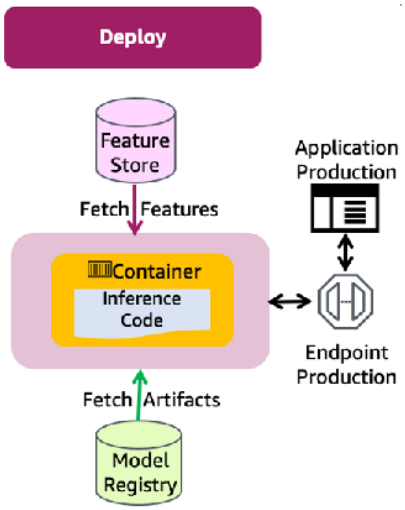
Model Development:

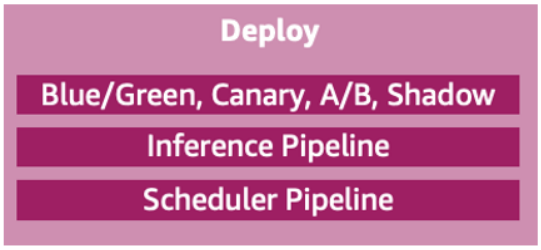
- ✓ consists of model building, training, tuning, and evaluation
- ✓ includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments



Deployment:

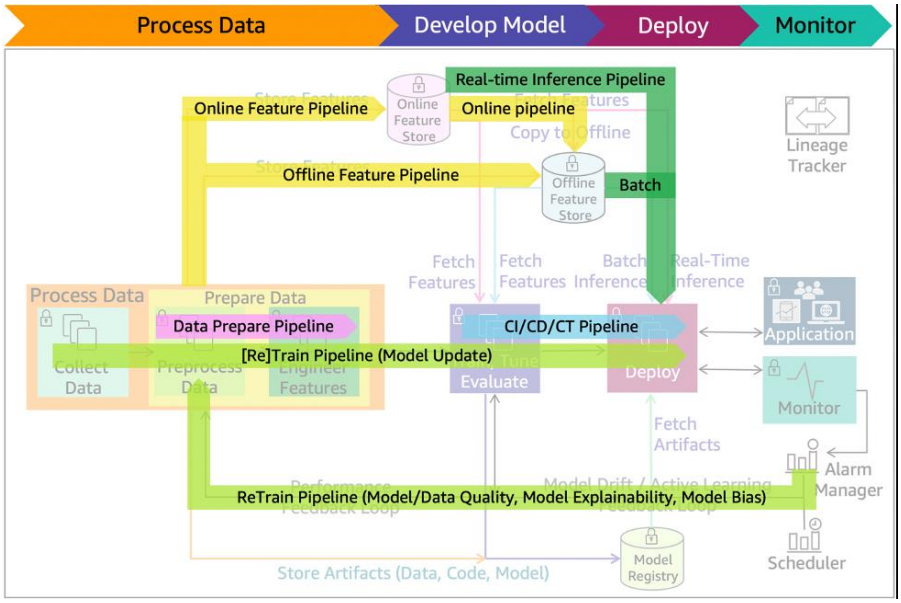
- ✓ After a model is trained, tuned, evaluated and validated, one can deploy the model into production
- ✓ can then make predictions and inferences against the model



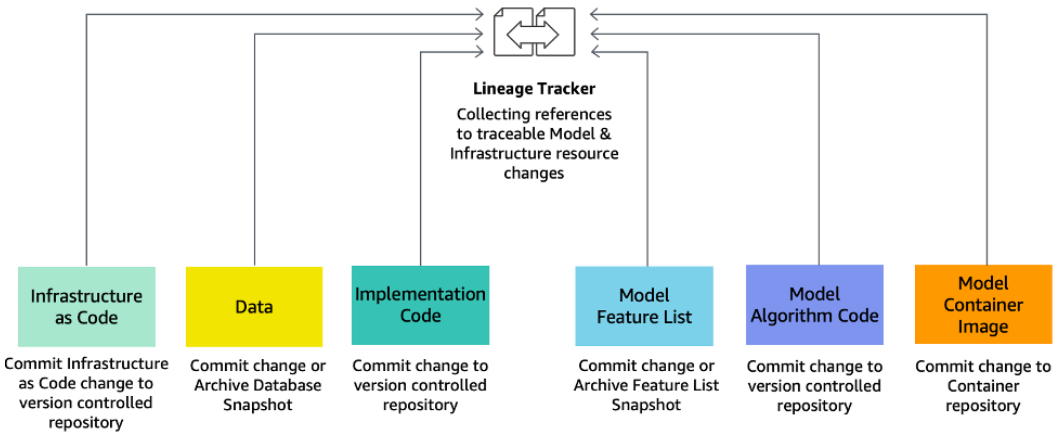


Monitoring:

- ✓ ensures model is maintaining a desired level of performance through early detection and mitigation



Lineage Tracker



Challenges with ML Lifecycle

- Data Collection and Quality:
  - ✓ Insufficient Data: Sometimes there's not enough data to train robust models.
  - ✓ Bias in Data: Data might be biased, leading to biased models.
  - ✓ Data Privacy and Security: Ensuring data privacy, especially with sensitive information, is crucial and challenging.

- Data Preparation
  - ✓ Cleaning and Preprocessing: Data often comes in unstructured or noisy forms and requires extensive cleaning and preprocessing.
  - ✓ Feature Engineering: Selecting the right features that effectively contribute to the predictive power of the model can be complex.

- Model Selection and Training
  - ✓ Choosing the Right Algorithm: With numerous algorithms available, selecting the most suitable one for a specific problem can be difficult.
  - ✓ Overfitting and Underfitting: Achieving the right balance between underfitting and overfitting is a key challenge.
  - ✓ Computational Resources: Training complex models require significant computational resources.

- Model Evaluation
  - ✓ Defining Success Metrics: Choosing appropriate metrics that align with business objectives is essential.
  - ✓ Cross-Validation and Generalization: Ensuring the model generalizes well to new, unseen data is a major challenge.

- Model Deployment
  - ✓ Integration with Existing Systems: Integrating ML models into existing business systems can be technically challenging.
  - ✓ Scalability: Ensuring models can handle large-scale data in a production environment.
  - ✓ Latency Requirements: Some applications require real-time predictions, demanding low-latency solutions.

- Model Monitoring and Maintenance
  - ✓ Model Drift: Models can degrade over time as data patterns change (concept drift).
  - ✓ Continuous Monitoring: Setting up systems to continuously monitor model performance.
  - ✓ Updating Models: Determining when and how to retrain or tweak models.

- Ethical Considerations and Fairness
  - ✓ Avoiding Discrimination: Ensuring models do not propagate or amplify biases.

- ✓ Transparency and Explainability: Making ML models transparent and understandable, especially in critical applications like healthcare or criminal justice.

- Regulatory Compliance
  - ✓ Adhering to Regulations: Complying with legal frameworks like GDPR for data privacy.
  - ✓ Documentation and Reporting: Keeping detailed records for regulatory purposes.

- Team Collaboration and Skill Sets
  - ✓ Cross-Disciplinary Collaboration: Effective collaboration between data scientists, engineers, and domain experts.
  - ✓ Skill Gap: Addressing the gap in skills required for effective implementation of ML solutions.

2.1 Motivation and Drivers for MLOps

- Gaps
  - ✓ Because bridging the gap between ML model building and practical deployments is still a challenging task
  - ✓ The main challenges people face when developing ML capabilities are scale, version control, model reproducibility, and aligning stakeholders
  - ✓

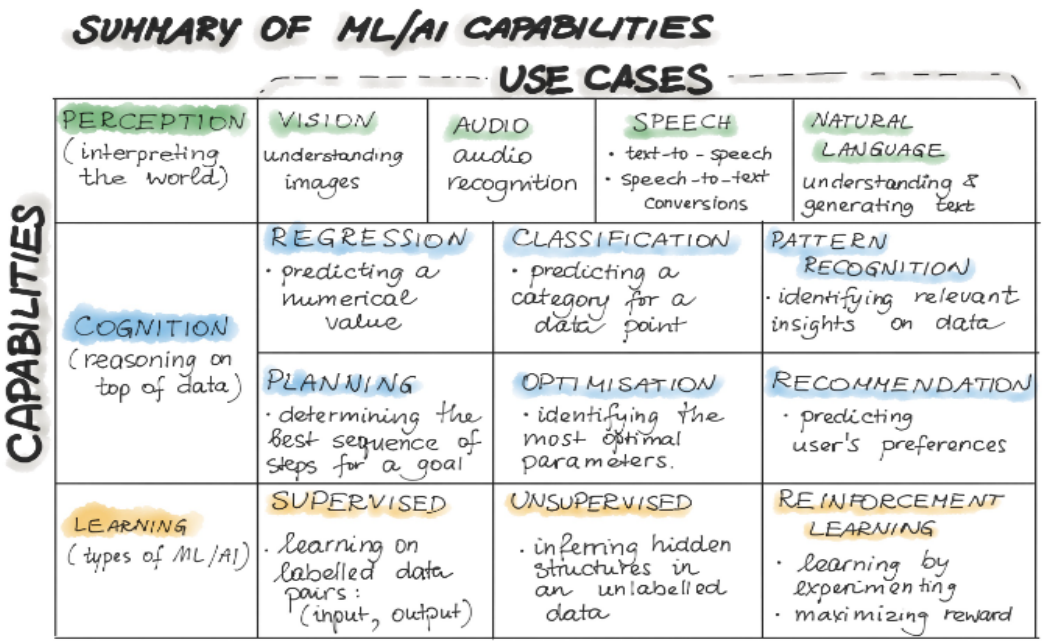
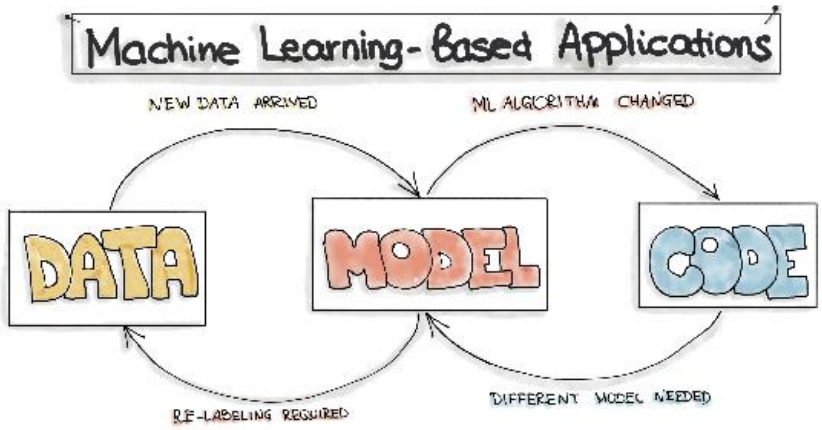
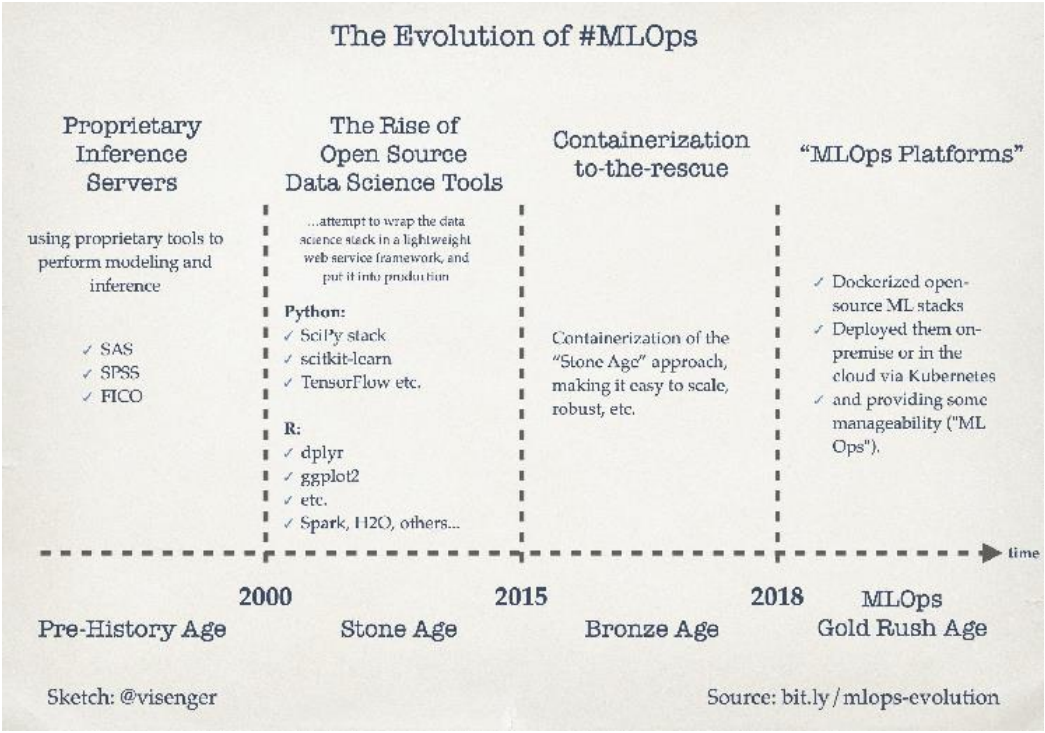


Table Source: "The AI Organization" by David Carmona





2.4 Peoples of MLOps

Role in machine learning model life cycle	MLOps requirements
Provide business questions, goals, or KPIs around which ML models should be framed	Easy way to understand deployed model performance in business terms
Continually evaluate and ensure that model performance aligns with or resolves the initial need	Mechanism or feedback loop for flagging model results that don't align with business expectations

Role in machine learning model life cycle	MLOps requirements
Build models that address the business question or needs brought by subject matter experts	Automated model packaging and delivery for quick and easy (yet safe) deployment to production
Deliver operationalizable models so that they can be properly used in the production environment and with production data	Ability to develop tests to determine the quality of deployed models and to make continual improvements
Assess model quality (of both original and tests) in tandem with subject matter experts to ensure they answer initial business questions or needs	Visibility into the performance of all deployed models (including side-by-side for tests) from one central location
	Ability to investigate data pipelines of each model to make quick assessments and adjustments regardless of who originally built the model

Role in machine learning model life cycle	MLOps requirements
Optimize the retrieval and use of data to power ML models	Visibility into performance of all deployed models
	Ability to see the full details of individual data pipelines to address underlying data plumbing issues

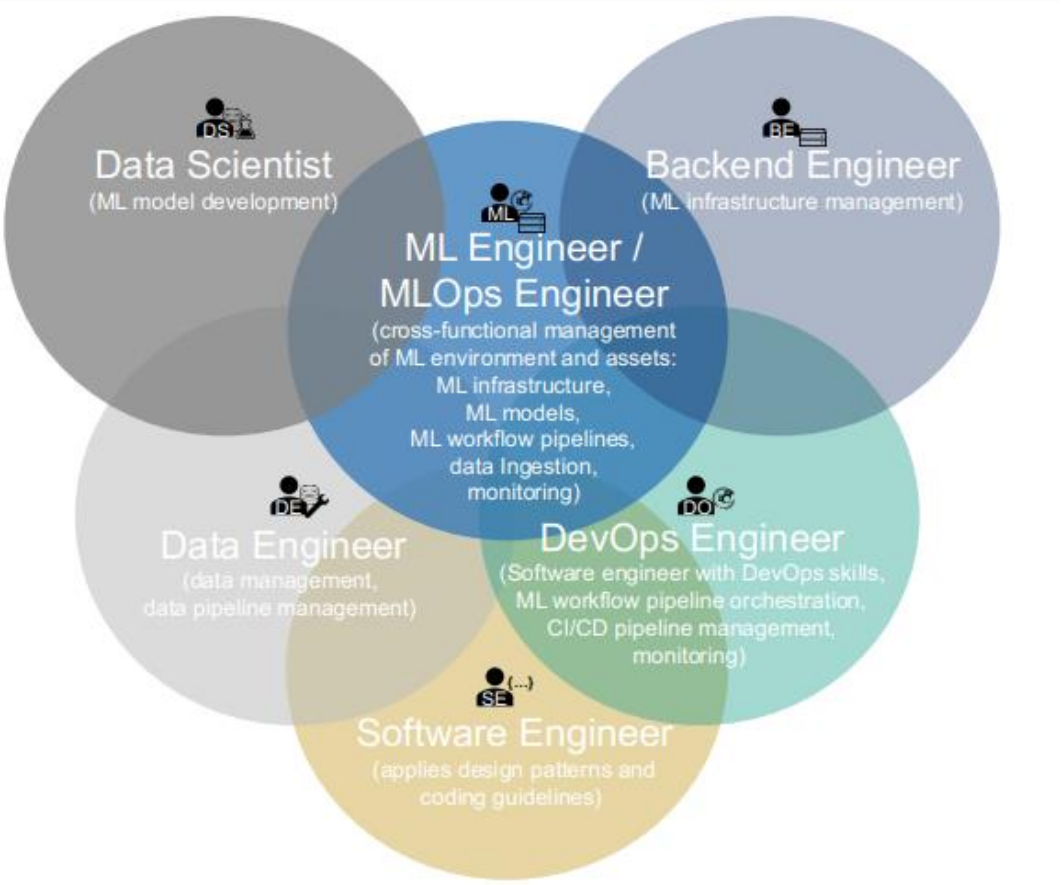
Role in machine learning model life cycle	MLOps requirements
Integrate ML models in the company's applications and systems	Versioning and automatic tests
Ensure that ML models work seamlessly with other non-machine-learning-based applications	The ability to work in parallel on the same application

Role in machine learning model life cycle	MLOps requirements
Conduct and build operational systems and test for security, performance, availability	Seamless integration of MLOps into the larger DevOps strategy of the enterprise
Continuous Integration/Continuous Delivery (CI/CD) pipeline management	Seamless deployment pipeline

Role in machine learning model life cycle	MLOps requirements
Minimize overall risk to the company as a result of ML models in production	Robust, likely automated, reporting tools on all models (currently or ever in production), including data lineage
Ensure compliance with internal and external requirements before pushing ML models to production	

Role in machine learning model life cycle	MLOps requirements
Ensure a scalable and flexible environment for ML model pipelines, from design to development and monitoring	High-level overview of models and their resources consumed
Introduce new technologies when appropriate that improve ML model performance in production	Ability to drill down into data pipelines to assess and adjust infrastructure needs

- ML Engineer
- MLOps Engineer
- Data Scientist
- Backend Engineer
- Data Engineer
- DevOps Engineer
- Software Engineer



3.1 Key MLOps features and Maturity Models

- Development
  - ✓ Establishing Business Objectives, Data Sources and Exploratory Data Analysis
  - ✓ Feature Engineering and Selection, Training and Evaluation

- ✓ Reproducibility

- Deployment
  - ✓ Productionalizing and deploying models is a key component of MLOps
  - ✓ The domain of the software engineer and the DevOps team
  - ✓ Without effective collaboration between the teams, delays or failures to deploy are inevitable

  - Model Deployment Types
    - Model-as-a-service, or live-scoring model
    - Embedded model

  - Model Deployment Requirements
    - Rapid, automated deployment is always preferred to labor-intensive processes
    - For short-lifetime, self-service applications, there often isn't much need to worry about testing and validation

- Monitoring

  - ✓ Once a model is deployed to production, it is crucial that it continue to perform well over time
  - ✓ DevOps Concerns
  - ✓ Data Scientist Concerns
  - ✓ Business Concerns

- Iteration
  - ✓ Developing and deploying improved versions of a model is an essential part of the MLOps life cycle
  - ✓ Various reasons to develop a new model version
  - ✓ Iteration
    - In some fast-moving business environments, new training data becomes available every day
    - With a new model version built, the next step is to compare the metrics with the current live model version
    - Even in the "simple" automated retraining scenario with new training data,
    - Retraining in other scenarios is likely to be even more complicated,

- Governance
  - ✓ Governance is the set of controls placed on a business to ensure that it delivers on its responsibilities
  - ✓ Legal obligations are the easiest to understand
  - ✓ Recently, governments across the world have imposed regulations
  - ✓ Governments are now starting to turn their regulatory eye to ML specifically

- ✓ With increasing public activism on the subject, businesses are engaging with ideas of Responsible AI
- ✓ Applying good governance to MLOPs is challenging
- ✓ Governance initiatives in MLOps broadly fall into one of two categories –
    - Data governance
    - Process governance

3.2 Maturity of the ML process

Three levels of MLOps

- starting from the most common level, which involves no automation, up to automating both ML and CI/CD pipelines

- ✓ MLOps level 0: Manual process
    - Manual, script-driven, and interactive process
    - Disconnection between ML and operations
    - Infrequent release iterations
    - No CI
    - No CD
    - Deployment refers to the prediction service
    - Lack of active performance monitoring

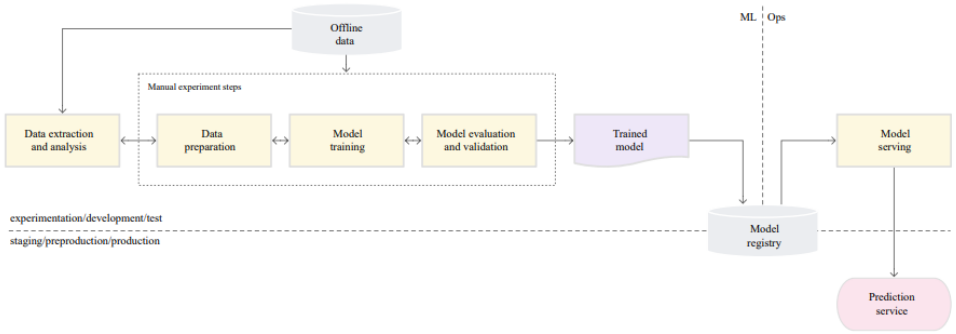


Figure 2. Manual ML steps to serve the model as a prediction service.

- ✓ MLOps level 1: ML pipeline automation
    - Rapid experiment
    - CT of the model in production
    - Experimental-operational symmetry

- Continuous delivery of models
- Modularized code for components and pipelines
- Pipeline deployment

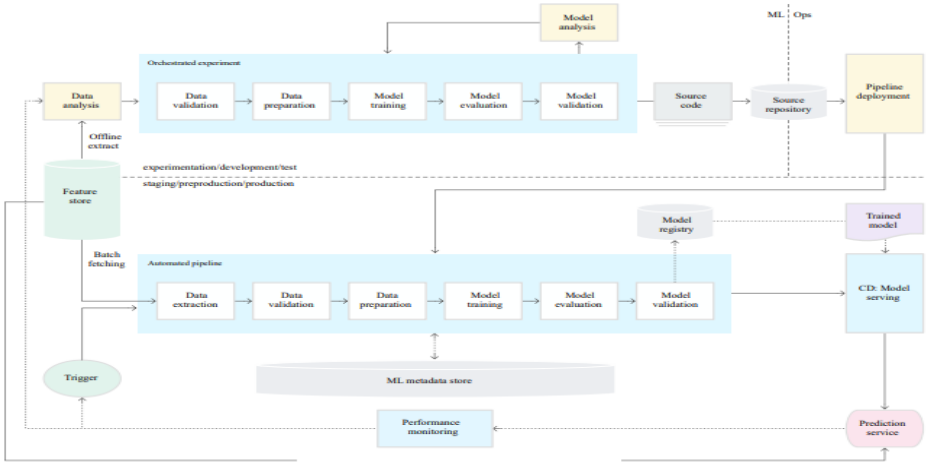


Figure 3. ML pipeline automation for CT.

✓ MLOps level 2: CI/CD pipeline automation

- Development and experimentation
- Pipeline continuous integration
- Pipeline continuous delivery
- Automated triggering
- Model continuous delivery
- Monitoring

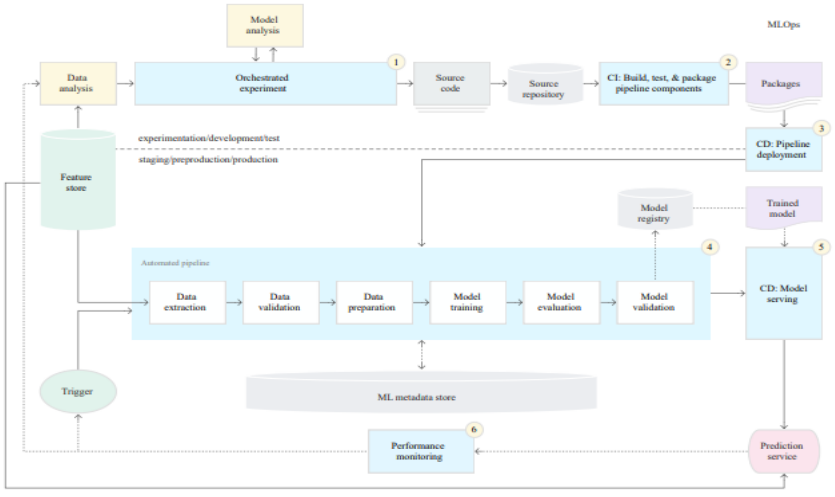
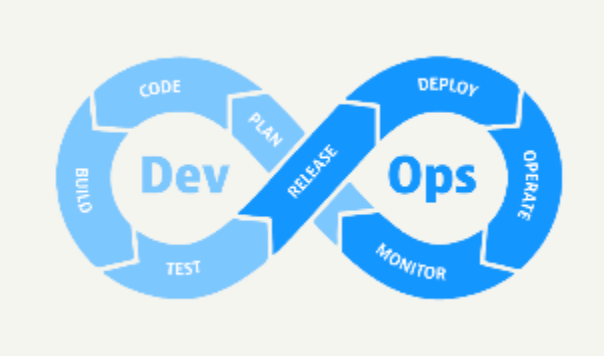


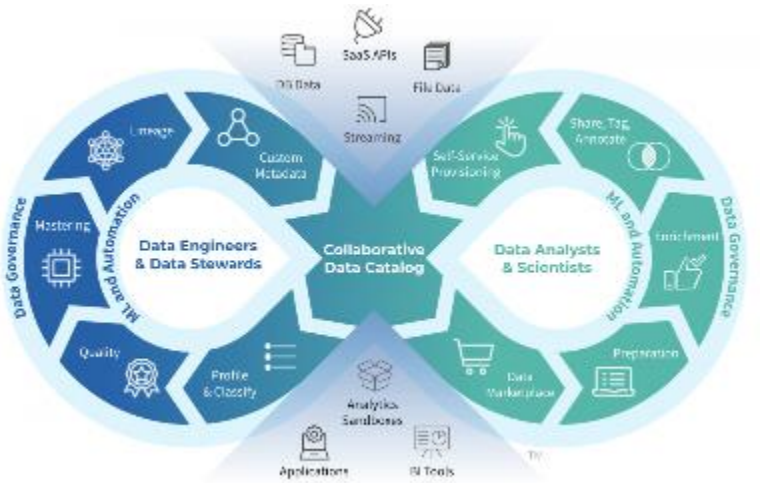
Figure 4. CI/CD and automated ML pipeline.

3.3 AllTheOps: DevOps, MLOps, ModelOps and AIOps

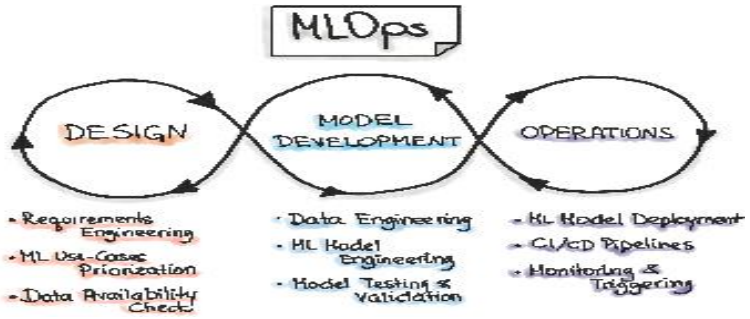
- DevOps
  - ✓ DevOps is a set of practices that combines software development (Dev) and information technology operations (Ops)
  - ✓ DevSecOps was defined and emphasizes the need to build a security foundation into DevOps initiatives



- DataOps
  - ✓ DataOps was defined when
    - DevOps improved the quality and scalability of software development
    - Data analytics discipline sought to bring same improvements to their practices
  - ✓ In a nutshell, DataOps applies agile development, DevOps, and lean manufacturing to data analytics (Data) and operations (Ops)
  - ✓ DataOps is a reference of how Data Engineers and DevOps engineers work together



- MLOps
  - ✓ MLOps is meant to standardize and streamline the lifecycle of machine learning models in production
  - ✓ MLOps (or ML Operations) refers to the process of managing ML workflows
  - ✓ MLOps is the practice of building, deploying and maintaining ML models
  - ✓ MLOps has emerged because ML engineers are increasingly being asked



- ModelOps
  - ✓ ModelOps incorporates MLOps, which is the process of managing ML models throughout their lifecycle at an enterprise scale
  - ✓ The advantage of ModelOps over MLOps is that MLOps focuses on the machine learning models only
  - ✓ The organization looking to set up ModelOps should set up MLOps first before moving on to ModelOps

- AIOps
  - ✓ AIOps (Artificial Intelligence for IT Operations) is the use of machine learning and other AI technologies to automate many processes that are currently done manually in an organization
  - ✓ AIOps is similar to MLOps in that it uses machine learning and other AI technologies to automate IT processes
  - ✓ It is different from MLOps in that
    - o the process automation occurs within an organization's IT operations department instead of an organization's machine learning and AI team
    - o uses AI to automate many processes, not just one or two tasks like MLOps does

4.1 MLOps LifeCycle, Process and Capabilities

- Complexity of ML application
  - ✓ Preparing and maintaining high-quality data for training ML models

- ✓ Tracking models in production to detect performance degradation
- ✓ Performing ongoing experimentation of new data sources, ML algorithms, and hyperparameters, and then tracking these experiments
- ✓ Maintaining the veracity of models by continuously retraining them on fresh data
- ✓ Avoiding training-serving skews that are due to inconsistencies in data and in runtime dependencies between training environments and serving environments
- ✓ Handling concerns about model fairness and adversarial attacks

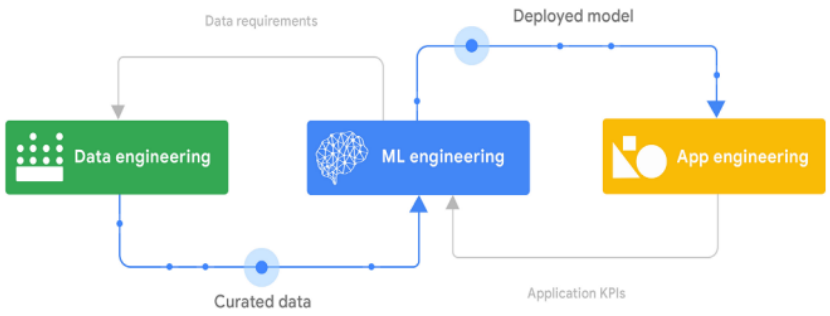
- ML engineering
  - ✓ The common theme is that ML systems cannot be built in an ad hoc manner,
  - ✓ Also cannot be built without adopting and applying sound software engineering practices
  - ✓ Organizations need an automated and streamlined ML process
  - ✓ ML engineering is at the center of building ML-enabled systems, which concerns the development and operationalizing of production-grade ML systems

- MLOps
  - ✓ MLOps is a methodology for ML engineering
  - ✓ MLOps provides a set of standardized processes and technology capabilities
  - ✓ MLOps supports ML development and deployment in the way that DevOps and DataOps support application engineering and data engineering (analytics)

  - ✓ MLOps practices can result in the following benefits over systems that do not follow MLOps practices

    - Shorter development cycles, and as a result, shorter time to market
    - Better collaboration between teams
    - Increased reliability, performance, scalability, and security of ML systems
    - Streamlined operational and governance processes
    - Increased return on investment of ML projects

- Building an ML enabled system

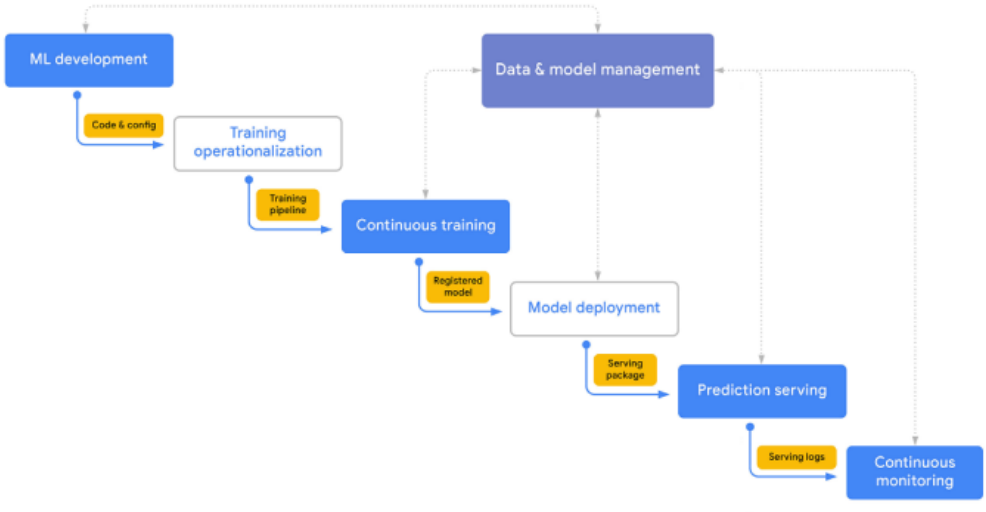


The relationship of data engineering, ML engineering, and app engineering

- MLOps Life Cycle

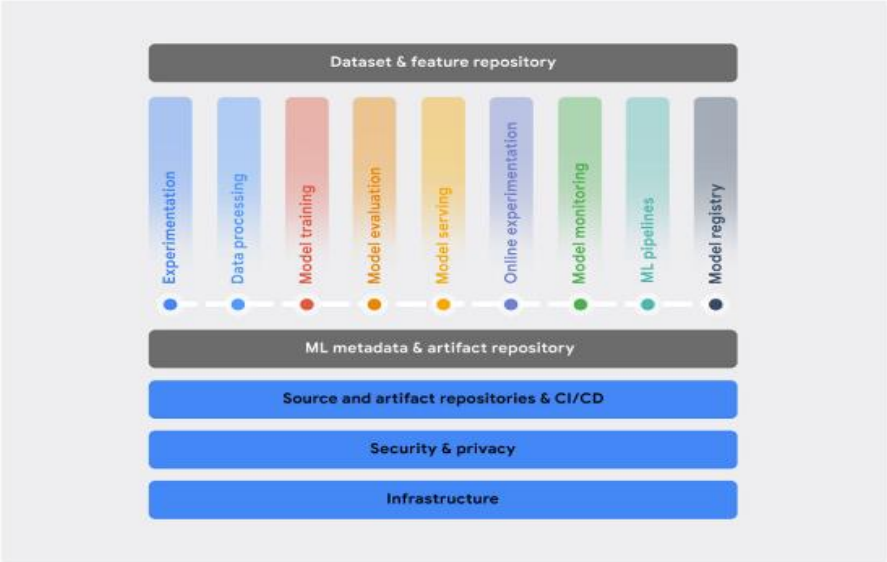
The MLOps lifecycle

- MLOps: An end to end workflow



The MLOps process

- MLOps Capabilities



Core MLOps technical capabilities

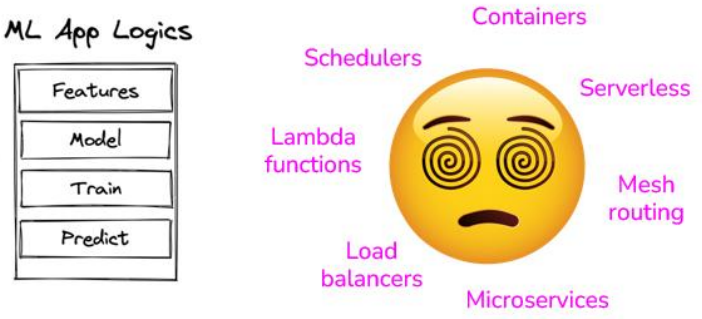
- MLOps Capability – Experimentation

  Lets data scientists and ML researchers collaboratively
  - ✓ perform exploratory data analysis,
  - ✓ create prototype model architectures
  - ✓ and implement training routines

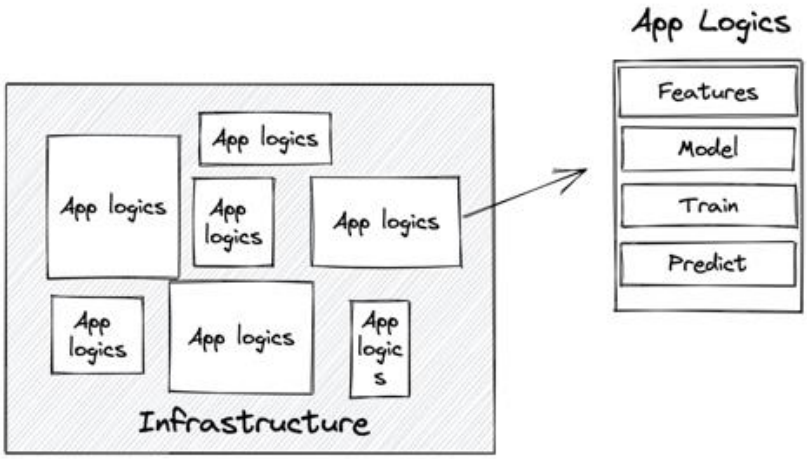
  An ML environment should also let them write modular, reusable, and testable source code

  Key functionalities in experimentation:
  - ✓ Provide notebook environments that are integrated with version control tools like Git
  - ✓ Track experiments, including information about the data, hyperparameters, and evaluation metrics for reproducibility and comparison
  - ✓ Analyze and visualize data and models
  - ✓ Support exploring datasets, finding experiments, and reviewing implementations
  - ✓ Integrate with other data services and ML services in platform

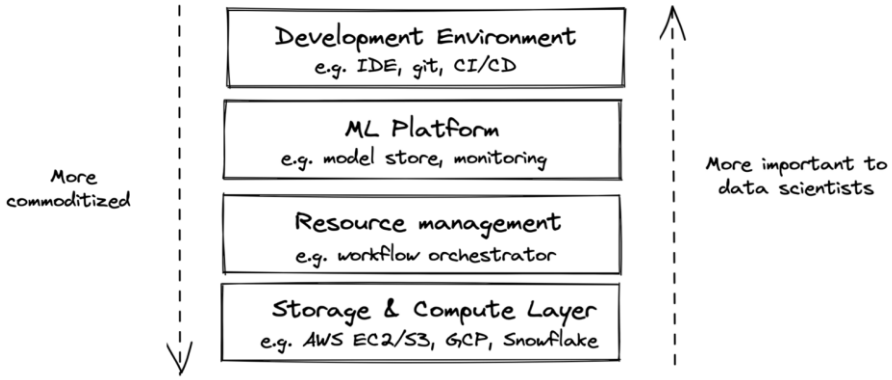
4.2 Infrastructure : Storage and Compute



- ML Infrastructure



- Infrastructure Layer



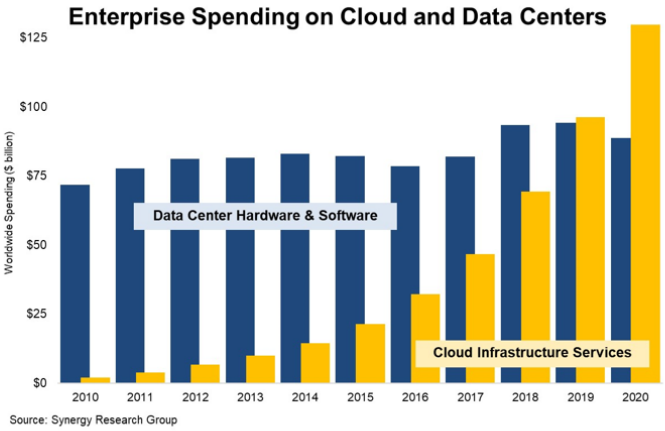
- Storage Layer

Part 2. Data Systems Fundamentals

- Data Sources
- **Data Formats**
  - JSON
  - Row-major vs. Column-major Format
  - Text vs. Binary Format
- Data Models
  - Relational Model
  - NoSQL
    - Document Model
    - Graph Model
  - Structured vs. Unstructured Data
- **Data Storage Engines and Processing**
  - Transactional and Analytical Processing
  - ETL: Extract, Transform, Load
    - ETL to ELT

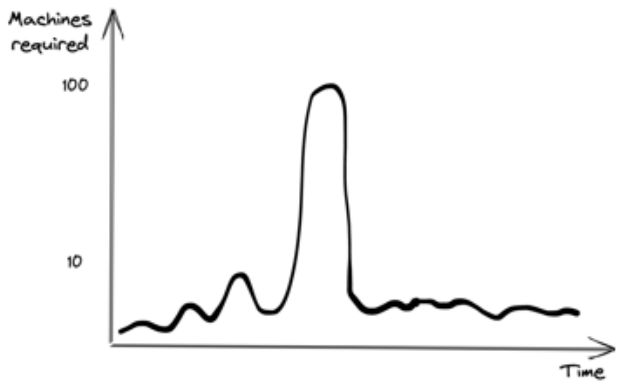
- Compute Unit
  - Compute layer can be sliced into smaller compute units to be used concurrently
    - A CPU core might support 2 concurrent threads
    - Each thread is used as a compute unit to execute its own job
  - Multiple CPUs can be joined to form a large compute unit to execute a large job

- Public Cloud v/s Private Cloud Datacenters

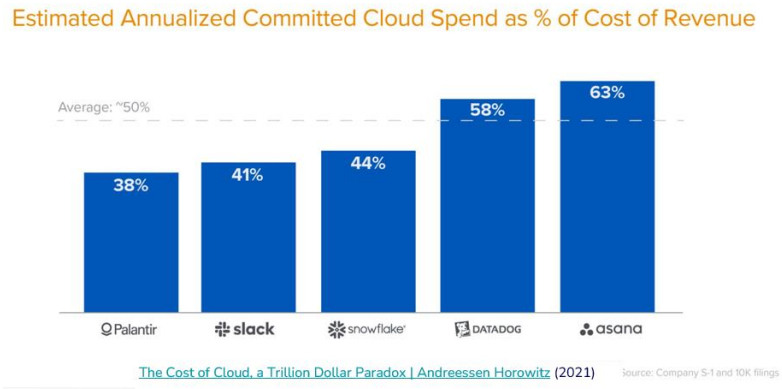


2020 – The Year that Cloud Service Revenues Finally Dwarfed Enterprise Spending on Data Centers | Synergy Research Group

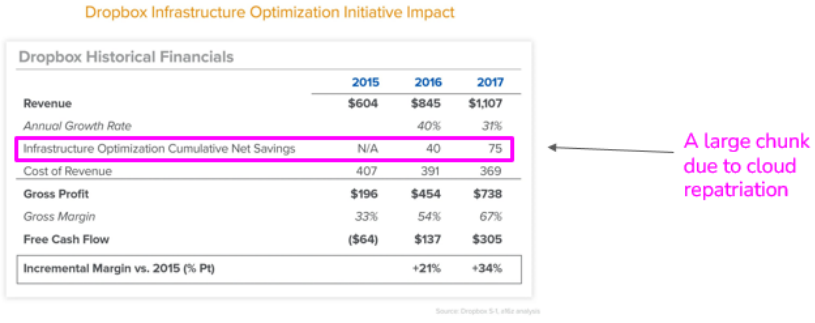
- Benefits of cloud
  - ✓ Easy to get started
  - ✓ Appealing to variable-sized workloads
    - Private: would need 100 machines upfront, most will be idle most of the time
    - Cloud: pay for 100 machines only when needed



- Drawback of cloud cost



- Cloud repatriation



4.3 and 4.4 Development/Production Environment Runtime

- Development Environment
  - Versioning
    - Git: code versioning

        - DVC: data versioning
        - WandB: experiment versioning

    - CI/CD test suite
        - test code before pushing it to staging/prod

- Standardize dev environment
    - Simplify IT support
    - Security: revoke access if laptop is stolen
    - Bring your dev env closer to prod env
    - Make debugging easier

- Container orchestration
    - Help deploy and manage containerized applications to a serverless cluster
    - Spinning up/down containers

- Preparation for Production
    - Confirming that something works in the laboratory has never been a sure sign it will work well in the real world
    - Not only is the production environment typically very different from the development environment
    - Important that the complexities of the transition to production are understood and tested

- Production Runtime Environment
    - Production environments take a wide variety of forms:
        - custom-built services,
        - data science platforms,
        - dedicated services like TensorFlow Serving,
        - low-level infrastructure like Kubernetes clusters,
        - JVMs on embedded systems, etc

- Adaption to Production from Development Environment
    - On One end of the spectrum, the development and production platforms are from the same vendor or are otherwise interoperable
    - On the other end of the spectrum, there are cases where the model needs to be reimplemented
    - Still the reality in many organizations - lack of appropriate tooling and processes
    - In all cases, it is crucial to perform validation in an environment that mimics production as closely as possible

- Tooling Consideration
    - The format required to send to production should be considered early as it may have
    - Teams should set up tooling while developing the model
    - Failure to create pipeline up front would block the validation process

- Performance consideration
    - Performance also comes into play when the production model must run on a low-power device
    - For these models, an optimized runtime is not enough
        - To obtain better performance, the model definition must be optimized
        - One solution is to use compression techniques
            - quantization
            - pruning
            - distillation

- Data Access before validation and launch to production
    - data can be frozen and bundled with the model
    - When this is not possible, the production environment should access a database
        - ✓ have to have the
            - appropriate network connectivity, libraries, or drivers required to communicate with the data storage installed
            - authentication credentials stored in some form of production configuration

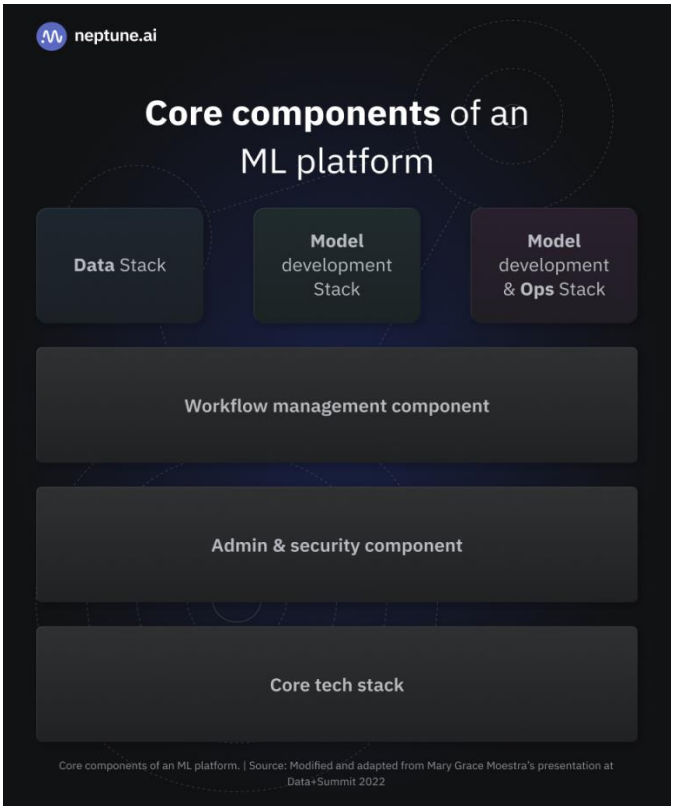
5.1 Machine Learning Platform

- What is ML Platform

    - An ML platform standardizes the technology stack for data team around best practices to
        - ✓ reduce incidental complexities with machine learning
        - ✓ better enable teams across projects and workflows

    - Machine learning operations (MLOps) should be easier with ML platforms
        - ✓ at all stages of a machine learning project's life cycle, from prototyping to production at scale,
        - ✓ as the number of models in production grows from one or a few to tens, hundreds, or thousands

    - The platform should be
        - ✓ designed to orchestrate machine learning workflow,
        - ✓ environment-agnostic (portable to multiple environments),
        - ✓ and work with different libraries and frameworks



Core components of an ML platform. | Source: Modified and adapted from Mary Grace Moestra's presentation at Data+Summit 2022

- The features of an ML platform and the core components that make up its architecture are:
    - Data stack and model development stack
    - Model deployment and operationalization stack
    - Workflow management component
    - Administrative and security component
    - Core technology stack

- MLOps principles that ML Platform can solve
    - MLOps principles that can help ML platforms solve different kinds of problems:
        - Reproducibility
        - Versioning
        - Automation
        - Monitoring
        - Testing
        - Collaboration
        - Scalability

- Users of ML Platform
    - Subject Matter Experts
        - SMEs are the non-developer experts in the business problem
        - have critical roles to play across the entire ML lifecycle
        - Work with other users, to make sure
            - data reflects the business problem,
            - experimentation process is good enough for the business
            - and results reflect what would be valuable to the business

    - ML Engineers and Data Scientists
        - Depending on the existing team structure and processes of the business,
            - ML engineers may work on delivering models to production,
            - Data scientists may focus on research and experimentation
        - Some organizations hire either person to own the end-to-end ML project and not parts of it
        - Goals
            - Frame business problem:

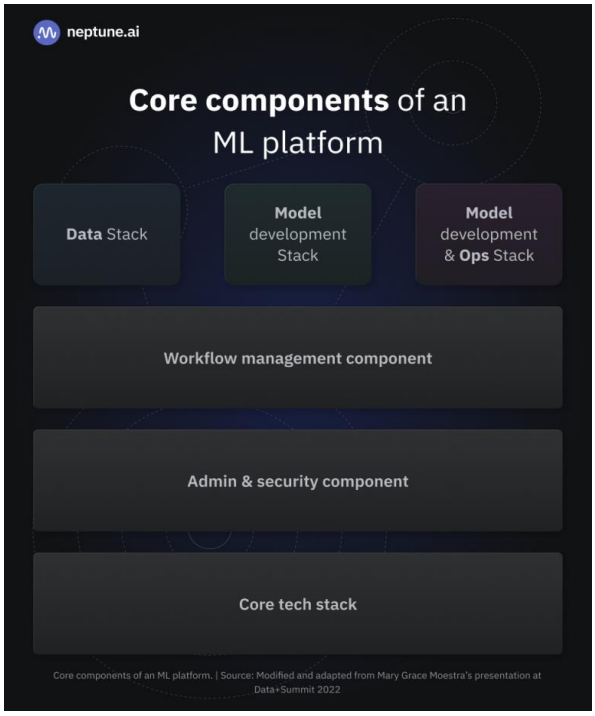
        - collaborate with subject matter experts to outline the business problem in such a way
        - that they can build a viable machine learning solution.
    - Model development:
        - access business data from upstream components,
        - work on the data (if needed),
        - run ML experiments (build, test, and strengthen models),
        - and then deploy the ML model
    - Productionalization:
        - often really subjective in teams because it's mostly the ML engineers that end up serving models

- DevOps Engineers
    - They are either pure software engineers or simply tagged "DevOps engineers" (or IT engineers)
    - They are responsible for CI/CD pipeline management across the entire organizational stack
    - They are mostly responsible for operationalizing the organization's software in production
    - Goal –
        - Perform operational system development and testing
            - to assure the security, performance, and availability of ML models
            - as they integrate into the wider organizational stack

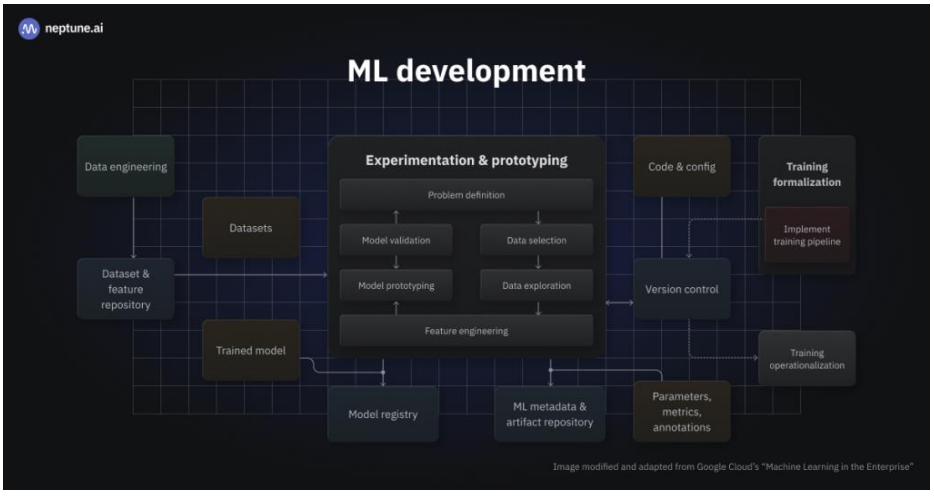
- Data Engineers
    - if the data platform is not particularly separate from the ML platform

- Analytics engineers and data analysts
    - if need to integrate third-party business intelligence tools and the data platform, is not separate

- ML Platform Architecture
    - The features of an ML platform and the core components that make up its architecture are:
        - Data stack and model development stack
        - Model deployment and operationalization stack
        - Workflow management component
        - Administrative and security component
        - Core technology stack



- Data & Model Development Stack
    - Main Components
        - Data and feature store
        - Experimentation component
        - Model registry
        - ML metadata and artifact repository



- Model registry, ML metadata and artifact repository

  - Model registry
    - helps to put some structure into the process of productionalizing ML models
    - stores the validated training model and the metadata and artifacts that go with it

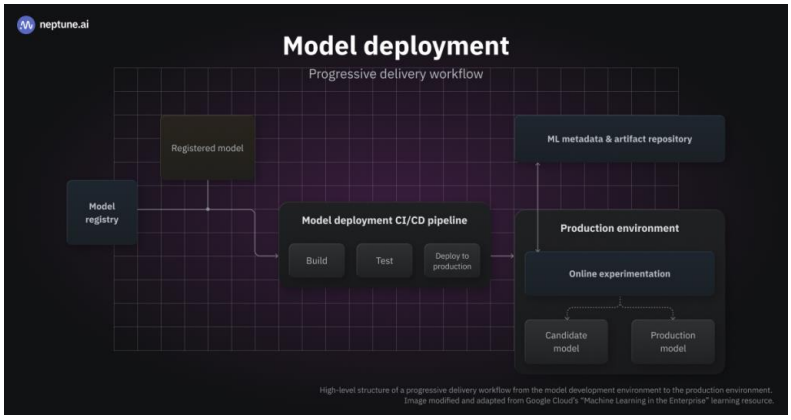
  - Central repository stores and organizes models in a way that
    - makes it more efficient to organize models across the team,
    - making them easier to manage, deploy, and,
    - in most cases, avoid production errors (for example, putting the wrong model into production)

  - ML metadata and artifact repository
    - Need the ML metadata and artifact repository to make it easier to compare model performance and test them in the production environment
    - A model can be tested against the production model, drawing from the ML metadata and artifact store to make those comparisons

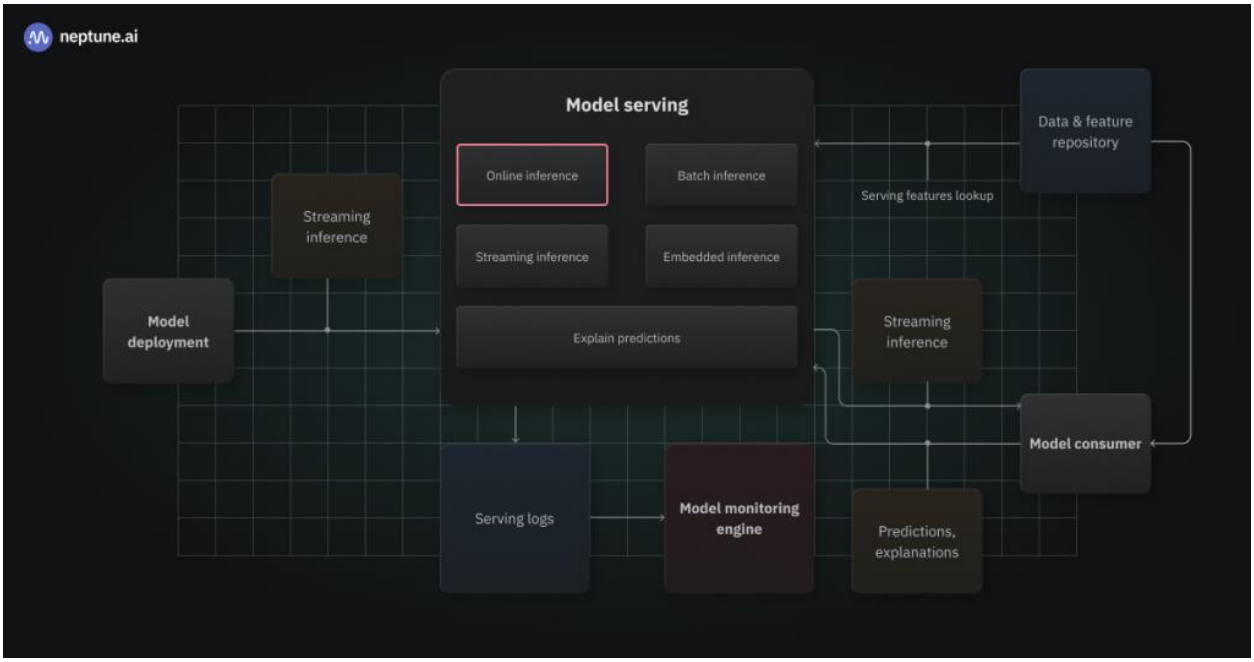
- Model deployment and operational stack

  Main components

  - Production environment
  - Model serving
  - Monitoring and observability
  - Responsible AI and explainability



- Types of Inferences



- Monitoring components
    - A monitoring agent regularly collects telemetry data,
        - such as audit trails, service resource utilization, application statistics, logs, errors, etc.
        - sends the data to the model monitoring engine, which consumes and manages it
    - Inside the engine is a metrics data processor that:
        - Reads the telemetry data,
        - Calculates different operational metrics at regular intervals,
        - And stores them in a metrics database.
    - The monitoring engine
        - has access to production data,
        - runs an ML metrics computer,
        - and stores the model performance metrics in the metrics database
    - An analytics service provides reports and visualizations of the metrics data
        - When certain thresholds are passed in the computed metrics, an alerting service can send a message

- Responsible AI and explainability component
    - Must implement this part together to make sure that the models and products meet the governance requirements, policies, and processes
    - Since ML solutions also face threats from adversarial attacks that compromise the model and data used for training and inference
        - makes sense to inculcate a culture of security for ML assets too, and not just at the application layer (the administrative component)

- Workflow management component

    Main components

    - Model deployment CI/CD pipeline
        - ML models that are used in production don't work as stand-alone software solutions!
            - Instead, they must be built into other software components to work as a whole
            - requires integration with components like APIs, edge devices, databases, microservices, etc
        - The CI/CD pipeline
            - ✓ retrieves the model from the registry,
            - ✓ packages it as executable software,
            - ✓ tests it for regression,
            - ✓ and then deploys it to the production environment
            - ✓ which could be embedded software or ML-as-a-service.
        - The idea of this component is automation
            - ✓ goal is to quickly rebuild pipeline assets ready for production when you push new training code to the corresponding repository
    - Training formalization (training pipeline)
        - Helps you manage repeatable ML training and testing workflows with little human intervention
            - The training pipeline functions to automate those workflows.
        - From:

- ✓ Collecting data from the feature store,
- ✓ To setting some hyperparameter combinations for training,
- ✓ Building and evaluating the model,
- ✓ Retrieving the test data from the feature store component,
- ✓ Testing the model and reviewing results to validate the model's quality,
- ✓ If needed, updating the model parameters and repeating the entire process.

- The pipelines primarily use schedulers
  - would help manage the training lifecycle through a DAG (directed acyclic graph)
  - makes the experimentation process traceable and reproducible,
  - provided the other components discussed earlier have been implemented alongside it

- Orchestrators
  - The orchestrators coordinate
    - ✓ how ML tasks run
    - ✓ where they get the resources to run their jobs
  - Orchestrators are concerned with lower-level abstractions
    - ✓ like machines, instances, clusters, service-level grouping, replication, and so on
  - Integral to managing the regular workflows data scientists run and how the tasks in those workflows communicate with the ML platform
  - Test environment
    - ✓ The test environment gives data scientists the infrastructure and tools
    - ✓ they need to test their models against reference or production data,
    - ✓ usually at the sub-class level,
    - ✓ to see how they might work in the real world before moving them to production

- Core Technology Stack

    Main components

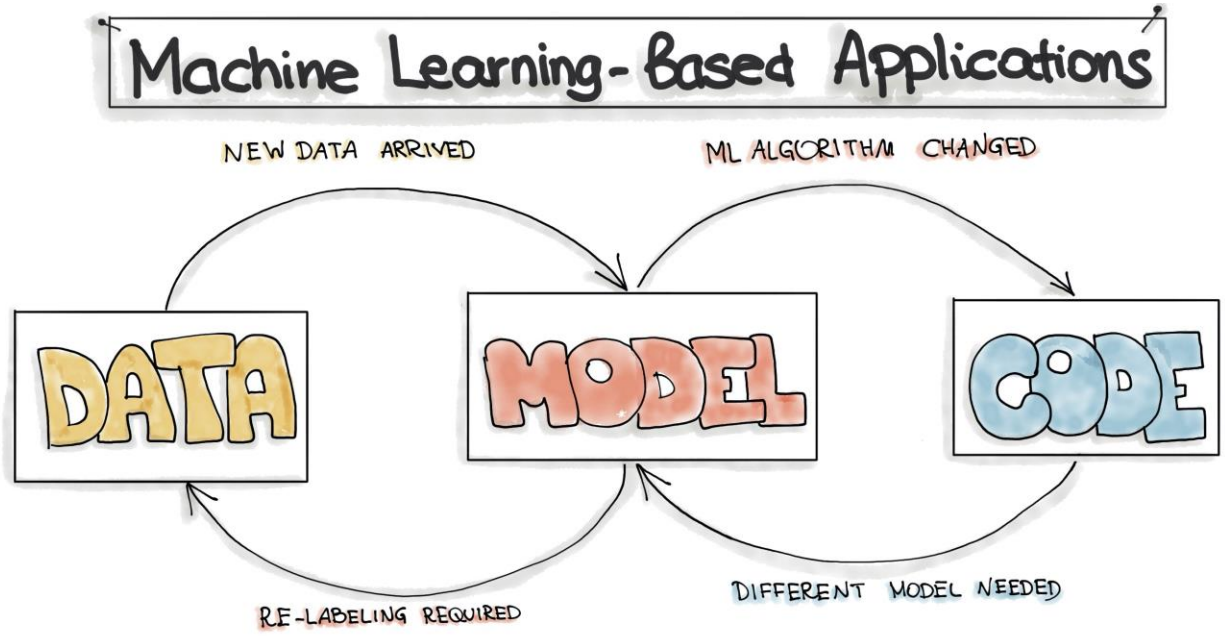
    - Programming Language
        - The programming language is another crucial component of the ML platform
            - the language would you use to develop the ML platform
            - the language your users would perform ML development with
        - Python
            - The most popular language with strong community support
            - that would likely ensure you are making your users' workflow efficient would likely be
    - Collaboration
        - One of the most important principles of MLOps that should be integrated into any platform is collaboration
        - The main components here include:
            - ✓ Source control repository
            - ✓ Notebooks and IDEs
            - ✓ Third-party tools and integrations
        - Source code repository
            - ✓ During experimentation, this component lets your data scientists share code bases, work together, peer review, merge, and make changes
            - ✓ A source code repository is used to keep track of code artifacts
            - ✓ like notebooks, scripts, tests, and configuration files that are generated during experimentation
        - Notebooks and IDEs
            - ✓ The notebook is the experimentation hub for data scientists,
            - ✓ there needs to be an agreement on what tools will be ideal for the team long-term—components that will be around in 5–10 years
        - Third-party tools and integrations

- ✓ Sometimes, data team might need to integrate with some external tool that wasn't built with the platform, perhaps due to occasional needs
- ✓ For example, the data team might want to use an external BI tool to make reports

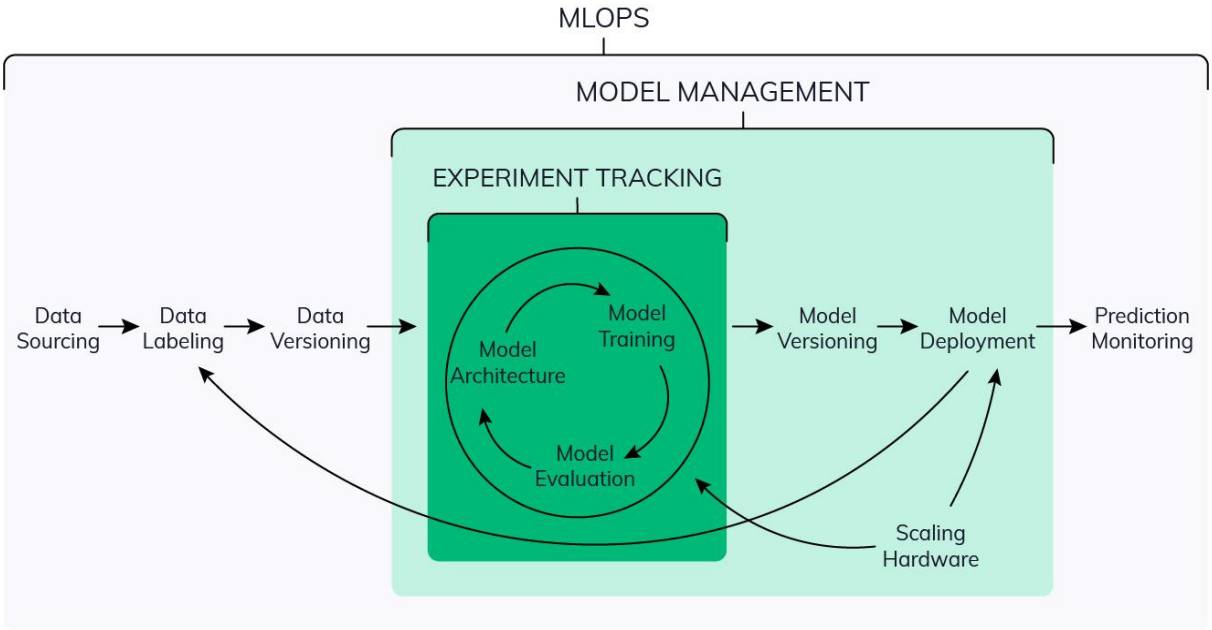
- Libraries and Frameworks
  - This component lets us natively integrate machine learning libraries and frameworks
    - ✓ That users mostly leverage into the platform
    - ✓ Some examples are TensorFlow, PyTorch, and so on.
- Infrastructure and Compute
  - arguably the most important layer to figure out, along with the data component
  - ML platform will run on this layer!
    - ✓ With the different moving parts and components you have seen, it can be quite tricky to tame and manage this layer of the platform
  - The infrastructure layer allows for scalability at both the data storage level and the compute level,
    - ✓ which is where models, pipelines, and applications are run
  - The considerations include:
    - ✓ Are your existing tools and services running on the Cloud, on-prem, or a hybrid?
    - ✓ Are the infrastructure services (like storage, databases, for example) open source, running on-prem, or running as managed services?
    - ✓ These considerations would help you understand how to approach designing platform.

5.3 ML Tools Landscape

Three essential components - Data, ML Model, and Code



- Key phases of MLOps



- MLOps Tools broad categories

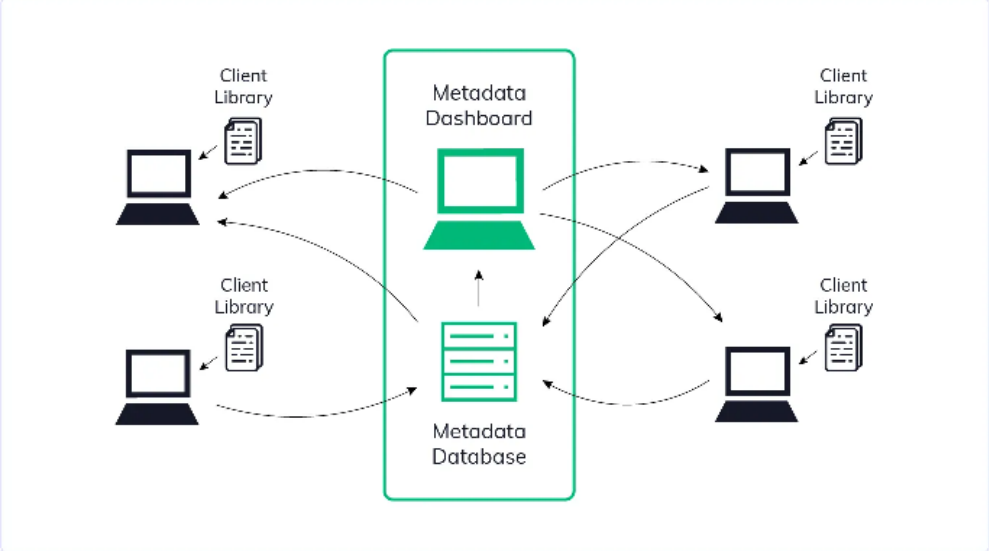
- End to end platform
    - Fully managed services that provide developers and data scientists
        - with the ability to build, train, and deploy ML models quickly.
    - The top commercial solutions are:
        - Amazon Sagemaker, a suite of tools to build, train, deploy, and monitor machine learning models
        - Microsoft Azure MLOps suite:
            - ✓ Azure Machine Learning to build, train, and validate reproducible ML pipelines
            - ✓ Azure Pipelines to automate ML deployments
            - ✓ Azure Monitor to track and analyze metrics
            - ✓ Azure Kubernetes Services and other additional tools.
        - Google Cloud MLOps suite:
            - ✓ Dataflow to extract, validate, and transform data as well as to evaluate models
            - ✓ AI Platform Notebook to develop and train models
            - ✓ Cloud Build to build and test machine learning pipelines
            - ✓ TFX to deploy ML pipelines
            - ✓ Kubeflow Pipelines to arrange ML deployments on top of Google Kubernetes Engine (GKE)

- Custom-built MLOps solution (the ecosystem of tools)
    - End-to-end solutions are great, but one can also build own solution with favorite tools,
        - by dividing MLOps pipeline into multiple microservices
    - Can help to avoid a single point of failure (SPOF), and make pipeline
        - which are robust, easier to audit, debug, and more customizable
    - With a microservices provider is having problems, can easily plug in a new one
        - ensure that each service is interconnected instead of embedded together
        - can have separate tools for model management and experiment tracking
    - Many MLOps tools available, top picks are:
        - Project Jupyter
        - Airflow
        - Kubeflow
        - MLflow
        - neptune.ai

6.1 ML Experiment Tracking

- What is experiment tracking
    - Experiment tracking is the process of saving all experiment related information that you care about for every experiment you run
    - This “metadata you care about” will strongly depend on your project, but it may include:
        - Scripts used for running the experiment
        - Environment configuration files
        - Versions of the data used for training and evaluation
        - Parameter configurations
        - Evaluation metrics
        - Model weights
        - Performance visualizations (confusion matrix, ROC curve)
        - Example predictions on the validation set (common in computer vision)

- Components
    - To do experiment tracking properly, need some sort of a system that deals with all this metadata
    - Typically, such a system will have 3 components:
        - Experiment database: A place where experiment metadata is stored and can be logged and queried
        - Experiment dashboard: A visual interface to your experiment database - A place where can see experiment metadata
        - Client library: Which gives methods for logging and querying data from the experiment database
    - Of course, you can implement each component in
    - many different ways, but the general picture will be very similar



- Why does experiment tracking matter?

  4 ways experiment tracking can make workflow better

    - One source of truth
        - All experiments and models are in single place
    - Automated process
        - It let us compare experiments, analyze results, and debug model training
    - Better collaboration
        - One can see what everyone is doing, share results and access experiment data programmatically

    - Live monitoring
        - One can see ML runs live and manage experiments from anywhere and anytime

- What should be tracked in ML experiments?

  Things that should be keep track of regardless of the project are:
    - Code:
        - preprocessing, training and evaluation scripts, notebooks used for designing features, other utilities.
        - All the code that is needed to run (and re-run) the experiment
    - Environment:
        - The easiest way to keep track of the environment is to save the environment configuration files like `Dockerfile` (Docker), `requirements.txt` (pip) or `conda.yml` (conda)
        - Can also save the Docker image on Docker Hub, but I find saving configuration files easier
    - Data:
        - saving data versions (as a hash or locations to data files) makes it easy to see what one's model was trained on
        - can also use modern data versioning tools like DVC (and save the .dvc files to one's experiment tracking tool)
    - Parameters:
        - saving one's run configuration is absolutely crucial
    - Metrics:
        - logging evaluation metrics on train, validation, and test sets for every run is pretty obvious

- How to set up experiment tracking
    - Spreadsheets + naming conventions
    - Versioning configuration files with Github
    - Using modern experiment tracking tools

6.3 Model Registry

- An ML model registry serves as a centralized repository, enabling effective model management and documentation

- Allows for
    - clear naming conventions,
    - comprehensive metadata,
    - and improved collaboration between data scientists and operations teams,
    - ensuring smooth deployment and utilization of trained models
- A data scientist can push trained models to the model registry
    - Once in the registry, models are ready to be tested, validated, and deployed to production

- Model registry Vs Model repository
    - Model Repository is a storage location for machine learning models
    - Model Registry is a more comprehensive system that tracks and manages the full lifecycle of machine learning models.
    - However, both are often used interchangeably, and the specific definitions may vary depending on the context or the tools and platforms being used.

- Model registry key features and functionality

  Acts as a centralized storage for effective collaboration
    - Model registry provides a central storage unit that holds models (including model artifacts) for easy retrieval by an application (or service)
    - Without the model registry, the model artifacts would be stored in files that are difficult to track and saved to whatever source code repository is established
    - The centralized storage also enables data teams to have a single view of the status of all models, making collaboration easier

  Bridges the gap between experiment and production activities
    - Model registry acts as a glue between ML experimentation and Operations,
    - enabling model development, software development, and operational teams to collaborate.

- How model registry typically works
    - Model Registration: When a new model is developed or trained, it is registered in the model registry.

- Version Control: The model registry maintains a history of all registered models and their versions.
- Model Comparison: The model registry allows users to compare performance metrics, model architectures, hyperparameters, and other relevant information in different versions of a model.
- Model Tracking: As new versions of the model are developed or trained, they are registered in the model registry as well, incrementing the version number.
- Retention and Archiving: The model registry typically retains older versions of the models, ensuring a complete history and traceability.
- By enabling model versioning, the model registry ensures that different iterations of a model can be stored, tracked, compared, and accessed conveniently.

6.5 Model Metadata

- A metadata system is designed to keep track of what we're doing
    - In the case of features and labels, it should minimally keep track of the feature definitions and the versions used in each model's definitions and trained models
- Most organizations start building their data sciences and ML infrastructure without a solid metadata system
- The next most common approach is to build several metadata systems, each targeted at solving a particular problem
    - make one for tracking feature definitions and mappings to feature stores
    - need a system for mapping model definitions to trained models, along with data about the engineers or teams
    - responsible for those models
    - need a model serving system is also going to need to keep track of trained model versions, when they were put into production
    - need a model quality or fairness evaluation systems will need to be read from all of these systems in order to identify
    - and track the likely contributing causes of changes in model quality or violations of our proposed fairness metrics

- Metadata about experiments and model training runs

- During experimentation, one usually care about debugging, visualizing, monitoring his model training to get to the best model.
- To do that, it is a good practice to log anything that happens during the ML run, including:
    - data version: reference to the dataset, md5 hash, dataset sample to know which data was used to train the model
    - environment configuration: requirements.txt, conda.yml, Dockerfile, Makefile to know how to recreate the environment where the model was trained
    - code version: git SHA of a commit or an actual snapshot of code to know what code was used to build a model
    - hyperparameters: configuration of the feature preprocessing steps of the pipeline, model training, and inference to reproduce the process if needed
    - training metrics and losses: both single values and learning curves to see whether it makes sense to continue training
    - record of hardware metrics: CPU, GPU, TPU, Memory to see how much your model consumes during training/inference
    - evaluation and test metrics: f2, acc, roc on test and validation set to know how your model performs
    - performance visualizations: ROC curve, Confusion matrix, PR curve to understand the errors deeply
    - model predictions: to see the actual predictions and understand model performance beyond metrics
    - …and about a million other things that are specific to one's domain

- Metadata about artifacts
    - Apart from experiments and model training runs, there is one more concept used in ML projects: artifact
        - input or output of those runs can be used in many runs across the project
        - Artifacts can change during the project, and typically have many versions of the same artifact at some point in a ML lifecycle
        - Artifacts could be datasets, models, predictions, and other file-like objects.

- Metadata about trained models
    - Trained models are such an important type of artifact in ML projects

        - Once one's model is trained and ready for production, needs change from debugging and visualization to knowing how to deploy a model package, version it, and monitor the performance on prod.
    - ML metadata may want to log –
        - Model package: Model binary or location to model asset
        - Model version: code, dataset, hyperparameters, environment versions
        - Evaluation records: History record of all the evaluations on test/validation that happened over time
        - Experiment versions: Links to recorded model training (and re-training) runs and other experiments associated with this model version
        - Model creator/maintainer: who build this model, and who should ask if/when things go wrong
        - Downstream datasets/artifacts: references of datasets, models, and other assets used downstream to build a model. This can be essential in some orgs for compliance.
        - Drift-related metrics: Data drift, concept drift, performance drift, for all the "live" production
        - Hardware monitoring: CPU/GPU/TPU/Memory that is consumed in production.
        - And anything else that will let one sleep at night while his model is sending predictions to the world

- Metadata about pipeline
    - One may be at a point where ML models are trained in a pipeline that is triggered automatically:
        - When the performance drops below certain thresholds
        - When new labeled data arrives in the database
        - When a feature branch is merged to develop
        - Or simply every week
    - Need for the metadata is a bit different than for experiments or models
        - This metadata is needed to compute the pipeline (DAG) efficiently:
- Input and output steps: information about what goes into a node and what goes out from a node and whether all the input steps are completed
- Cached outputs: references to intermediate results from a pipeline so that one can resume calculations from a certain point in the graph

7.1 Model Packaging

The process of exporting the final ML model into a specific format

- e.g. PMML, PFA, or ONNX,
- which describes the model to be consumed by the business application

- Why package ML models?
  - ✓ MLOps enables a systematic approach to train and evaluate models
  - ✓ ML doesn't work like traditional software engineering, which is deterministic in nature
  - ✓ Engineering ML solutions is non-deterministic involves serving ML models to make predictions or analyze data
  - ✓ To serve the models, they need to be packed into software artifacts
  - ✓ ML models need to be packaged for the following reasons:
    - Portability
    - Inference
    - Interoperability
    - Deployment agnosticity

- How to package ML models
  - ML models can be packaged and shipped in three ways
    - Serialized files
      - Serialization is a vital process for packaging an ML model - enables model portability, interoperability, and model inference
      - Serialization is the method of converting an object or a data structure (for example, variables, arrays, and tuples) into a storable artefact, for example, into a file or a memory buffer
        - that can be transported or transmitted (across computer networks)
        - only saves the data structure as it is in a storable artefact such as a file
      - The main purpose of serialization is to reconstruct the serialized file into its previous data structure in a different environment
        - for example, a serialized file into an ML model variable
      - A newly trained ML model can be
        - serialized into a file
        - exported into a new environment

- can de-serialized back into an ML model variable or data structure for ML inferencing

Sr. No.	Format	File extension	Framework	Quantization
1	Pickle	.pkl	scikit-learn	No
2	HD5	.h5	Keras	Yes
3	ONNX	.onnx	TensorFlow, PyTorch, scikit-learn, caffe, keras, mxnet, IoS Core ML	Yes
4	PMML	.pmml	scikit-learn	No
5	Torch Script	.pt	PyTorch	Yes
6	Apple ML model	.mlmodel	IoS core ML	Yes
7	MLeap	.zip	PySpark	No
8	Protobuf	.pb	TensorFlow	Yes

Popular ML model serialization formats

- Packetizing or containerizing
    - Every environment possesses different challenges when it comes to deploying ML models,
    - A container is a standard unit of software made up of code and all its dependencies
    - Docker - industry standard at developing and orchestrating containers
- Microservice generation and deployment

7.2 ML Model Serialization Formats

ML Model Serialization Formats

- Various formats to distribute ML models
- In order to achieve a distributable format, the ML model should be present and should be executable as an independent asset
    - For example, might want to use a Scikit-learn model in a Spark job - means that the ML models should work outside of the model-training environment
- Two broad exchange formats for ML models.
    - Language-agnostic
    - Vendor-specific

- Language-agnostic exchange formats

- Amalgamation
    - simplest way to export an ML model - model and all necessary code to run are bundled as one package
    - usually, a single source code file that can be compiled on nearly any platform as a standalone program
    - straightforward concept, and the exported ML models are portable
    - For example,
        - can create a standalone version of an ML model by using SKompiler
        - python package provides a tool for transforming trained Scikit-learn models into other forms,
        - such as SQL queries, Excel formulas, Portable Format for Analytics (PFA) files, or SymPy expressions
- PMML
    - a format for model serving based on XML with the file extension .pmml
    - has been standardized by the Data Mining Group (DMG)
    - Basically, .ppml describes a model and pipeline in XML
    - does not supports all of the ML algorithms
    - usage in open source-driven tools is limited due to licensing issues

- PFA (Portable Format for Analytics)
    - designed as a replacement for PMML
    - To run ML models as PFA files, will need a PFA-enabled environment
- ONNX (Open Neural Network eXchange)
    - an ML framework independent file format
    - was created to allow any ML tool to share a single model format
    - is supported by many big tech companies such as Microsoft, Facebook, and Amazon
- YAML (Yet Another Markup Language)

    - YAML is used to package models as part of the MLFlow framework for ML pipelines on Spark

- Vendor specific Exchange formats
    - Scikit-Learn
        - saves models as pickled python objects, with a .pkl file extension
    - H2O
        - allows to convert the models built to either POJO (Plain Old Java Object) or MOJO (Model Object, Optimized)
    - SparkML
        - models can be saved in the MLeap file format and served in real-time using an MLeap model server
        - supports Spark, Scikit-learn, and Tensorflow for training pipelines and exporting them to an MLeap Bundle
        - The MLeap runtime is a JAR that can run in any Java application
    - TensorFlow
        - saves models as .pb files; which is the protocol buffer files extension
    - PyTorch
        - serves models by using their proprietary Torch Script as a .pt file
        - model format can be served from a C– application
    - Keras
        - saves a model as a .h5 file, which is known in the scientific community as a data file saved in the Hierarchical Data Format (HDF)
        - file contains multidimensional arrays of data
    - Apple
        - has its proprietary file format with the extension .mlmodel to store models embedded in iOS applications
        - The Core ML framework has native support for Objective-C and Swift programming languages
        - Applications trained in other ML frameworks, such as TensorFlow, Scikit-Learn, and other frameworks need to use tools like such as coremltools and Tensorflow

converter to translate their ML model files to the .mlmodel format for use on iOS

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-read-able	Compression
"almagi-nation"	–	–	–	–	–	–	✓
PMML	✓	DMG	.pmml	AGPL	R, Python, Spark	✓ (XML)	✗
PFA	✓	DMG	JSON		PFA-en-abled runtime	✓ (JSON)	✗
ONNX	✓	SIG	.onnx		TF, CNTK,	–	✓

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-read-able	Compression
TF Serving Format	✓	Google	.pf		Tensor Flow	✗	g-zip
Pickle Format	✓		.pkl		scikit-learn	✗	g-zip
JAR/ POJO	✓		.jar		H2O	✗	✓
HDF	✓		.h5		Keras	✗	✓
MLEAP	✓		.jar/ .zip		Spark, TF, scikit-learn	✗	g-zip
Torch Script	✗		.pt		PyTorch	✗	✓
Apple .mlmodel	✗	Apple	.mlmodel		TensorFlow, scikit-learn,	–	✓

-

ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Named Entity Recognition

PhD Study Report

Michal Konkol

Technical Report No. DCSE/TR-2012-04
June, 2012

Distribution: Public

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitní 8
30614 Pilsen
Czech Republic

Contents

1 Introduction

"The lurking suspicion that something could be simplified is the world's richest source of rewarding challenges."
Edsger Dijkstra

Named Entity Recognition (NER) is one of the important parts of Natural Language Processing (NLP). NER is supposed to find and classify expressions of special meaning in texts written in natural language. These expressions range from proper names of persons or organizations to dates and often hold the key information in texts.

NER can be used for different important tasks. It can be used as a self-standing tool for full-text searching and filtering. Also it can be used as a preprocessing tool for other NLP tasks. These tasks can take advantage of marked Named Entities (NE) and handle them separately, which often results in better performance. Some of these tasks are Machine Translation, Question Answering, Text Summarization, Language Modelling or Sentiment Analysis.

NER task was firstly introduced at MUC-6 in 1995. Since that time it has moved from rule-based systems to statistical systems with variety of advanced features. The state-of-the-art performance is around 90% for English and 70% for Czech. The performance for other languages greatly varies depending on properties of a given language. It is thus very important to find new approaches to fill this gap in performance between different languages. Following example shows a typical output of a NER system.

```
<organization>The European Union</organization> was formally
established when the <other>Maastricht Treaty</other> came
into force on <date>1 November 1993</date>.
```

1.1 Outline

The second chapter will be devoted to measures for NER. We believe that it is important to define the evaluation techniques before some results are presented.

The third chapter describes the factors that may influence the performance of NER system. These factors are usually given and independent on the method. It is important that the results must be interpreted in context and not only as absolute numbers.

The fourth chapter introduces the methods that are used for NER. For each of these methods one or two examples of a NER system are given.

The fifth chapter covers the most used features for NER. The features seems to have at least the same importance as the methods.

The last chapter summarizes the open challenges of NER task.

2 Measures

"Measure what is measurable, and make measurable what is not so."
Galileo Galilei

This chapter will be devoted to performance measurement. In any area of research it is important to evaluate and compare results of new methods. There is thus a need to use some objective measure (or measures), which would well cover the purpose of the research.

Unlike other NLP tasks (e.g. Machine Translation) NER uses only one standard method of measurement, which is generally accepted. This method uses three metrics to describe the performance of NER system, each for different aspect of the task. These metrics are called precision, recall and F-measure (also F-score or F_1 score).

We will define these measures on a general classification of objects into two classes; positive and negative. Then there exist four following classes of classification results.

- Positive (P) - positive object marked as positive.
- Negative (N) - negative object marked as negative.
- False positive (FP)- negative object marked as positive.
- False negative (FN) - positive object marked as negative.

This is well shown on fig. 2.1, where the curves show the distribution of positive and negative objects, the dotted line shows the decision threshold of classifier. In the areas marked as FN and FP are some objects marked incorrectly.

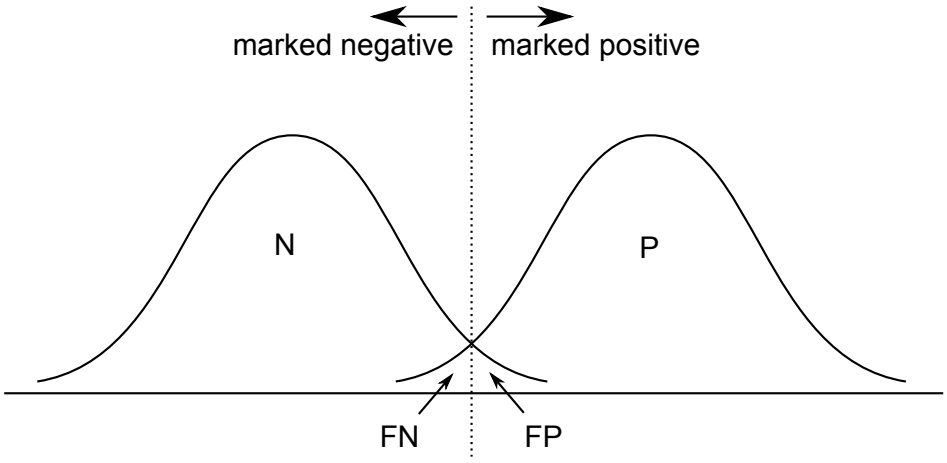


Figure 2.1: Precision and recall

Now we can easily define precision, recall and F-measure as follows.

$$\text{Precission} = \frac{P}{P + FP}$$

$$\text{Recall} = \frac{P}{P + FN}$$

$$\text{F-measure} = \frac{2P}{2P + FP + FN}$$

Precision is a measure of trust, that the objects marked as positive are really positive. Recall is a measure of trust, that all the positive objects are marked. It is obvious that precision and recall describe different aspects of results. Moreover, these measures are competing. As shown on fig. 2.2, if the decision threshold is moved to the left, there will be fewer FN objects and more FP objects, resulting in high recall and lower precision. This is important in evaluation of a classifier, because high recall (resp. precision) classifier can be better for various tasks. F-measure is a harmonic mean between precision and recall and is something like overall perspective.

The last piece of information needed for NER performance evaluation is to define, what is counted as P, N, FP and FN. This definition slightly differs between NER conferences. The following sections will describe evaluation

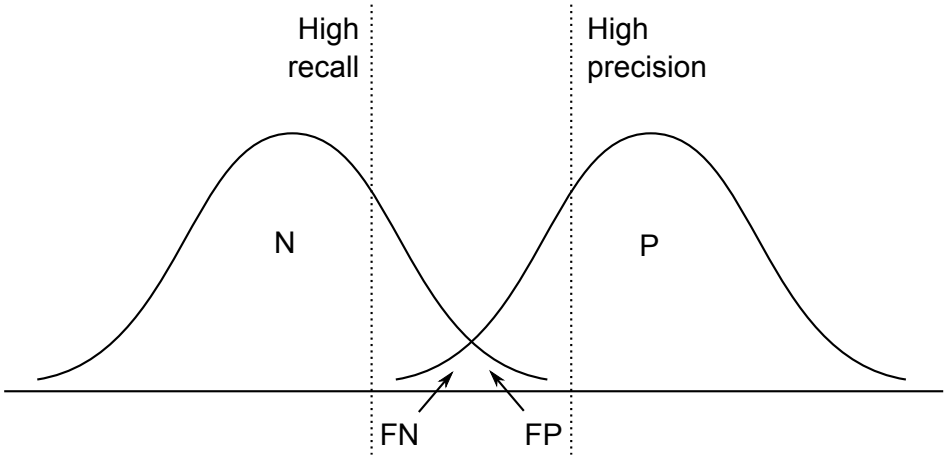


Figure 2.2: Precision and recall change

methods used on major NER conferences and also the evaluation method used in this thesis.

2.1 MUC-6 Evaluation

The NER task was introduced at MUC-6[1]. So the initial evaluation technique was defined at this conference. The choice of evaluation metrics was based on other information retrieval tasks. Since that time precision, recall and F-measure are used as a standard in NER.

At MUC-6 span (or text) and type of the entity was handled separately. The type is counted as correct, if it is the same as the original type and the span overlaps the entity. The text is considered correct, if it matches the original text of the entity. The text comparison involves operations like trimming or removing unimportant parts (ltd., the, etc.). Each entity can have an alternative text, which is also considered as correct.

Three numbers were counted for both type and span: COR (correct answers), POS (number of original entities) and ACT (number of guesses). The overall results of the system were acquired by adding these number for type and span together. The precision, recall and F-measure were then computed in a standard way using these numbers.

The evaluation techniques for all MUC tasks are covered in The Message Understanding Conference Scoring Software User's Manual [1].

2.2 CoNLL Evaluation

The CoNLL 2002[2] and 2003[3] have used an exact match evaluation. The entity is considered correct only if it has exactly the same span and type.

The advantage of this method is, that it is clear, simple and gives a lower estimate of the evaluated system. The disadvantage is, that in some cases it is too strict. If the original entity is "The United States" and "United States" is marked by the system, then "The United States" is considered as FN and "United States" as FP. The result is, that the system is penalized in two ways for almost good answer.

2.3 ACE Evaluation

The Automatic Content Extraction program consists of various NLP tasks. There are two tasks directly focused on NER, Entity Detection and Tracking (EDT)[4] and Time Expression Recognition and Normalisation (TERN)[5]. Both tasks extends the standard definition of NER tasks with deeper level of detail.

The evaluation of EDT task did not used standard metrics. The evaluation is based on a special scoring system, where each type of error and also each type of entity has different weight. The scoring system is very complex. On one hand, it can be adjusted and used to properly evaluate systems regarding various needs. On the other hand, the weights must be the same to compare two systems and it is hard to get direct feedback.

[1] http://www-nlpir.nist.gov/related_projects/muc/muc_sw/muc_sw_manual.html

2.4 Our evaluation

Each entity is defined by two attributes: type and span. Both these attributes are important, but in many cases, you prefer to have correct type. Sometimes the span of entity is hard to guess even for humans. This can be exemplified on entities in table 2.1, where not correctly marked span gives important information.

Entity	Marked as
Západočeská univerzita v Plzni	Západočeská univerzita
University of West Bohemia in Pilsen	*University of West Bohemia*
Kongres Spojených států amerických	Kongress
United States Congress	Congress
IBM Česká republika	IBM
IBM Czech Republic	*IBM*

Table 2.1: Examples of entities marked with not correct span, which can give a valid information.

This is the motivation to use extended model, where the partially good span can be taken into account. In this thesis, we categorize entities into the following bins. It is not necessary to keep track of N, because it is not needed for our measures.

- Correct - the entity is marked on correct span with correct type.
- Partially correct - the entity is marked with correct type, but the span is not exact.
- Not correct - something is marked, but it is not an entity.
- Not marked - entity is not marked.

We then use these bins to compute two versions of all previously mentioned measures. These versions differ in the way, how the bins are mapped to P, FP and FN. In both versions correct is mapped to P, not correct to FP and not marked to FN. The first version is strict and maps the partially good to FP. The second version is lenient and maps partially good to P. All is summarized in the following formulas.

Strict

$$\text{Precission} = \frac{Correct}{Correct + Not\ correct + Partially\ correct}$$

$$\text{Recall} = \frac{Correct}{Correct + Not\ marked}$$

$$\text{F-measure} = \frac{2 \cdot Correct}{2 \cdot Correct + Not\ correct + Not\ marked + Partially\ correct}$$

Lenient

$$\text{Precission} = \frac{Correct + Partially\ correct}{Correct + Not\ correct + Partially\ correct}$$

$$\text{Recall} = \frac{Correct + Partially\ correct}{Correct + Not\ marked}$$

$$\text{F-measure} = \frac{2 \cdot (Correct + Partially\ correct)}{2 \cdot (Correct + Partially\ correct) + Not\ correct + Not\ marked}$$

3 Performance influencing factors

"Success is a science; if
you have the conditions,
you get the result."
Oscar Wilde

There are many factors that can radically change the performance of a NER system. The most important of these factors are the language and domain of the processed texts and the information we are looking for.

3.1 Language

The language itself is obviously one of the most important factors. The first systems based on rules were build for a specific language and it was not possible to easily alter them to different language. With the advent of systems based on machine learning, it was possible to choose features independent on the language and use the system for different languages. The performance of the systems is significantly affected by the language and for some languages the difference in performance is more then 20% [6].

The majority of systems were logically created for English. But many languages have at least some experiments with state-of-the-art methods. The results for various languages are presented in tab. 3.1, but keep in mind that these results are affected by several conditions.

We are obviously interested in Czech. So far, there are only two published results [10][15]. The best performance for all entities using similar level of detail as CoNLL is F-measure 71%. There is a big gap between state-of-the-art English and Czech NER. Even for many other languages the difference is quite big.

Language	F-measure
Arabic [7]	79.21 %
Bulgarian [8]	89 %
Chinese [9]	≅ 90 %
Czech [10]	71 %
Dutch [11]	77.05 %
English [12]	88.76 %
German [12]	72.41 %
Greek [13]	71–94 %
Hungarian [14]	91.95 %
Spanish [11]	81.39 %

Table 3.1: Results of NER for various languages.

3.2 Domain

The domain of corpora can highly influence the performance of NER system. Some domains seems to be easier for NER then others, e.g. news articles and texts from social networks.

The systems are usually trained on one domain and it would be desirable to directly use them for other domains. Unfortunately, a fundamental performance degradation was detected if the domains are slightly different. Drop in F-measure from 90.8% (CoNLL) to 64.3% (Wall Street Journal) was reported in [16]. Similar performance degradation was reported in [17] for NER trained on MUC-6 corpus and used for more informal texts like emails. The domain adaptation problem is one of the challenges of NER [18].

3.3 Searched entities

Another important aspect are the types of entities we are looking for. Some categories of NEs are easier to find then others, e.g. countries are easier then organizations [2][3]. Of course, this depends on the definition of the class.

For example the datetime class can be defined very strictly to contain only absolute dates (2007; 5.3.2001; June 5, 2004) or it can include relative dates (next Saturday, in December), obviously the second case is harder. This can be applied to other classes more or less.

There also exist different levels of detail. On one hand, some authors use coarse-grained categories. The annotation scheme defined at MUC-6[1] has three types of NEs, where each category has two or three subtypes. At CoNLL[2][3], there were only four categories. On the other hand, some authors have defined many detailed categories. A finer-grained categories were used by BBN for Question Answering[19]. A hierarchy of 150 (later 200) categories was proposed by [20]. The Czech named entity corpus defines ten categories and 62 subcategories [15]. Generally, it is harder to classify entities into more categories [20][15].

The common NE categories are Person, Organization, Location (GPE), Date (and time), Numbers (of different kinds) and Miscellaneous. Another branch of NER is focused on biology and thus uses categories like Protein, DNA etc. [21]

4 Methods

"An algorithm must be seen to be believed."
Donald Knuth

Many different approaches were used for NER. This chapter is an attempt to choose the most important methods and describe their usage, training and other properties.

4.1 Method division

There are various aspects which can be used to divide or describe the methods. The divisions presented in the following paragraphs are compilation of notations used in NER or generally NLP field. They are not rigid, but they should give some idea about the methods.

All the methods have two basic development steps, creation and usage. The first division is based on creation phase. We say that the system is *handcrafted*, if all the parameters (in this meaning parameter can be even a rule) of the system are made or set by human. The system is denoted as *machine learning* system, if the parameters are somehow estimated by a computer.

The machine learning systems can be further divided based on the data they need to find the parameters. If the system needs corpus with already marked entities, then the system uses *supervised learning*. If the system needs only small amount of marked examples and then tries to improve the performance using unmarked text, then it uses *semi-supervised learning*. The last option is *unsupervised learning* which estimates the parameters from unmarked text.

The methods can be also divided by their usage to *deterministic* and *stochastic*. The difference between these groups is simple, the stochastic models are based on probability distributions while the deterministic are not. The difference can be seen on the results of these methods. The stochastic methods assign a set of labels and their probabilities for each word, while the deterministic methods assign for each word only one label.

There is also a term *rule-based* system. This term suggests another division based on how exactly is the NER task executed. Rule-based systems recognizes NEs by applying rules. The other category would be for example classification-based, but this term is not commonly used.

In many cases not all of the presented terms are mentioned. For example when a rule-based system is mentioned, it is automatically considered as hand-crafted deterministic rule-based system. Similarly if we talk about a system based on some classification method, we consider that it uses some machine learning method to find the parameters for the classifier.

4.2 Rule-based Methods

4.2.1 Dictionaries

The simplest method which can be used are dictionaries. NER based on dictionaries tries to find NE in the dictionary for each word (or group of words). If it finds some NE, then it is marked. Dictionaries of NEs are often called *gazetteers*.

The performance of this method corresponds to its simplicity. Even a very large gazetteers contains only a small portion of all used NEs. NEs are also used in various forms (e.g. Pilsner University), not only the basic form (e.g. University of West Bohemia), so it is necessary to enumerate all possible forms in the dictionary. For inflectional languages it is necessary to enumerate all word forms or to use lemmatization or stemming.

This method is not usually used separately, but is often used as part in more complex systems. Extended version of this method was used in [22].

4.2.2 Regular Expressions

One of the most widely used tools for text processing (not only NER) are regular expressions (RE). REs are a grammar classified as regular in Chmosky hierarchy. That means that they can be processed by finite state automaton and that their processing is very fast. The grammar itself slightly differs from one implementation to another and will not be described here.

REs were used as a part of many systems and some of the simpler rule based systems can be built only on REs.

4.2.3 Context Free Grammars

Context Free Grammars (CFG) are more general level of grammar than regular expressions in Chomsky hierarchy. CFG can create more complex rules then RE and thus have advantage for rule-based systems. On the other side, the complex rules written in CFG cannot be applied as machine learning feature without loosing adaptability. The more complex rules tend to be more dependent on a particular domain or preprocessing tool (e.g. tokenizer).

CFG have been used in many systems in the rule based era of NER. Examples of these systems can be found on MUC-6 and MUC-7 conferences [23][24].

4.3 Statistical Methods

Statistical methods for NER are modelling the probability distribution $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{y}$ is the sequence of NE classes and $\mathbf{x}$ is the sequence of words. Some methods classify each word separately, other methods are classifying the whole sequence.

There are two different approaches to classification called generative and discriminative. The generative classifiers are based on the Bayes theorem (4.1) and are modelling $p(x|y)$ and $p(x)$. The typical generative classifier is Naive Bayes. The discriminative approach models directly $p(y|x)$ and one of the typical examples is Maximum Entropy.

$$p(y|x) = \frac{p(x|y)p(x)}{p(y)} \tag{4.1}$$

It is also necessary to adopt some model for handling multi word NEs. All the described methods assign probabilities of labels to each word. If there are two words with the same assigned class, then by using simple labelling (one label for one class) it is not possible to distinguish, if the second word is start of another entity or continuation of the first. An example in Czech follows (in this case, the English translation does not have the same problem).

Ukázal dopis od Petra Pavlovi.
He showed the letter from Peter to Paul.

The easiest way to handle it is to ignore it. It is not very common, that one entity of the same type directly follows another (of course it depends on language). So it is possible to mark all consecutive occurrences as one entity. Another way is to encode one entity type with multiple labels. A common way is to use BIO (or IOB) model [2][25], where B stands for begin, I for inside and O for outside. All NE classes then have two labels (e.g. person_B, person_I, city_B) and the O is for non-entity class. Some authors are extending BIO model with E (end) tag [26].

The following sections will describe the most used classification methods.

4.3.1 Hidden Markov Models

Markov Models are modelling a Markov process and are based on a state graph. Markov process is a stochastic process for which the state transmission probability distribution depends only on the present state. Hidden Markov Models are modelling a process, where the states are not directly observable. Hidden Markov Model is fully described by following properties.

- $X = \{x_1, \ldots, x_n\}$ – Set of observations.
- $Y = \{y_1, \ldots, y_m\}$ – Set of states.
- y_0 – Initial state.

- $p(y_k|y_{k-1})$ – The state transition probability distribution.
- $p(x|y_k, y_{k-1})$ – The observation emission probability distribution.

For NER the states are the NE classes and observations are words. The Viterbi algorithm is then used to find the sequence of states (NE classes) with highest probability. The Baum-Welch algorithm can be used to improve the parameters of HMM using unmarked texts. A typical example of a HMM system is [27]. HMM were only used in combination with other classifier at CoNLL 2003 [3]. Recent systems often prefer Conditional Random Fields, which have similar ability to handle sequences.

4.3.2 Support Vector Machines

The Support Vector Machines will be described in the simplest possible way following the original description [28]. We will assume only binary classifier for classes $y = -1, 1$ and linearly separable training set $\{(x_i, y_i)\}$. It means that the conditions (4.2) are met.

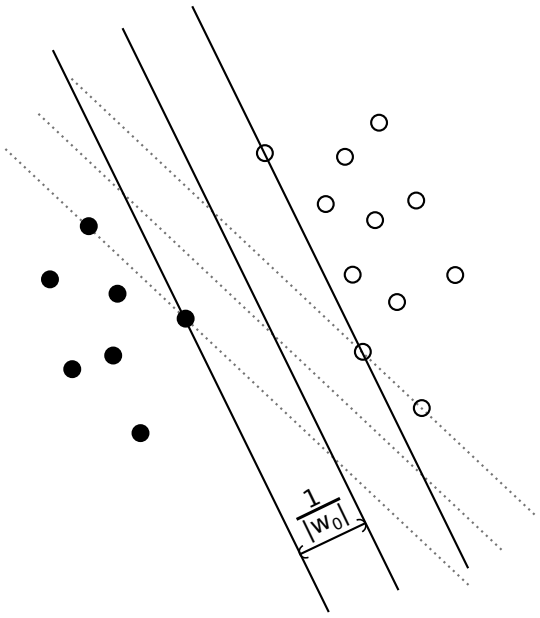


Figure 4.1: Optimal (and suboptimal) hyperplane.

$$\begin{array}{rcrll} \mathbf{w}\cdot\mathbf{x_i}+b & \leq & -1 & \text{if} & y_i = -1 \\ \mathbf{w}\cdot\mathbf{x_i}+b & \geq & 1 & \text{if} & y_i = 1 \end{array} \tag{4.2}$$

Thanks to the choice of y labels, we can rewrite the conditions (4.2) in one equation (4.3) that covers all objects in the training set.

$$y_i \cdot (\mathbf{w}_0 \cdot \mathbf{x} + b_0) \geq 1 \tag{4.3}$$

SVM are based on the search of the optimal hyperplane (4.4) that separates both classes with the maximal margin. We need to measure the distance between the classes in the direction given by $\mathbf{w}$. The formula for this is (4.5).

$$\mathbf{w}_0 \cdot \mathbf{x} + b_0 = 0 \tag{4.4}$$

$$d(\mathbf{w}, b) = \min_{x;y=1} \frac{\mathbf{x}\cdot\mathbf{w}}{|\mathbf{w}|} - \max_{x;y=-1} \frac{\mathbf{x}\cdot\mathbf{w}}{|\mathbf{w}|} \tag{4.5}$$

The optimal hyperplane maximizes the distance $d(\mathbf{w}, b)$ and can be expressed as (4.6). Therefore the parameters $\mathbf{w}_0$ and b_0 can be found by maximizing $|\mathbf{w}_0|$. For better orientation the optimal hyperplane (and also one suboptimal) is shown on figure 4.1.

$$d(\mathbf{w}_0, b_0) = \frac{2}{|\mathbf{w}_0|} \tag{4.6}$$

The classification is then done by looking on which side of the hyperplane the object is. Mathematically written as (4.7).

$$l(\mathbf{x}) = \text{sign}(\mathbf{w}_0 \cdot \mathbf{x} + b_0) \tag{4.7}$$

The best presented result for Czech NER was achieved with SVM [10]. SVM are also often used in systems which combine multiple classifiers [29], because they are able to generalize very well and the principle is quite different from other methods.

4.3.3 Maximum Entropy

The most uncertain probability distribution is the uniform one, because then everything has the same probability. If some constraints are added to the model, the model has to be modified to satisfy these constraints, but there is infinite number of probability distributions satisfying them. The principle of maximum entropy [30] says that the best distribution is the most uncertain one subject to the constraints. A constraint is given in the following form.

$$E_{\tilde{p}}(f_i) = E_p(f_i) \tag{4.8}$$

Where $E_{\tilde{p}}(f_i)$ is the expected value of feature f_i observed from data and $E_p(f_i)$ is the expected value of maximum entropy model. The features are in the following form.

$$f(x,y) = \begin{cases} 1 & \text{if } y \text{ is PERSON and } x \text{ starts with capital letter} \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Where parameter y is a class of a NE and x is the classified object, in our case word (or lemma). It is not necessary to have only binary features, but all feature values have to be positive.

For named entity recognition we want to find conditional probability distribution $p(y|x)$, where y is class of word and x are words used for classification. Following the principle of maximum entropy we want $p(y|x)$ to have maximum entropy of all possible distributions.

$$\arg\max_{p(y|x)} H(p(y|x)) = -\sum_{x\in\Omega} p(x) \sum_{y\in\Psi} p(y|x) \log p(y|x)$$

Because $H(p(y|x))$ is a concave function, there is only one maximum. It can be shown by Lagrange method [31] that the best probability distribution has the following parametric form.

$$p(y|x) = \frac{1}{Z(x)} \exp \sum_{i=1}^{n} \lambda_j f_j(x,y) \tag{4.10}$$

$$Z(x) = \sum_{y} \exp \sum_{i} \lambda_i f_i(x, y)$$

$Z(x)$ is only a normalizing factor which ensures that $p(y|x)$ is a probability distribution. $\Lambda = \{\lambda_0, \ldots, \lambda_n\}$ are parameters and have to be set properly to gain maximum entropy. The parameters are found using Generalized Iterative Scaling[32], Improved Iterative Scaling[31], Limited memory BFGS[33][34] or other minimization method[35].

ME is one of the most popular and successful methods. On CoNLL 2003 five of 16 systems used ME [3]. A typical pure ME classifier is presented in [36].

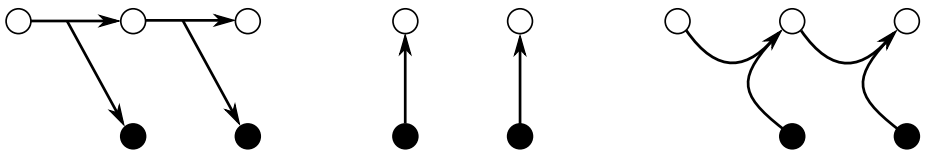
4.3.4 MEMM

A Maximum Entropy Markov Model [37] is a combination of Maximum Entropy and Markov Models. The motivation is to get the best of both methods. The HMM's ability to find sequences and ME's ability to use a lot of diverse features. In other words, we want to model probability distribution $p(y|x, y')$ using the Maximum Entropy principle. This can be achieved by splitting the problem into probability distributions $p_{y'}(x|y)$, creating one ME classifier (4.11) for each previous class.

$$p_{y'}(x|y) = \frac{1}{Z_{y'}(x)} \exp \sum_{i=1}^{n} \lambda_j f_j(x, y) \tag{4.11}$$

The transformation of dependencies can be seen on fig. 4.2. The black points are observations and the white ones are states or labels. The HMM uses a generative approach and models two distributions for state transition and observation emission. The ME uses a discriminative approach, but cannot exploit the Viterbi or Baum-Welch algorithm. The MEMM is a discriminative method that can use altered Viterbi and Baum-Welch method.

So far, MEMM seems to have only advantages. There is also one very important disadvantage that is data sparseness. While in ME data are used to train one classifier, in MEMM the data has to be split and used for $|y|$ classifiers, where $|y|$ is number of labels.



(a) (b) (c)

Figure 4.2: Dependency graphs for (a) HMM, (b) ME nad (c) MEMM.

4.3.5 Conditional random fields

Conditional Random Fields (CRF) were introduced in [38]. The idea of CRF is strongly based on ME. The difference is that ME classifies one instance after another while CRF classify the whole sequence at once. Mathematically written, ME estimates $p(y_i|x_i)$ for $i = 1, \ldots, n$ and CRF estimate $p(\mathbf{y}|\mathbf{x})$ where $\mathbf{y}$ and $\mathbf{x}$ are n-dimensional vectors. The probability $p(\mathbf{y}|\mathbf{x})$ can be computed using matrices and a variant of forward-backward algorithm.

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_j \lambda_j f_j(\mathbf{y}, \mathbf{x})\right) \tag{4.12}$$

The features are extended and can use the previous state in contrast to ME. Two types of features are used, state s and transition t. The state features can be considered as a subset of transition features, where the previous state is not used, and a general feature definition (4.14) can be used.

$$\{f(y_{i-1}, y_i, \mathbf{x}, i)\}_1^{k+l} = \{s(y_i, \mathbf{x}, i)\}_1^k \cup \{t(y_{i-1}, y_i, \mathbf{x}, i)\}_1^l \tag{4.13}$$

Following the dependency graphs from 4.2 we can see the dependency graph for CRF on fig. 4.3. The parameters of this model are found using similar methods like ME, e.g. L-BFGS.

Initial tests on the NER task was done in [39]. Since their introduction, many systems used them with very good results[8][7]. CRF are considered to be the most successful classification method for NER.

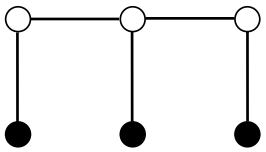


Figure 4.3: Dependency graph for CRF.

4.4 Semi- and unsupervised methods

Semi-supervised methods are usually based on a small set of training examples for each NE class. These examples are then used to find patterns or contexts around them. New NEs are then found using these patterns and contexts. This process is then iterated and the new NEs are always used as examples for the next step. This method is called *bootstrapping*.

Unsupervised methods are typically based on clustering. The clustering methods can use contexts, patterns etc. and rely on a large corpora. It is often difficult to find the boundary between these two classes.

One of the systems using bootstrapping is [22]. It uses small set of NEs for each class as a seed for automatic gazetteer generation and list-lookup strategy (extended by heuristics). Another system in this category is [40].

As the previous examples show, semi-supervised methods are often related with the problem of automatic gazetteer generation. There are many various ways how to deal with this task [41][42].

4.5 Method combinations

In this section we will talk about how to combine different classification methods to improve the results of a solo classifier. In [13] a two passes algorithm is used to identify NEs. In both phases separate classifier is used for each class. The results of the first phase are used as features in the second phase.

Some authors have used a method similar to the one described earlier

[43][44]. The method is called *stacking* and its principle is that output of one classifier is taken as input to another one. Multiple classifier can be combined this way.

Another way to combine classifiers is *voting*. There are two types of voting. The first one [45] is called *majority voting* and uses the final results, i.e. one class of NE is assigned to each word. The second one [45] is called *weighted voting* and uses the raw output of classifiers, i.e. probabilities for all classes are assigned to each word. Both of these types can use weights α_i for classifiers $p_i(y|x)$. The α_i parameters can be found using some form of Expectation-Maximization algorithm.

$$p(y|x) = \alpha_1 p_1(y|x) + \ldots + \alpha_n p_n(y|x) \tag{4.14}$$

In [46] and [47] an ensemble of classifiers is used. All the classifiers uses the same classification method but different feature set. Genetic algorithms (GA) based classifier selection is used to find the best set of classifiers. GA are also used to choose between majority and weighted voting. Similar approach is also used in [29], but the problem is taken as multi-objective optimization.

In general, classifier combinations give better results then the individual classifiers. When dealing with NER task, it is advisable to use some combination of classifiers based on different approaches or feature sets.

5 Features

"What we observe is not nature itself, but nature exposed to our method of questioning."
Werner Heisenberg

If a machine learning approach to NER is used, the features are something like the senses for human. That is why choosing the right feature set has the highest importance. From the beginning of the NER task various features have been used. In the following paragraphs and sections we will try to describe some terms used in connection to features and the most important features.

At first we will focus on the context that the feature uses. Commonly, two categories are used, *local* and *global*. Local features use only a small neighbourhood of the classified word, while global features uses the whole document or corpus. Sometimes some meta-information about the document is also considered as global feature.

Another important property of the feature is a language dependency. The feature is *language independent*, if it can be used for another language without any changes. If we want to create a language independent NER, we obviously need to use only language independent features and methods.

Sometimes a term *external* is used to indicate a feature which uses an external system or information source (e.g. Wikipedia). Gazetteers and dictionaries are sometimes considered external, but more often have their own category, *dictionary* or *list-lookup* features.

Usually, the features are acquired for some small neighbourhood (window) of the classified word. This window is often $-2, \ldots, +2$ or $-3, \ldots, +3$.

5.1 Local features

5.1.1 Orthographic features

Orthographic features are based on the appearance of the word, e.g. the first letter is a capital letter, all letters are capital or the words consists of digits. These features are used very often [3], because they need only the word, are language independent and still very effective for many languages. Interesting approach for these features was presented in [16].

5.1.2 Affixes

Another language independent feature are affixes. Some types of NEs often share the same word ending or prefix. Only a small portion of affixes are meaningful and thus some kind of threshold or feature selection is needed to choose a reasonable number of affixes.

5.1.3 Word

A word itself can be used as a feature. In many cases all letters are converted to upper or lower case to capture a word at the start of a sentence as the same feature as in the middle[45].

5.1.4 Stemming and lemmatization

Stemming is a task which is trying to find the root of each word, resp. to remove all affixes. A stem can be used directly as a feature similarly to a simple word feature. Stemming can also improve the performance of other features, e.g. gazetteers.

Lemmatizetion is a task similar to stemming but the output is a lemma instead of a stem. Lemma is basic word form of a word often used as a dictionary entry.

The importance of stemming and lemmatization is influenced by the lan-

guage. For highly inflectional languages like Czech, stemming or lemmatization is almost a must because it is necessary to reduce the high number of different word forms.

5.1.5 N-grams

Two types of n-grams are used in NER. The less common type is a character n-gram. Character N-grams are used to capture the inner structure of a word. Affix feature described above can be considered a special case of character n-grams, where only n-grams at the beginning (resp. end) of the word are used.

The second type are word n-grams. Word n-grams are used to capture word sequences that are often NEs or that often have NEs in their neighbourhood. Unigrams are identical to word feature described above.

5.1.6 Part of speech and morphology

Morphological tags are a useful feature. Generally, NEs are most often nouns, adjectives and numbers. Other types like prepositions appear less frequently and some like verbs are rare. In inflective languages, morphological tags also give us a possibility to detect consecutive words in the same case which can improve the NE detection.

5.1.7 Patterns

Various types of patterns have been used in NER. The rule-based systems are based on patterns, but machine learning methods can also exploit patterns as features. For machine learning, it is good to use some method for automatic extraction of patterns [48][49]. Usage of more complex hand-made rules in machine learning systems tends to have a negative impact on the adaptability.

An interesting approach was presented in [50]. One of six categories is assigned for each word. Each category is represented by one character. The pattern is then a string of the category characters.

5.2 Global features

5.2.1 Previous appearance

Some authors use a previous appearance of an NE in the document. If NE is already marked in the text, new appearance of the same NE will be probably NE with the same class.

Sometimes this is not used as a feature but as a postprocessing step. After the classification step, all the found NEs are reviewed and the NE classes can be changed if they appeared in other class with higher probability.

5.2.2 Meta information

For some documents meta information is available. This meta information can consist of various things and some of them can be used as a features for NER. Good example of documents that can include meta information are news articles or emails. When dealing with news articles (resp. emails) it can be useful to use category of that article, its title (resp. subject) etc. However, usage of meta information has negative impact on adaptability, because the meta information will not be available or will be different for other domain or data source.

5.3 List-lookup features

5.3.1 Gazetteers

Gazetteers are lists of NEs. The opinions on gazetteers are mixed. Some authors stated that usage of gazetteers has not improved their results while other says it improved it significantly. The fact that gazetteers are used in many systems speaks for the second claim.

Systems with gazetteers are obviously loosing the possibility of direct use for another languages. There were attempts to mitigate this problem by unsupervised or semi-supervised gazetteer creation.

5.3.2 Trigger words

Trigger words are words which are not NEs, but are often in the neighbourhood of NEs. For example 'president' can be a trigger word for Person NE. List of trigger words can be automatically learned from corpora or can be made by hand (e.g. list of degrees).

5.4 External Features

5.4.1 Wikipedia

Wikipedia is a rich source on information, it is thus natural that some authors try to exploit it in NER. In [51][52] the authors use the categories of some word sequences as a feature for NER. The results are obviously dependent on the language of Wikipedia, because English has many times more articles than other languages.

Wikipedia have been used for another purposes then as a feature in NER. It was used as a automatically created corpus for NER [53] or for automatic creation of gazetteers [54]. Wikipedia is also often used as a source of information for Named Entity Disambiguation task, which is related to NER.

6 Future work

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."
Winston Churchill

Even though much effort was devoted to NER there are still challenging problems in this task. In this section we will try to enumerate them.

The first challenge is lower performance of Czech NER compared to English. The previous experiments show that Czech NER has significantly lower performance using the same methods and features like English. It is thus important to search new ways to improve it. We believe that solving the problem for Czech can also improve the results for other languages with similar problems.

Another challenge is multilingualism of NER. It is important to develop methods and features which works for all or at least many languages and do not decrease the performance of language dependent methods. All the main NER conferences addressed multilingualism. The proposed methods still have lower performance for some languages like Czech or German, this problem is thus connected with the one above.

It is also important to find some way to effectively adapt systems trained on one domain to other domains. This challenge can be extended if the new domain uses different set of classes than the original. Even though this is an important task, not many authors have addressed it.

Another area which needs further research are unsupervised and semi-supervised methods for NER.

One task is deeply connected to NER and it is Named Entity Disambiguation. This task follows the NER task and tries to connect all NEs which refer

to one object and to assign some standard form to it. It is significant help for other tasks which uses NER as preprocessing tool.

We also feel the importance of quality and reusable implementation of NER system, so it can be directly and effectively used for other projects and for commercial use. We will try to find or define a standard way or interface for NER.

6.1 Aims of Doctoral Thesis

- Develop new recognition methods and features to improve performance for Czech and other languages.
- Propose semi-supervised approaches to improve the adaptability of NER.
- Experiment with disambiguation on small subset of selected named entities.
- Create quality and reusable NER system, which will provide standard interfaces.

Bibliography

[1] Ralph Grishman and Beth Sundheim. Message understanding conference-6: a brief history. In *Proceedings of the 16th conference on Computational linguistics - Volume 1*, pages 466–471, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[2] Erik F. Tjong Kim Sang. Introduction to the conll-2002 shared task: language-independent named entity recognition. In *proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02, pages 1–4, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[3] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[4] G. Doddington, A. Mitchell, M. Przybocki, L. Ramshaw, S. Strassel, and R. Weischedel. The Automatic Content Extraction (ACE) Program–Tasks, Data, and Evaluation. *Proceedings of LREC 2004*, pages 837–840, 2004.

[5] L. Ferro, L. Gerber, I. Mani, B. Sundheim, and G. Wilson. TIDES 2005 Standard for the Annotation of Temporal Expressions. Technical report, MITRE, September 2005.

[6] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition with a maximum entropy approach. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 160–163. Edmonton, Canada, 2003.

[7] Yassine Benajiba, Mona Diab, and Paolo Rosso. Arabic named entity recognition using optimized feature sets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 284–293, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[8] Georgi Georgiev, Preslav Nakov, Kuzman Ganchev, Petya Osenova, and Kiril Simov. Feature-rich named entity recognition for bulgarian using conditional random fields. In *Proceedings of the International Conference RANLP-2009*, pages 113–117, Borovets, Bulgaria, September 2009. Association for Computational Linguistics.

[9] *On Using Ensemble Methods for Chinese Named Entity Recognition*, Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing, 2006.

[10] Jana Kravalová and Zdeněk Žabokrtský. Czech named entity corpus and svm-based recognizer. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, NEWS '09, pages 194–201, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[11] Xavier Carreras, Lluís Màrques, and Lluís Padró. Named entity extraction using adaboost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei, Taiwan, 2002.

[12] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada, 2003.

[13] Georgios Lucarelli, Xenofon Vasilakos, and Ion Androutsopoulos. Named entity recognition in greek texts with an ensemble of svms and active learning. *International Journal on Artificial Intelligence Tools*, 16(6):1015–1045, 2007.

[14] Richárd Farkas, György Szarvas, and András Kocsor. Named entity recognition for hungarian using various machine learning algorithms. *Acta Cybern.*, 17(3):633–646, January 2006.

[15] Magda Ševčíková, Zdeněk Žabokrtský, and Oldřich Krůza. Named entities in czech: annotating data and developing ne tagger. In *Proceedings of the 10th international conference on Text, speech and dialogue*, TSD'07, pages 188–195, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] Massimiliano Ciaramita and Yasemin Altun. Named-entity recognition in novel domains with external lexical knowledge. *Proceedings of the NIPS Workshop on Advances in Structured Learning for Text and Speech Processing*, (Section 00):0–3, 2005.

[17] Thierry Poibeau and Leila Kosseim. Proper Name Extraction from Non-Journalistic Texts. *Language and Computers*, pages 144–157, December 2001.

[18] Jing Jiang and ChengXiang Zhai. Exploiting domain structure for named entity recognition. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 74–81, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[19] Ada Brunstein. Annotation guidelines for answer types, 2002.

[20] S. Sekine, K. Sudo, and C. Nobata. Extended named entity hierarchy. In M. Gonzáles Rodríguez and C. Paz Suárez Araujo, editors, *Proceedings of 3^{rd} International Conference on Language Resources and Evaluation (LREC'02)*, pages 1818–1824, Canary Islands, Spain, May 2002.

[21] Dan Shen, Jie Zhang, Guodong Zhou, Jian Su, and Chew lim Tan. Effective adaptation of a hidden markov model-based named entity recognizer for biomedical domain. In *In: Proceedings of NLP in Biomedicine, ACL*, pages 49–56, 2003.

[22] David Nadeau, Peter D. Turney, and Stan Matwin. Unsupervised named-entity recognition: generating gazetteers and resolving ambiguity. In *Proceedings of the 19th international conference on Advances in Artificial Intelligence: Canadian Society for Computational Studies of Intelligence*, AI'06, pages 266–277, Berlin, Heidelberg, 2006. Springer-Verlag.

[23] K. Humphreys, R. Gaizauskas, S. Azzam, C. Huyck, B. Mitchell, H. Cunningham, and Y. Wilks. University of sheffield: Description of the lasie-ii system as used for muc-7. In *In Proceedings of the Seventh Message Understanding Conferences (MUC-7*. Morgan, 1998.

[24] Andrei Mikheev, Claire Grover, and Marc Moens. Description of the ltg system used for muc-7. In *In Proceedings of 7th Message Understanding Conference (MUC-7*, 1998.

[25] Lance Ramshaw and Mitch Marcus. Text Chunking Using Transformation-Based Learning. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Somerset, New Jersey, 1995. Association for Computational Linguistics.

[26] Silviu Cucerzan and David Yarowsky. Language independent ner using a unified model of internal and contextual evidence. In *Proceedings of CoNLL-2002*, pages 171–174. Taipei, Taiwan, 2002.

[27] GuoDong Zhou and Jian Su. Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 473–480, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[28] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[29] Asif Ekbal and Sriparna Saha. Multiobjective optimization for classifier ensemble and feature selection: an application to named entity recognition. *International Journal on Document Analysis and Recognition*, pages 1–24. 10.1007/s10032-011-0155-7.

[30] Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7:42–48, 1985. 10.1007/BF03023004.

[31] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22:39–71, March 1996.

[32] J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):pp. 1470–1480, 1972.

[33] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45:503–528, December 1989.

[34] Jorge Nocedal. Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[35] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[36] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition with a maximum entropy approach. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 160–163, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[37] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[38] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[39] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*, pages 188–191, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[40] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134, June 2005.

[41] Jun ichi Kazama and Kentaro Torisawa. Inducing gazetteers for named entity recognition by large-scale clustering of dependency relations. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, pages 407–415. The Association for Computer Linguistics, 2008.

[42] Marius Pasca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts - step one: the one-million fact extraction challenge. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, AAAI'06, pages 1400–1405. AAAI Press, 2006.

[43] Radu Florian. Named entity recognition as a house of cards: classifier stacking. In *proceedings of the 6th conference on Natural language learn-*

ing - Volume 20, COLING-02, pages 1–4, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[44] Koji Tsukamoto, Yutaka Mitsuishi, and Manabu Sassano. Learning with multiple stacking for named entity recognition. In *Proceedings of CoNLL-2002*, pages 191–194, 2002.

[45] Z. Kozareva, O. Ferrández, A. Montoyo, R. Muñoz, A. Suárez, and J. Gómez. Combining data-driven systems for improving named entity recognition. *Data & Knowledge Engineering*, 61(3):449 – 466, 2007. <ce:title>Advances on Natural Language Processing</ce:title> <ce:subtitle>NLDB 05</ce:subtitle>.

[46] Asif Ekbal and Sriparna Saha. Classifier ensemble selection using genetic algorithm for named entity recognition. *Research on Language & Computation*, 8:73–99, 2010. 10.1007/s11168-010-9071-0.

[47] Bart Desmet and Veronique Hoste. Dutch named entity recognition using ensemble classifiers. In Eline Westerhout, Thomas Markus, and Paola Monachesi, editors, *Computational Linguistics in the Netherlands 2010 : selected papers from the twentieth CLIN meeting (CLIN 2010)*, pages 29–41. Landelijke Onderzoeksschool Taalwetenschap (LOT), 2010.

[48] Sujan Kumar Saha, Sudeshna Sarkar, and Pabitra Mitra. A hybrid feature set based maximum entropy hindi named entity recognition. In *Proceedings of the Third International Joint Conference on Natural Language Processing*, 2008.

[49] Partha Pratim Talukdar, Thorsten Brants, Mark Liberman, and Fernando Pereira. A context pattern induction method for named entity extraction. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 141–148, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[50] Xavier Carreras, Lluís Màrquez, and Lluís Padró. A simple named entity extractor using adaboost. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 152–155. Edmonton, Canada, 2003.

[51] Jun'ichi Kazama and Kentaro Torisawa. Exploiting Wikipedia as External Knowledge for Named Entity Recognition. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 698–707, 2007.

[52] Alexander E Richman, Patrick Schone, and Fort George G Meade. Mining wiki resources for multilingual named entity recognition. *Computational Linguistics*, (June):1–9, 2008.

[53] Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. Learning multilingual named entity recognition from wikipedia. *Artificial Intelligence*, (0):–, 2012.

[54] A. Toral and R. Munoz. A proposal to automatically build and maintain gazetteers for Named Entity Recognition by using Wikipedia. *EACL 2006*, 2006.

Neural Machine Translation: A Review of Methods, Resources, and Tools

Zhixing Tan[a,c,d], Shuo Wang[a,c,d], Zonghan Yang[a,c,d], Gang Chen[a,c,d], Xuancheng Huang[a,c,d], Maosong Sun[a,c,d,e], Yang Liu[a,b,c,d,e,*]

[a]*Department of Computer Science and Technology, Tsinghua University*
[b]*Institute for AI Industry Research, Tsinghua University*
[c]*Institute for Artificial Intelligence, Tsinghua University*
[d]*Beijing National Research Center for Information Science and Technology*
[e]*Beijing Academy of Artificial Intelligence*

Abstract

Machine translation (MT) is an important sub-field of natural language processing that aims to translate natural languages using computers. In recent years, end-to-end neural machine translation (NMT) has achieved great success and has become the new mainstream method in practical MT systems. In this article, we first provide a broad review of the methods for NMT and focus on methods relating to architectures, decoding, and data augmentation. Then we summarize the resources and tools that are useful for researchers. Finally, we conclude with a discussion of possible future research directions.

Keywords: Neural machine translation, Attention mechanism, Deep learning, Natural language processing

1. Introduction

Machine Translation (MT) is an important task that aims to translate natural language sentences using computers. The early approach to machine translation relies heavily on hand-crafted translation rules and linguistic knowledge. As natural languages are inherently complex, it is difficult to cover all language irregularities with manual translation rules. With the availability of large-scale parallel corpora, data-driven approaches that learn linguistic information from data have gained increasing attention. Unlike rule-based machine translation, Statistical Machine Translation (SMT) [1, 2] learns latent structures such as word alignments or phrases directly from parallel corpora. Incapable of modeling long-distance dependencies between words, the translation quality of SMT is far from satisfactory. With the breakthrough of deep learning, Neural Machine Translation (NMT) [3, 4, 5, 6] has emerged as a new paradigm and quickly replaced SMT as the mainstream approach to MT.

Neural machine translation is a radical departure from previous machine translation approaches. On the one hand, NMT employs continuous representations instead of discrete symbolic representations in SMT. On the other hand, NMT uses a single large neural network to model the entire translation process, freeing the need for excessive feature engineering. The training of NMT is end-to-end as opposed to separately tuned components in SMT. Besides its simplicity, NMT has achieved state-of-the-art performance on various language pairs [7]. In practice, NMT also becomes the key technology behind many commercial MT systems [8, 9].

As neural machine translation attracts much research interest and grows into an area with many research directions, we believe it is necessary to conduct a comprehensive review of NMT. In this work, we will give an overview of the key ideas and innovations behind NMT. We also summarize the resources and tools that are useful and easily accessible. We hope that by tracing the origins and evolution of NMT, we can stand on the shoulder of past studies, and gain insights into the future of NMT.

The remainder of this article is organized as follows: Section 2 will review the methods of NMT. We first introduce the basics of NMT, and then we selectively describe the recent progress of NMT. We focus on methods related to architectures, decoding, and data augmentation. Section 3 will summarize the resources such as parallel or monolingual corpora that are publicly available to researchers. Section 4 will describe tools that are useful for training and evaluating NMT models. Finally, we conclude and discuss future directions in Section 5.

2. Methods

As a data-driven approach to machine translation, NMT also embraces the probabilistic framework. Mathematically speaking, the goal of NMT is to estimate an unknown conditional distribution $P(\boldsymbol{y}|\boldsymbol{x})$ given the dataset $\mathcal{D}$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are random variables representing source input and target output, respectively. We strive to answer the three basic questions of NMT:

- *Modeling.* How to design neural networks to model the conditional distribution?
- *Inference.* Given a source input, how to generate a translation sentence from the NMT model?

*Corresponding author.
Email address: liuyang2011@tsinghua.edu.cn (Yang Liu)

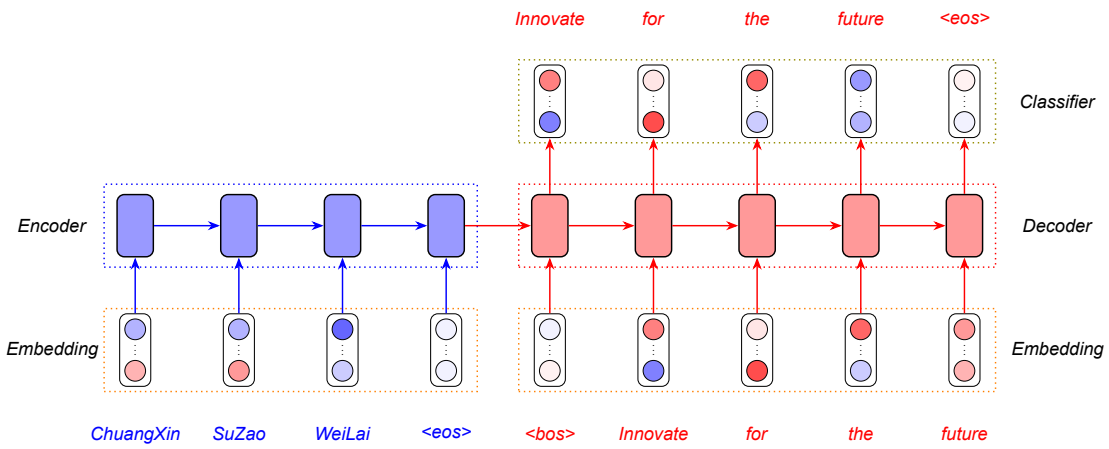


Figure 1: An overview of the NMT architecture, which consists of embedding layers, a classification layer, an encoder network, and a decoder network. We use different colors to distinguish different languages.

- *Learning.* How to effectively learn the parameters of NMT from data?

In 2.1, we first describe the basic methods of NMT for addressing the above three questions. We then dive into the details of NMT architectures in 2.2 and introduce bidirectional inference and non-autoregressive NMTs in 2.3. We discuss alternative training objectives and using monolingual data in 2.4 and 2.5, respectively.

Despite the great success, NMT is far from perfect. There are several theoretical and practical challenges faced by NMT. We survey the research progress of some important directions. We describe methods for open vocabulary in 2.6, prior knowledge integration in 2.7, and interpretability and robustness in 2.8.

2.1. Overview of NMT

2.1.1. Modeling

Translation can be modeled at different levels, such as document-, paragraph-, and sentence-level. In this article, we focus on sentence-level translation. Besides, we also assume the input and output sentences are sequences. Thus the NMT model can be viewed as a *sequence-to-sequence* model. Assuming we are given a source sentence $\mathbf{x} = \{x_1, \ldots, x_S\}$ and a target sentence $\mathbf{y} = \{y_1, \ldots, y_T\}$. By using the chain rule, the conditional distribution can be factorized from left-to-right (L2R) as

$$P(\boldsymbol{y} = \mathbf{y}|\boldsymbol{x} = \mathbf{x}) = \prod_{t=1}^{T} P(y_t|y_0, \ldots, y_{t-1}, x_1, \ldots, x_S). \quad (1)$$

NMT models which conform the Eq. (1) is referred to as *L2R autoregressive* NMT [3, 4, 5, 6], for the prediction at time-step t is taken as a input at time-step $t + 1$.

Almost all neural machine translation models employ the *encoder-decoder framework* [4]. The encoder-decoder framework consists of four basic components: the embedding layers, the encoder and decoder networks, and the classification layer. Figure 1 shows a typical autoregressive NMT model using the encoder-decoder framework, which we shall use as an example. "<bos>" and "<eos>" are special symbols that mark the beginning and ending of a sentence, respectively.

The embedding layer embodies the concept of *continuous representation*. It maps a discrete symbols x_t into a continuous vector $\mathbf{x}_t \in \mathbb{R}^d$, where d denotes the dimension of the vector. The embeddings are then fed into later layers for more finer-grained feature extraction.

The encoder network maps the source embeddings into hidden continuous representations. To learn expressive representations, the encoder must be able to model the ordering and complex dependencies that existed in the source language. Recurrent neural networks (RNN) are suitable choice for modeling variable-length sequences. With RNNs, the computation involves in encoder can be described as

$$\mathbf{h}_t = \mathrm{RNN}_{\mathrm{ENC}}(\mathbf{x}_t, \mathbf{h}_{t-1}). \quad (2)$$

By iteratively applying the state transition function $\mathrm{RNN}_{\mathrm{ENC}}$ over the input sequence, we can use the final state $\mathbf{h}_S$ as the representation for the entire source sentence, and then feed it to the decoder.

The decoder can be viewed as a language model conditioned on $\mathbf{h}_S$. The decoder network extracts necessary information from the encoder output, and also models the long-distance dependencies between target words. Given the start symbol $y_0 =$ <bos> and the initial state $\mathbf{s}_0 = \mathbf{h}_S$, the RNN decoder compresses the decoding history $\{y_0, \ldots, y_{t-1}\}$ into a state vector $\mathbf{s}_t \in \mathbb{R}^d$:

$$\mathbf{s}_t = \mathrm{RNN}_{\mathrm{DEC}}(\mathbf{y}_{t-1}, \mathbf{s}_{t-1}). \quad (3)$$

The classification layer predicts the distribution of target tokens. The classification layer is typically a linear layer with *softmax* activation function. Assuming the vocabulary of target language is $\mathcal{V}$, and $|V|$ is the size of the vocabulary. Given an decoder output $\mathbf{s}_t \in \mathbb{R}^d$, the classificaition layer first maps $\mathbf{h}$ to a vector $\mathbf{z}$ in the vocabulary space $\mathbb{R}^{|V|}$ with the linear map. Then the softmax function is used to ensure the output vector is

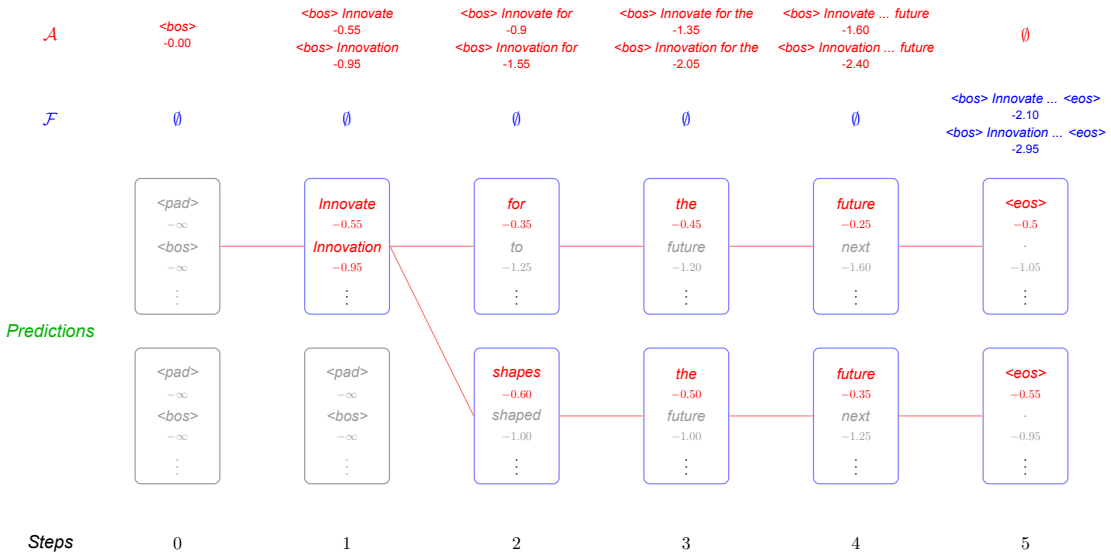


Figure 2: A running example of the beam-search algorithm.

a valid probability:

$$\text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{i=1}^{|V|} \exp(\mathbf{z}_{[i]})}, \tag{4}$$

where we use $\mathbf{z}_{[i]}$ to denote the i-th component in $\mathbf{z}$.

2.1.2. *Inference*

Given an NMT model and a source sentence $\mathbf{x}$, how to generate a translation from the model is an important problem. Ideally, we would like to find the target sentence $\mathbf{y}$ which maximizes the model prediction $P(\mathbf{y}|\boldsymbol{x} = \mathbf{x}; \boldsymbol{\theta})$ as the translation. However, due to the intractably large search space, it is impractical to find the translation with the highest probability. Therefore, NMT typically uses local search algorithms such as *greedy search* or *beam search* to find a local best translation.

Beam search is a classic local search algorithm which have been widely used in NMT. Previously, beam search have been successfully applied in SMT. The beam search algorithm keeps track of k states during the inference stage. Each state is a tuple $\langle y_0 \dots y_t, v \rangle$, where $y_0 \dots y_t$ is a candidate translation, and v is the log-probability of the candidate. At each step, all the successors of all k states are generated, but only the top-k successors are selected. The algorithm usually terminates when the step exceed a pre-defined value or k full translation are found. It should be noted that the beam search will degrade into the greedy search if $k = 1$.

The pseudo-codes of the beam search algorithm are given in 1. We also give a running example of the algorithm in Figure 2.

Algorithm 1: The beam search algorithm

$t \leftarrow 1$;
$\mathcal{A} = \{\langle \texttt{<bos>}, 0 \rangle\}$; ▷ *The set of alive candidates*
$\mathcal{F} = \{\}$; ▷ *The set of finished candidates*
while $t <$ `max_length` **do**
  $\mathcal{C} = \{\}$;
  for $\langle y_0 \dots y_{t-1}, v \rangle \in \mathcal{A}$ **do**
    $\boldsymbol{p} \leftarrow \text{NMT}(y_0 \dots y_{t-1}, \mathbf{x})$;
    for $w \in \mathcal{V}$ **do**
      $y_t \leftarrow w$;
      $l \leftarrow \log(\boldsymbol{p}[w])$;
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle y_0 \dots y_t, v + l \rangle\}$;
    end
  end
  $\mathcal{C} \leftarrow \text{TopK}(\mathcal{C}, k)$;
  for $\langle y_0 \dots y_t, v \rangle \in \mathcal{C}$ **do**
    if $y_t == \texttt{<eos>}$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{\langle y_0 \dots y_t, v \rangle\}$;
    else
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{\langle y_0 \dots y_t, v \rangle\}$;
    end
  end
  $\mathcal{A} \leftarrow \text{TopK}(\mathcal{A}, k)$;
  $\mathcal{F} \leftarrow \text{TopK}(\mathcal{F}, k)$;
  $t \leftarrow t + 1$;
end
$\langle y_0 \dots y_t, v \rangle \leftarrow \text{Top}(\mathcal{F})$;
return $y_1 \dots y_t$

2.1.3. *Training of NMT Models*

NMT typically uses maximum log-likelihood (MLE) as the training objective function, which is a commonly used method of estimating the parameters of a probability distribution. Formally, given the training set $\mathcal{D} = \{\langle \mathbf{x}^{(s)}, \mathbf{y}^{(s)} \rangle\}_{s=1}^{S}$, the goal of training is to find a set of model parameters that maximize the log-likelihood on the training set:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \Big\{ \mathcal{L}(\boldsymbol{\theta}) \Big\}, \tag{5}$$

where the log-likelihood is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{s=1}^{S} \log P(\mathbf{y}^{(s)}|\mathbf{x}^{(s)}; \boldsymbol{\theta}). \tag{6}$$

By the virtue of *back-propagation* algorithm, we can efficiently compute the gradient of $\mathcal{L}$ with respect to $\boldsymbol{\theta}$. The training of NMT models usually adopts *stochastic gradient search* (SGD) algorithm. Instead of computing gradients on the full training set, SGD computes the loss function and graidents on a *minibatch* of the training set. The plain SGD optimzier updates the parameters of an NMT model with the following rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{7}$$

where α is the *learning rate*. With well-chosen learning rate, the parameters of NMT are guaranteed to converge into a local optima. In practice, instead of plain SGD optimizer, adaptive learning rate optimizers such as Adam [10] are found to greatly reduce the training time.

2.2. Architectures

2.2.1. Evolution of NMT Architectures

Since 2013, there are attempts to build a pure neural MT. Early NMT architectures such as RCTM [3], RNNEncdec [4], and Seq2Seq [5] adopt a *fixed-length* approach, where the size of source representation is fixed regardless the length of source sentences. These works typically use recurrent neural networks (RNN) as the decoder network for generating variable-length translation. However, it is found that the performance of this approach degrades as the length of the input sentence increases [11]. Two explanations can account for this phenomenon:

1. The fixed-length representations have become the bottleneck during the encoding process for long sentences [4]. As the encoder is forced to compress the entire source sentence into a set of fixed-length vectors, some important information may be lost in this process.
2. The longest path between the source words and target words is $O(S + T)$, and it is challenging for neural networks to learn long-term dependencies [12]. Sutskever et al. [5] found that reverse the source sentence can significantly improve the performance of the fixed-length approach. By reversing the source sentence, the paths between the beginning words of source and target sentences are reduced, thus the optimization problem becomes easier.

Due to these limitations, later NMT architectures switch to *variable-length* source representations, where the length of source representations depends on the length of the source sentence. The RNNsearch architecture [6] introduces *attention mechanism*, which is an important approach to implementing variable-length representations. Figure 3 shows the comparison between fixed-length and variable-length approaches. By using the attention mechanism, the paths between any source and target words are within a constant length. As a result, the attention mechanism has eased optimization difficulty.

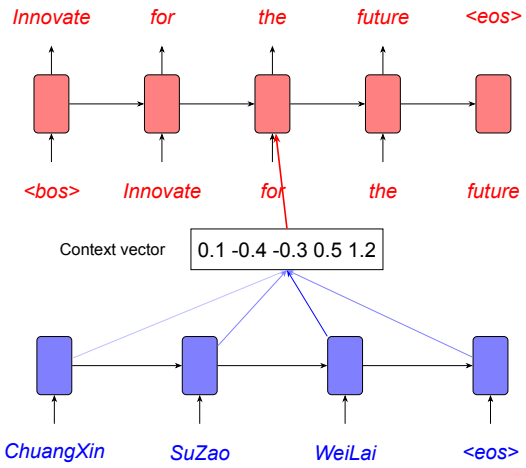


Figure 3: At each decoding step, the attention mechanism dynamically generates a context vector based on the most relevant source representations for predicting the next target word.

With the breakthrough of deep learning, NMT with deep neural networks have attracted much research interest. Seq2Seq [5] is the first architecture demonstrate the potential of deep NMT. Later architectures such as GNMT [8], ByteNet [13], ConvSeq-2Seq [14], and Transformer [15] all use multi-layered neural networks. ByteNet and ConvSeq2Seq have replaced RNNs with convolutional neural networks (CNN) in their architectures while Transformer relies entirely on self-attention networks (SAN). Both CNNs and SANs can reduce the sequential operations invovled in RNNs, and benefit from the parallel computation provided by modern devices such as GPU or TPU. Importantly, SAN can further reduce the longest path between two target tokens. We shall later describe the techniques for building these architectures.

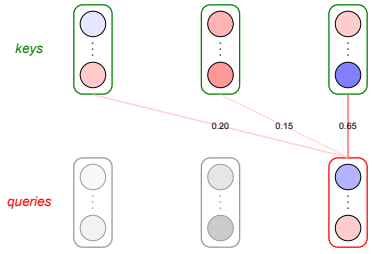
2.2.2. Attention Mechanism

The introduction of attention mechanism [6] is a milestone in NMT architecture research. The attention network computes the relevance of each value vector based on queries and keys. This can also be interpreted as a content-based addressing scheme [16]. Formally, given a set of m query vectors $\mathbf{Q} \in \mathbb{R}^{m \times d}$, a set of n key vectors $\mathbf{K} \in \mathbb{R}^{n \times d}$ and associated value vectors $\mathbf{V} \in \mathbb{R}^{n \times d}$, the computation of attention network involves two steps. The first step is to compute the relevance between keys and values, which is formally described as

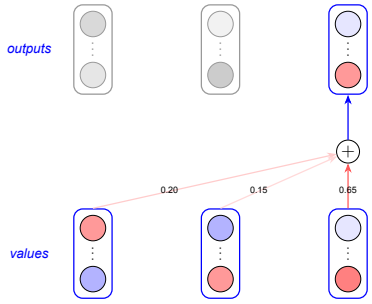
$$\mathbf{R} = \text{score}(\mathbf{Q}, \mathbf{K}), \tag{8}$$

where $\text{score}(\cdot)$ is a scoring function which have several alternatives. $\mathbf{R} \in \mathbb{R}^{m \times n}$ is a matrix storing the relevance score between each keys and values. The next step is compute the output vectors. For each query vector, the corresponding output vector is expressed as a weighted sum of value vectors:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{R}) \cdot \mathbf{V}. \tag{9}$$



(a) Given a query vector and key vectors, the attention network first computes a weight vector through the scoring function.



(b) Each output vector is computed as a weighted sum of value vectors.

Figure 4: Detailed computations involved in the attention mechanism.

Figure 4 depicts the two steps involved in the computation of attention mechanism.

Considering on the scoring function, the attention networks can be roughly classified into two categories: additive attention [6] and dot-product attention [17]. The additive attention models score through a feed-forward neural network:

$$\mathbf{R}_{[i,j]} = \mathbf{v}^\top \tanh(\mathbf{W}_s\mathbf{Q}_{[i]} + \mathbf{U}_s\mathbf{K}_{[j]}), \tag{10}$$

where $\mathbf{W}_s \in \mathbb{R}^{d\times d}$, $\mathbf{U}_s \in \mathbb{R}^{d\times d}$, and $\mathbf{v} \in \mathbb{R}^{d\times 1}$ are learnable parameters. On the other hand, the dot-product attention uses dot production to compute the matching score:

$$\mathbf{R}_{[i,j]} = \mathbf{Q}_{[i]}^\top \mathbf{K}_{[j]}. \tag{11}$$

In practice, the dot-product attention is much faster than the additive attention. However, the dot-product attention is found to be less stable than the additive attention when d is large [15]. Vaswani et al. [15] suspect that the dot-products grow large in magnitude for large values of d, which may resulting extremely small gradients caused by the softmax function. To remedy this issue, they propose to scale the dot-products by $\frac{1}{\sqrt{d}}$.

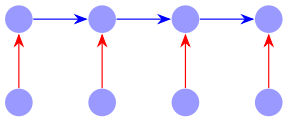
The attention mechanism is usually used as a part of the decoder network. Another type of attention network called self-attention network, is widely used in both the encoder and decoder of NMT. We shall describe self-attention and other variants of attention network later.

2.2.3. *RNNs, CNNs, and SANs*

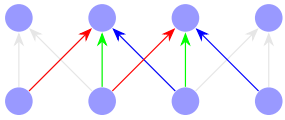
There are many methods of building powerful encoders and decoders, which can roughly divide into three categories: the recurrent neural network (RNN) based methods, convolutional neural network (CNN) based methods, and self-attention network (SAN) based methods. There are several aspects we need to take into considerations for building an encoder and decoder:

1. *Receptive field*. We hope each output produced by the encoder and decoder can potentially encode arbitrary information in the input sequence.
2. *Computational complexity*. It is desirable to a use network with lower computational complexity.
3. *Sequential operations*. Too many sequential operations preclude the parallel computation within the sequence.
4. *Position awareness*. The network should distinguish the ordering presents in the sequence.

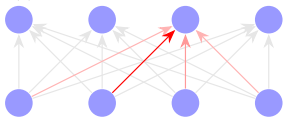
Table 1 summarizes the computation as well as the above-mentioned aspects of typical RNN, CNN, and SAN.



(a) RNN.

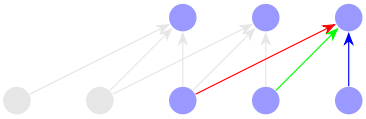


(b) CNN with kernel width $k = 3$.

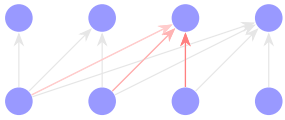


(c) SAN.

Figure 5: Overview of the computation diagram of RNN, CNN, and SAN. To be clarity, we use a node to denote the input or output vector of a specific layer.



(a) CNN with additional zero paddings.



(b) SAN with causal masking.

Figure 6: The computation of CNN and SAN during decoding.

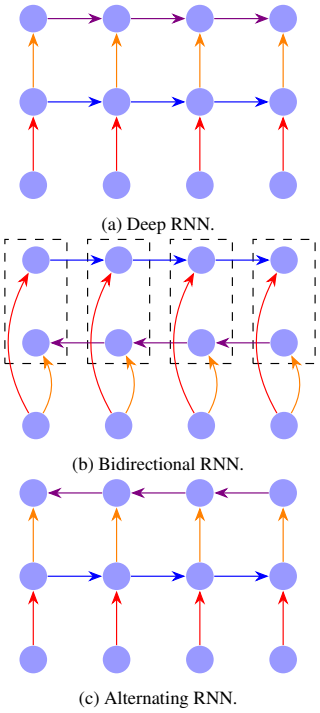
Figure 5 gives an overview of the ways of RNN, CNN, and SAN to encode sequences, respectively. In order to keep the auto-regressive property of NMT decoder during training, CNN and SAN furthur needs additional padding and masking to prevent the network from seeing future words. Figure 6 shows padding and masking used in CNN and SAN.

As we can see in Figure 5(a), RNNs are a family of sequential models that repeatedly apply the same state transition function to sequences. In theory, RNNs are among the most

Layer	Computation	R.F.	Complexity	S.O.	P.A.
RNN	$\mathbf{h}_{l,t} = \mathbf{W}\mathbf{h}_{l-1,t} + \mathbf{U}\mathbf{h}_{l,t-1}$	∞	$O(n \cdot d^2)$	$O(n)$	Yes
CNN	$\mathbf{h}_{l,t} = \sum_{i=1}^{k} \mathbf{W}^{(i)}\mathbf{h}_{l-1,t+i-\lceil \frac{k+1}{2} \rceil}$	k	$O(k \cdot n \cdot d^2)$	$O(1)$	Yes
SAN	$\mathbf{h}_{l,t} = \sum_{i=1}^{n} \alpha_{l,i}\mathbf{h}_{l-1,i}$	∞	$O(n^2 \cdot d)$	$O(1)$	No

Table 1: Comparisons between different neural network layers. We use R.F. to denote the receptive field, S.O. to denote the number of sequential operations, and P.A. to denote the position awareness of the layer. t is the position in the sequence, l is the layer number. For CNN, k is the filter width and $\mathbf{W}^{(i)}$ is the weight of the i-th filter.

powerful family of neural networks [18]. However, it suffers from severe vanishing and exploding gradient problem [12] in practice. RNNs with gates, such as long short-term memory (LSTM) [19] and gated recurrent unit (GRU) [4] have been proposed to alleviate this problem. Another way to stabilize the training is to incorporate normalization layers, such as layer normalization [20].



(a) Deep RNN.

(b) Bidirectional RNN.

(c) Alternating RNN.

Figure 7: Three extensions to RNNs.

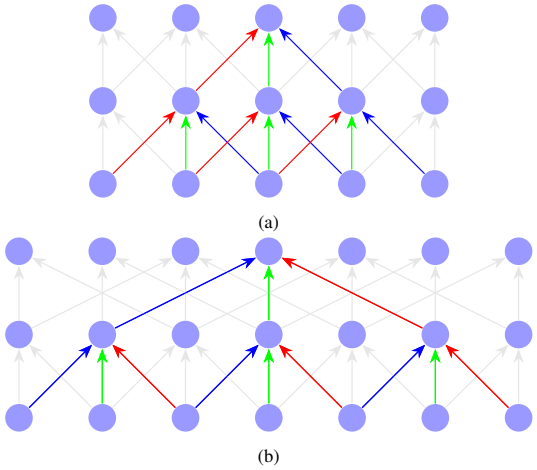
Figure 7 shows three extensions to RNNs that are widely used in NMT literature. Deep RNNs is one important way to increase the expressive power of RNNs. However, training deep neural networks is challenging because it also faces the vanishing and exploding gradient problem. There are many ways to construct deep RNNs, and the most popular one is by stacking multiple RNNs with *residual connections* [21]. The residual connection is an important method to construct deep neural networks. Residual connections use the identity mappings as the skip connections, which is formally described as

$$\mathbf{y} = \mathbf{x} + f(\mathbf{x}), \tag{12}$$

where $\mathbf{x}$, and $\mathbf{y}$ are input and output, respectively. f is the neural network. By using identity mappings, the gradient signal can directly propagate into lower layers. Bidirectional RNNs [6] use two RNNs to process the same sequence in opposite directions, and concatenating the results of both RNNs to be the final output. In this way, each output of bidirectional RNNs encodes all the tokens in the sequence. An alternative to bidirectional RNNs is alternating RNNs [22], which consists of RNNs in opposite directions in adjacent layers.

Besides the difficulty training of RNNs, another major drawback of RNNs is that RNNs are sequential models in nature, which cannot benefit from the parallel computations provided by modern GPUs. CNNs and SANs, however, which fully exploit the parallel computation within sequences, are widely used in newer NMT architectures.

Convolutional neural network (CNN) was first introduced into NMT in 2013 [3]. However, it was not as successful as RNNs until 2017 [14]. The main obstacle for applying CNNs is its limited receptive field. Stacking L CNNs with kernel width k can increase the receptive field from k to $L \cdot (k-1)+1$. The network needs to go deeper with large L and adopt large kernel size k to model long sentences. However, learning deep CNNs is challenging, and using large kernel size k may significantly increase the complexity and parameters involved in CNNs.



(a)

(b)

Figure 8: Comparison between CNN and dilated CNN. (a) Two layers CNN with filter width $k = 3$ for each layer. (b) Dilated CNN with filter width $k = 3$ for all layers, dilation rate $r = 1$ in layer 1 and $r = 2$ in layer 2.

One solution to increase the receptive field without using a large k is through dilation [13]. Figure 8 shows the comparison between plain CNN and dilated CNN. Plain CNN can be viewed as a special case of dilated CNN with a dilation rate $r = 1$. The computation of dilated CNN is mathematically for-

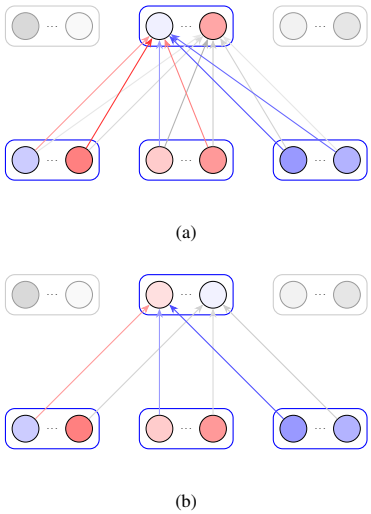


Figure 9: Comparison between CNN and depthwise CNN. Each node in the graph represents a neuron instead of a vector. (a) Plain CNN. We highlight the computation of the first neuron in the output vector. (b) Depthwise CNN. Note that the connections are significantly reduced compared with plain CNN.

mulated as

$$\mathbf{h}_{l,t} = \sum_{i=1}^{k} \mathbf{W}^{(i)} \mathbf{h}_{l-1,t+(i-\lceil \frac{k+1}{2} \rceil)\times r}. \tag{13}$$

Stacking L dilated CNNs whereby the dilation rates are doubled every layer, the receptive field increases to $(2^L-1)\cdot(k-1)+1$. As a result, the receptive field grows exponentially with L, as opposed to linearly with L in plain CNN.

Another solution is to reduce the computations involved in CNN. Depthwise convolution [23] reduces the complexity from $O(kd^2)$ to $O(kd)$ by performing convolution independently over channels. Figure 9 depicts the comparison between CNN and depthwise CNN. The output of the depthwise convolution layer is defined as

$$\mathbf{h}_{l,t} = \sum_{i=1}^{k} \mathbf{w}^{(i)} \odot \mathbf{h}_{l-1,t+i-\lceil \frac{k+1}{2} \rceil}, \tag{14}$$

where $\mathbf{w}^{(i)}$ is the i-th column of weight matrix $\mathbf{W} \in \mathbb{R}^{k\times d}$. Lightweight convolution [24] further reduces the number of parameters of depthwise convolution through weight sharing.

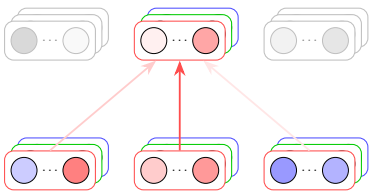


Figure 10: An illustration of multi-head attention.

Self-attention network (SAN) [15] is a special case of attention network where the queries, keys, and values come from the same sequence. Similar to CNN, SAN is trivial to parallelize. Furthermore, Each output in SAN also has infinite receptive fields, which is the same with RNN. In SAN, the queries, keys, and values are typically obtained through a linear map of the input representations. The scaled dot-product self-attention mechanism can be formally described as

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}})\mathbf{V}. \tag{15}$$

Multi-head attention [15] is an extended attention network with multiple parallel heads. Each head attends information from different subspace across value vectors. As a result, multi-head attention can perform more flexible transformations than the single-head attention. We give an illustration of multi-head attention in Figure 10.

The major disadvantage of SAN network is that it ignores the ordering of words in the sequence. To remedy this, SAN needs additional position encoding to differentiate orders. Vaswani et al. [15] proposed a sinusoid style position encoding, which is formulated as

$$\text{timing}(t, 2i) = \sin(t/10000^{2i/d}), \tag{16}$$

$$\text{timing}(t, 2i+1) = \cos(t/10000^{2i/d}), \tag{17}$$

where t is the position and i is the dimension index. Another popular way of position encoding is to learn an additional position embedding. Finally, the position encoding is added to each word representation, so the same words with different positions can have different representations.

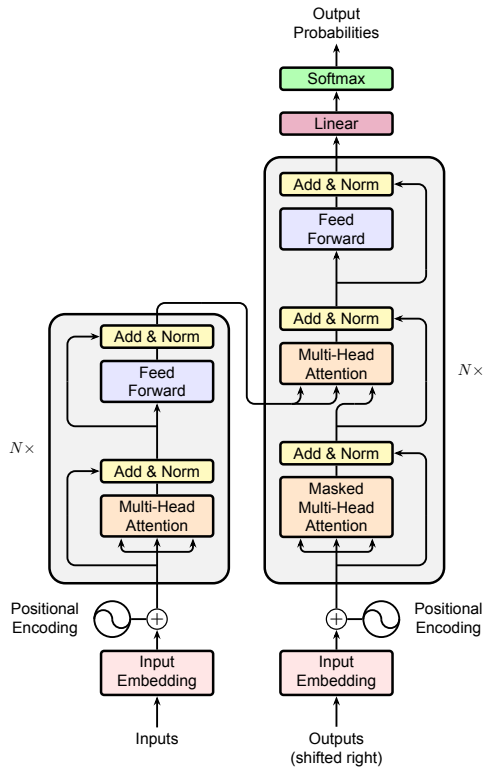
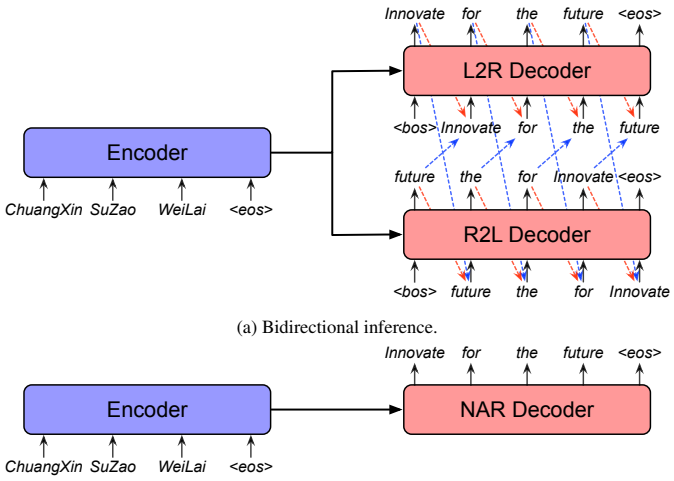


Figure 11: The Transformer architecture.



(a) Bidirectional inference.

(b) Non-autoregressive decoding.

Figure 12: Comparisons of different decoding strategies. (a) Bidirectional decoding: generates a sentence in both left-to-right (L2R) and right-to-left (R2L) directions; (b) Non-autoregressive (NAR) decoding: generates a sentence at one time.

2.2.4. Comparison of Fundamental Architectures

We take the state-of-the-art Transformer architecture [15] as an example to put all things together. Figure 11 shows the architecture of Transformer. The Transformer model relies solely on attention networks, with additional sinusoid-style position encoding added to input embedding. The Transformer network consists of a stack of 6 encoder layers and 6 decoder layers. Each encoder layer contains two sub-layers whereas each decoder layer contains three sub-layers. To stabilize optimization, Transformer uses residual connection and layer normalization in each sub-layer.

We summarize the comparison of fundamental NMT architectures in Table 2. We highlight several important aspects of these fundamental architectures.

2.3. Bidirectional Inference and Non-autoregressive NMT

The dominate approach to NMT factorizes the conditional probability $P(\mathbf{y}|\mathbf{x})$ from left to right (L2R) auto-repressively. However, the factorization of the distribution is not unique. Researchers [25, 26, 27, 28] have found that models with right-to-left (R2L) factorization are complementary to L2R models. The bidirectional inference is an approach to simultaneously generating translation with both L2R and R2L decoders. In addition to auto-regressive approaches where each output word on previously generated outputs, non-autoregressive NMTs [29] avoids this auto-regressive property and produces outputs in parallel, allowing much lower latency during inference.

2.3.1. Bidirectional Inference

Ignoring the future context is another obvious weakness of AR decoding. Thus, a natural idea is that the quality of translation will be improved if autoregressive models can "know" the future information. From this perspective, many approaches have been proposed to improve translation performance by exploring the future context. Some researchers proposed to model both past and future context [30, 31, 32] and some others also found that L2R and R2L autoregressive models can generate complementary translations [25, 26, 27, 28]. For instance, Zhou et al. [28] analyzed the translation accuracy of the first and last 4 tokens for L2R and R2L models, respectively. The statistical results show that, in Chinese-English translation, L2R performs better in the first 4 tokens while R2L translates better in the last 4 tokens.

Based on the findings mentioned above, a number of methods have been proposed to combine the advantages of L2R and R2L decoding. These approaches are collectively referred to as bidirectional decoding. Bidirectional decoding based methods can be mainly fall into four categories [33]: (1) agreement between L2R and R2L [25, 34, 35], (2) rescore with bidirectional decoding [25, 36], (3) asynchronous bidirectional decoding [27, 37], and (4) synchronous bidirectional decoding [28, 38, 39].

Mathematically, the L2R translation order is rather arbitrary, and other arrangements such as R2L factorization are equally correct:

$$P(\mathbf{y}|\mathbf{x}) = \underbrace{\prod_{t=1}^{T} P(y_t|\mathbf{y}_{<t}, \mathbf{x})}_{\text{L2R model}} = \underbrace{\prod_{t=1}^{T} P(y_t|\mathbf{y}_{>t}, \mathbf{x})}_{\text{R2L model}}. \quad (18)$$

Based on this theoretical assumption, Liu et al. [25], Yang et al. [34], and Zhang et al. [35] proposed joint training schemes in which each direction is used as a regularizer for the other direction. Empirical results show that these methods can lead to significant improvements compared with standard L2R and R2L models.

Another common scheme to combine L2R and R2L translations is rescoring (also known as reranking). A strong L2R model firstly produces an n-best list of translations, and then an R2L model rescores each translation in the n-best list [25, 36, 40]. As the scores from L2R and R2L directions are based on complementary models, the quality of translation can be improved by rescoring. Recently, Zhang et al. [27] introduced a new strategy to exploit both L2R and R2L models. They named this method asynchronous bidirectional decoding (ASBD), which first produces outputs (hidden states) by an R2L model and then uses these outputs to optimize the L2R model. ASBD can be done in three steps: The first step is to train a R2L model with bilingual corpora. The second step is to obtain outputs for each given source sentence using the trained R2L model. Finally, the output of R2L model is used as the additional context with the training data to train the L2R model. Thanks to incorporating the future information from the R2L model, the performance of L2R model can be substantially improved.

Although ASBD improves the quality of translation, it also incurs other problems. The L2R and R2L models are trained separately so that they have no chance to interact with each other. Besides, the L2R model translates source sentences based on the outputs of an R2L model, this degrades the efficiency of inference. To address these problems, Zhou et al. [28] further proposed a synchronous bidirectional decoding (SBD) method which generates translations using both L2R and R2L

Model	Encoder	Decoder	Complexity	V.R.	Path_E	Path_D
RCTM 1 [3]	CNN	RNN	$O(S^2+T)$	No	S	T
RCTM 2 [3]	CNN	RNN	$O(S^2+T)$	Yes	S	T
RNNENCDEC/SEQ2SEQ [4, 5]	RNN	RNN	$O(S+T)$	No	$S+T$	T
RNNSEARCH [6]	RNN	RNN	$O(ST)$	Yes	1	T
BYTENET [13]	CNN	CNN	$O(S+T)$	Yes	c	c
CONVSEQ2SEQ [14]	CNN	CNN	$O(ST)$	Yes	1	c
TRANSFORMER [15]	SAN	SAN	$O(S^2+ST+T^2)$	Yes	1	1

Table 2: Comparison of fundamental architectures. V.R. denotes whether the architecture employs variable representation. Path_E denotes the longest path between the source and target tokens. Path_D denotes the longest path between two target tokens.

inference synchronously and interactively. Specifically, SBD uses a new synchronous attention model to allow both L2R and R2L models "communicating" with each other. As shown in Figure 12a, the dotted arrows illustrate interactions between L2R and R2L decoding. Zhou et al. [28] also designed a variant of the standard beam search algorithm to hold L2R and R2L decoding concurrently. The idea behind this algorithm is to maintain that each half beam contains L2R and R2L predictions, respectively. Empirical results show that SBD can significantly improve performance with a slight cost to decoding speed.

Mehri and Sigal [41] proposed a novel middle-out decoder architecture that begins from an initial middle-word and simultaneously expands the sequence in both L2R and R2L directions. Zhou et al. [38] also proposed a similar method that allows L2R and R2L inferences to start concurrently from the left and right sides, respectively. Both L2R and R2L inferences terminate at the middle position. Extensive experiments demonstrate that this method can improve not only the accuracy of translation but also decoding efficiency.

2.3.2. Non-autoregressive NMTs

To reduce the latency during inference, Gu et al. [29] first proposed the non-autoregressive NMT (NAT) to generate the target words in parallel. Formally, given the source sentence $\mathbf{x}$, the probability of the target sentence $\mathbf{y}$ is modeled as follows:

$$P_{\mathcal{NA}}(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = P_L(T|\mathbf{x};\boldsymbol{\theta}) \cdot \prod_{t=1}^{T} P(y_t|\mathbf{x};\boldsymbol{\theta}), \tag{19}$$

where $P_{\mathcal{NA}}(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ is the NAT model, $P_L(T|\mathbf{x};\boldsymbol{\theta})$ is a length sub-model to determine the length of target sentence, and $\boldsymbol{\theta}$ denotes the set of model parameters.

How to predict the length of target sentence (i.e., $P_L(T|\mathbf{x};\boldsymbol{\theta})$ in Eq. (19)) is critical for NAT. Gu et al. [29] proposed a fertility predictor to predict the length of translation. The fertility of a word in the source side determines how many target words it is aligned to. The fertility predictor can be denoted as

$$P_F(\mathbf{f}|\mathbf{x};\boldsymbol{\theta}) = \prod_{s=1}^{S} P(f_s|\mathbf{x};\boldsymbol{\theta}), \tag{20}$$

where $\mathbf{f} = \{f_1, \cdots, f_S\}$ is the fertility of the source sentence that consists of S words, and $\boldsymbol{\theta}$ is the set of parameters. At the training phase, the gold fertility of each sentence pair in the training data can be obtained by a word alignment system. At the inference phase, the length of the target sentence can be determined by the fertility predictor:

$$\hat{T} = \sum_{s=1}^{S} \hat{f}_s, \tag{21}$$

$$\hat{f}_s = \operatorname*{argmax}_{f_s} P(f_s|\mathbf{x};\hat{\boldsymbol{\theta}}), \tag{22}$$

where $\hat{T}$ is the number of words in the translation of the source sentence $\mathbf{x}$, and $\hat{\boldsymbol{\theta}}$ is the set of learned parameters.

Different from autoregressive NMT models that take the previous words (ie., $\mathbf{y}_{<t}$) as the input to predict the next target word y_t, NAT lacks such history information. Gu et al. [29] also noticed that missing the input of the decoder can greatly impair translation quality. Thus, the authors proposed to copy each source token to the decoder, and the times each input token to be copied is its "feritility". Gu et al. [29] also used knowledge distillation [42], which employs strong autoregressive models as the "teachers" to improve the performance. Knowledge distillation has proven necessary for non-autoregressive translation [43, 29, 44, 45, 46].

Despite the promising success of NAT, which can boost the decoding efficiency by about 15 times speedup compared with vanilla Transformer, NAT suffers from considerable quality degradation. Recently, many methods have been proposed to narrow the performance gap between non-autoregressive NMT and autoregressive NMT [44, 47, 48, 49, 50, 51, 52, 53, 46, 54].

To take advantage of both autoregressive NMT and non-autoregressive NMT, Wang et al. [47] designed a semi-autoregressive Transformer (SAT) model. SAT keeps the autoregressive property in global but performs parallel translation in local. Specifically, SAT produces K sequential words per time-step independently to others. Consequently, SAT can balance autoregressive NMT ($K = 1$) and non-autoregressive NMT ($K = T$) by adjusting the value of K. Akoury et al. [53] moved a further step to propose a syntactically supervised Transformer (SynST), which first autoregressively predicts a chunked parse tree and then generates all words in one shot conditioned on the predicted parse.

A critical issue of NAT is that NAT copies the source words as the input of the decoder while ignores the difference between the source and target semantics. To address this problem, Guo et al. [48] proposed to use a phrase table to covert source words

to target words. They adopt a maximum match algorithm to greedily segment the source sentence into several phrases and then map these source phrases into target phrases by retrieving a pre-defined phrase table. Thanks to the enhanced decoder input, translation quality is significantly improved.

Inspired by the mask-predict task proposed by Devlin et al. [55], Ghazvininejad et al. [46] introduced a conditioned masked language model (CMLM) to generate translation by iterative refinement. CMLM trains the conditioned language model using a mask-predict manner and produces target sentences by iterative decoding during inference. Specifically, in the training phase, CMLM first randomly masks the words in the target sentence and then predicts these masked words. In the inference, CMLM generates the entire target sentence in a preset number of decoding iteration N. At iteration $n \in [1, N]$, the decoder input is the entire target sentence with $T - \frac{T(N-t+1)}{N}$ words masked. The decoding process starts with a fully-masked target sentence and the words with the lowest prediction probabilities will be masked. With a proper number of decoding iteration, CMLM can effectively close the gap with fully autoregressive models and maintain the decoding efficiency.

2.4. Alternative Training Objectives

NMT trained with maximum likelihood estimation or MLE have achieved state-of-the-art results on various language pairs [7]. Despite the remarkable success, Ranzato et al. [56] indicate two drawbacks of MLE for NMT. First, NMT models are not exposed to their errors during training, which is referred to as the *exposure bias* problem. Second, MLE is defined at word-level rather than sentence-level. Due to these limitations, researchers have investigated several alternative objectives.

Ranzato et al. [57] introduce Mixed Incremental Cross-Entropy Reinforce (MIXER) for sequence-level training. The MIXER algorithm borrows ideas from reinforcement learning for backpropagating gradients from non-differentiable metrics such as BLEU. Wu et al. [58] give a study of reinforcement learning for NMT. Shen et al. [59] proposed minimum risk training (MRT) to alleviate the problem. In MRT, the risk is defined as the expected loss with respect to the posterior distribution:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{s=1}^{S} \mathbb{E}_{\mathbf{y}|\mathbf{x}^{(s)};\boldsymbol{\theta}} \left[\Delta(\mathbf{y}, \mathbf{y}^{(s)})\right] \tag{23}$$

$$= \sum_{s=1}^{S} \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}^{(s)})} P(\mathbf{y}|\mathbf{x}^{(s)}; \boldsymbol{\theta}) \Delta(\mathbf{y}, \mathbf{y}^{(s)}), \tag{24}$$

where $\mathcal{Y}(\mathbf{x}^{(s)})$ is a set of all possible candidate translations for $\mathbf{x}^{(s)}$; $\Delta(\mathbf{y}, \mathbf{y}^{(s)})$ measures the difference between model prediction and gold-standard. Shen et al. [59] indicate three advantages for MRT over MLE. Firstly, MRT direct optimize NMT with respect to evaluation metrics. Secondly, MRT can incorporate with arbitrary loss functions. Finally, MRT is transparent to architectures and can be applied to any end-to-end NMT systems. MRT achieves significant performance improvements than MLE training for RNNSearch. However, recent literature [60] has also pointed out the weakness of reinforcement learning for NMT, including discussion about optimization goals and difficulty in convergence.

Efforts on improving training objectives reveal the art of translating motivation into functions and rewrite the conventional loss function with them or integrating them into it as regularizers. A collection of classical structured prediction losses are reviewed and compared in Edunov et al. [61], including MLE, sequence-level MLE, MRT, and max-margin learning. Yang et al. [62] leveraged the idea of max-margin learning in reducing word omission errors in NMT. They artificially constructed negative examples by omitting words in target reference sentences, forcing the NMT model to assign a higher probability to a ground-truth translation and a lower probability to an erroneous translation. Wieting et al. [63] aimed at improving the semantic similarity between ground-truth references and translation outputs from NMT systems. They proposed to use a margin-based loss as an alternative reward function, encouraging NMT models to output semantically correct hypotheses even if they mismatch with the reference in the lexicon. Chen et al. [64] aimed at improving model capability of capturing long-range semantic structure. They proposed to explicitly model the source-target alignment with optimal transport (OT), and couple the OT loss with the MLE loss function. Kumar and Tsvetkov [65] aimed at improving model efficiency and reducing the memory footprint of NMT models. Observing that the softmax layer usually takes considerable memory usage and the longest computation time, they proposed to replace the softmax layer with a continuous embedding layer, using Von Mises-Fisher distribution to implement soft ranking as softmax layer functions. As a result, the novel probabilistic loss enables NMT models to train much faster and handle very large vocabularies.

2.5. Using Monolingual Data and Unsupervised NMT

The amount of parallel data significantly affects the training of parameters as NMT is found to be data-hungry [66]. Unfortunately, large-scale parallel corpora are not available for the vast majority of language pairs. In contrast, monolingual corpora are abundant and much easy to obtain. As a result, it is important to augment the training set with monolingual data.

2.5.1. Using Monolingual Data

As NMT is trained in an end-to-end way, it raises the difficulties in taking advantage of monolingual data. In the past few years researchers have proposed various methods to make use of the source- and target-side monolingual data in neural machine translation.

For target-side monolingual data, early attempts try to incorporate a language model trained on large-scale monolingual data into NMT. Gulcehre et al. [67] proposed two ways to integrate a language model. One way is called *shallow fusion*, which uses a language model during decoding to rescore the n-base list. Another way is called *deep fusion*, which combines the decoder and language model with a controller mechanism. However, the improvements of these approaches are limited.

Another way to use target-side monolingual data is called *Back-translation* (BT) [68]. BT can make use of target-side monolingual data without changing the architecture of NMT. In Sennrich et al. [68], they first trained a target-to-source translation model using the parallel corpus. Then, the target-side monolingual data are used to build a synthetic parallel corpus, whose source sides are generated by the target-to-source translation model. Finally, the concatenation of parallel corpus and synthetic parallel corpus is used to learn a source-to-target translation model. Although the architecture and decoding algorithm is kept unchanged, the monolingual data is fully utilized to improve the translation quality. The authors attributed the effectiveness of using monolingual data to domain adaptation effects, reductions of overfitting, and improved fluency. BT has shown to be the most simple and effective method to leverage target-side monolingual data [68, 69]. It is especially useful when only a small number of parallel data is available [70]. Imamura et al. [71] found that the diversities of source sentences affect the performance of BT. In the meantime, Edunov et al. [72] analyzed BT extensively and showed that noised-BT, which builds a synthetic corpus by sampled source sentences or noised output of beam-search, leads to higher accuracy. Caswell et al. [73] investigated the role of noise in noised-BT. They revealed that the noises work in a way of making the model be able to distinguish the synthetic data and genuine data. The model can further take advantages of helpful signal and ignore harmful signal. As a result, they proposed a simple method called tagged-BT, which appends a preceding tag (e.g., <BT>) to every synthetic source sentence. Wang et al. [74] proposed to consider uncertainty-based confidence to help NMT models distinguish synthetic data from authentic data.

Besides target-side monolingual data, source-side monolingual data are also important resources to improve the translation quality of semi-supervised neural machine translation. Zhang and Zong [75] explored two ways to leverage source-side monolingual data. The former one is knowledge distillation (also called self-training), which utilizes the source-to-target translation model to build a synthetic parallel corpus. The latter is multi-task learning that simultaneously learns translation and source sentence reordering tasks.

There are many works to make use of both source- and target-side monolingual data. Hoang et al. [76] found that the translation quality of the target-to-source model in BT matters and then proposed iterative back-translation, making the source-to-target and target-to-source to enhance each other iteratively. Cheng et al. [77] presented an approach to train a bidirectional neural machine translation model, which introduced autoencoders on the monolingual corpora with source-to-target and target-to-source translation models as encoders and decoders by appending a reconstruction term to the training objective. He et al. [78] proposed a dual-learning mechanism, which utilized reinforcement learning to make the source-to-target and target-to-source model to teach each other with the help of source- and target-side language models. Zheng et al. [79] proposed a mirror-generative NMT model to integrate source-to-target and target-to-source NMT models and both-side language models, which can learn from monolingual data naturally.

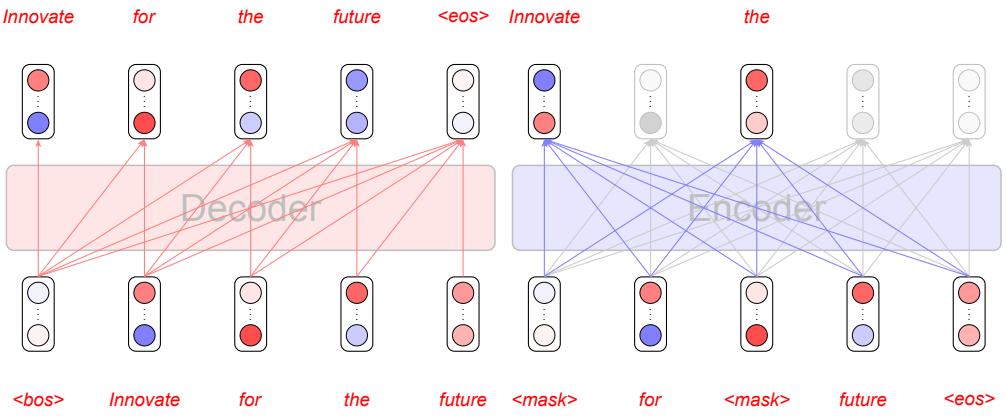
Pre-training is an alternative way to utilize monolingual data for NMT, which is shown to be beneficial by further combining with back-translation in the supervised and unsupervised NMT scenario [80, 81]. Recently, pre-training has attracted tremendous attention because of its effectiveness on low-resource language understanding and language generation tasks [82, 83, 55]. Researchers found that models trained on large-scale monolingual data can learn linguistics knowledge [84]. These knowledge can be transferred into downstream tasks by initializing the task-oriented models with the pre-trained weights. Language modeling is a commonly used pre-training method. The drawback of standard language modeling is that it is unidirectional, which may be sub-optimal as a pre-training technique. Devlin et al. [55] proposed a masked pre-training language model (MLM) objective, which allows the model to make full use of context at the price of losing the ability to generate sequences. Combining language modeling and masking with sequence-to-sequence models, however, do not suffer from these limitations [80, 85, 81].

Edunov et al. [86] fed the output representations of ELMO [82] to the encoder of NMT. Zhu et al. [87] proposed to fuse extracted representations into each layer of encoder and decoder through attention mechanism. Song et al. [80] proposed to pre-train a sequence-to-sequence model first, and then finetune the pre-trained model on translation task directly. BART [85] took various noising method to pre-train a denoising sequence-to-sequence model and then finetune the model with an additional encoder that replaces the word embeddings of the pre-trained encoder. Liu et al. [88] proposed mBART which is trained by applying BART to large-scale monolingual data across many languages.

2.5.2. Unsupervised NMT

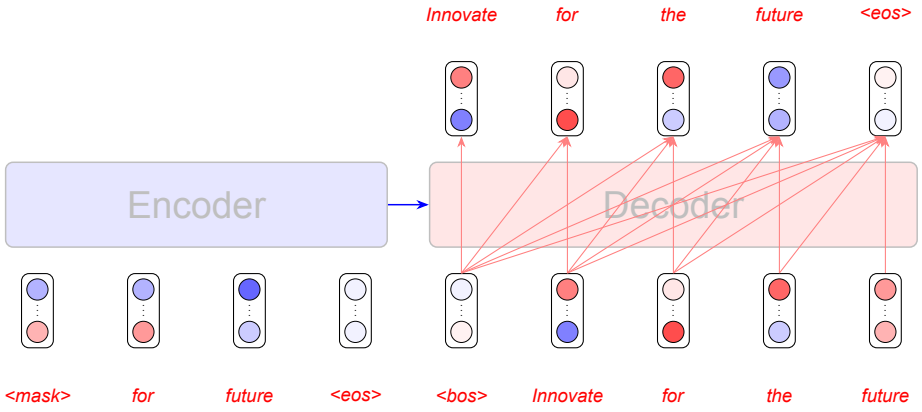
Due to insufficient parallel corpus, it is not feasible to use supervised methods to train an NMT model on many language pairs. Unsupervised neural machine translation aims to obtain a translation model without using parallel data. Apparently, unsupervised machine translation is much more difficult than the supervised and semi-supervised settings.

Unsupervised neural machine translation is composed of three parts. First, by the virtue of recent advances on unsupervised cross-lingual embeddings [89, 90] and word-by-word translation systems [91], the unsupervised translation models can be initialized by weak translation models with fundamental cross-lingual information. Second, denoising autoencoders [92] are used to embed the sentences into dense latent representations. The sentences of different languages are assumed to be embedded into the same latent space so that the latent representations of source sentences can be decoded into the target language. Third, iterative back-translation is used to strengthen the source-to-target and target-to-source translation models. Lample et al. [93] and Artetxe et al. [94] first successfully built an unsupervised NMT system as described above. Specifically, Lample et al. [93] utilizes a discriminator to force the encoder to embed sentences of each language to the same latent space.



(a) Unidirectional language model pre-training.

(b) Bi-directional language model pre-training.



(c) Sequence-to-sequence model pre-training.

Figure 13: Three commonly used ways for pre-training.

While Lample et al. [93] used a shared encoder and a shared decoder, Artetxe et al. [94] adopt a shared encoder but two separate decoder approach. Yang et al. [95] conjectured that sharing of the encoder and decoder between two languages may lose their language characteristics. Therefore they proposed leveraging two separate encoders with some shared layers and using two different GANs to restrict the latent representations. Artetxe et al. [96] and Lample et al. [97] found that an unsupervised statistical machine translation system with iterative back-translation can easily outperform the unsupervised NMT counterpart. Lample et al. [97] summarized that initialization, language modeling, and iterative back-translation are three principles in fully unsupervised MT and they further found that combining unsupervised SMT and unsupervised NMT can reach better performances.

Ren et al. [98] suggested that the noises and errors existed in pseudo-data can be accumulated and hinder the improvements during iterative back-translations. Therefore, they proposed to use SMT which is less sensitive to noises as posterior regularizations to unsupervised NMT. As the unsupervised NMT is usually initialized by unsupervised bilingual word embeddings (UBWE), Sun et al. [99] proposed to utilize UBWE agreement to enhance unsupervised NMT. Wu et al. [100] considered that pseudo sentences predicted by weak unsupervised MT systems are usually of low quality. To alleviate this issue, they proposed an extract-edit approach, which is an alternative to back-translation. First, they extracted the most relevant target sentences from target monolingual data given the source sentence. Then, extracted target sentences were edited to be aligned with the source sentences. This method makes it possible to use real sentence pairs to train the unsupervised NMT system. Ren et al. [101] also proposed a similar retrieve-and-rewrite method to initialize an unsupervised SMT system. Artetxe et al. [102] improved unsupervised SMT by exploiting subword information, developing a theoretically well-founded unsupervised tuning method, and incorporating a joint refinement procedure. Finally, they utilized the improved unsupervised SMT to initialize NMT model and get state-of-the-art results. As a unique method to utilize monolingual data, cross-lingual pre-trained models are used by Lample and Conneau [103] to initialize unsupervised MT systems.

2.6. Open Vocabulary

NMT typically operates with a fixed vocabulary. Due to practical reasons such as computational concerns and memory constraints, the vocabulary size of NMT models often ranges from 30k to 50k. For word-level NMT, the limited size of vocabulary results in a large number of unknown words. Therefore, word-level NMT is unable to translate these words and performs poorly in open-vocabulary settings [5, 6].

Although word-level NMT is unable to translate out-of-vocabulary words, character-level NMT do not have this problem. By splitting words into characters, the vocabulary size is much smaller and every rare word can be represented. Chung et al. [104] found that the NMT model with subword-level encoder and character-level decoder can also work well. Lee et al. [105] introduced a fully character-level NMT with convolutional network and found that character-to-character NMT is suitable in many-to-one multilingual setting. Luong and Manning [106] built hybrid systems that translate mostly at the word level and consult the character components for rare words. Passban et al. [107] proposed an extension to the model of Chung et al. [104], which works at the character level and boosts the decoder with target-side morphological information. Chen et al. [108] proposed an NMT model at different levels of granularity with a multi-level attention. Gao et al. [109] found that self-attention performs very well on character-level translation.

Character-level NMT also has its imperfection, splitting words into characters results in longer sequences in which each symbol contains less information, creating both modeling and computational challenges [110]. Other than word-level and character-level methods, subword-level method is another choice to model input and output sentences. Sennrich et al. [111] first adapted byte-pair-encoding (BPE) to word segmentation task, which is a simple but effective method. BPE making the NMT model capable of open-vocabulary translation by encoding rare and unknown words as sequences of subword units. This method reaches a compromise between vocabulary size and sequence length with stabilized better performance over word- and character-level methods. Moreover, it is an unsupervised method with few hyper-parameters, making it the most commonly used method for word segmentation for neural machine translation and text generation. Kudo [112] presented a simple regularization method, namely subword regularization, to improve the robustness of subword-level NMT. Provilkov et al. [113] introduced BPE-dropout to regularize the subword segmentation algorithm BPE, which is more compatible with conventional BPE than the method proposed by Kudo [112]. Wang et al. [114] investigated byte-level subwords, specifically byte-level BPE (BBPE), which is more efficient than using pure bytes only.

2.7. Prior Knowledge Integration

As NMT modeling the entire translation process with a neural network, it is hard to integrate knowledge into NMT. On the one hand, existing linguistic knowledge such as dictionaries is potentially useful for NMT. On the other hand, NMT often leads to over-translation and under-translation [115], which raises the need for adding prior knowledge to NMT.

One line of studies focus on inducing lexical knowledge into NMT models. Zhang et al. [116] proposed a general framework that can integrate prior knowledge into NMT models through posterior regularization and found that bilingual dictionary is useful to improve NMT models. Morishita et al. [117] found that feeding hierarchical subword units to different modules of NMT models can also improve the translation quality. Liu et al. [118] proposed a novel shared-private word embedding to capture the relationship of different words for NMT models. Chen et al. [119] distinguished content words and functional words depending on the term frequency inverse document frequency (i.e., *TF-IDF*) and then added an additional encoder and an additional loss for content words. Weller-Di Marco and Fraser [120] studied strategies to model word formation in NMT to explicitly model fusional morphology.

Modeling the source-side syntactic structure has also drawn a lot of attention. Eriguchi et al. [121] extended NMT models to an end-to-end syntactic model, where the decoder is softly aligned with phrases at the source side when generating a target word. Sennrich and Haddow [122] explored external linguistic information such as lemmas, morphological features, POS tags and dependency labels to improve translation quality. Hao et al. [123] presented a multi-granularity self-attention mechanism to model phrases which are extracted by syntactic trees. Bugliarello and Okazaki [124] proposed the Parent-Scaled Self-Attention to incorporate dependency tree to capture the syntactic knowledge of the source sentence. There are also some works that use multi-task training to learn source-side syntactic knowledge, in which the encoder of a NMT model is trained to perform POS tagging or syntactic parsing [125, 126].

Another line of studies directly model the target-side syntactic structures [127, 128, 129, 130, 131, 132, 133, 134]. Aharoni and Goldberg [130] trained a end-to-end model to directly translate source sentences into constituency trees. Similar approaches are proposed to use two neural models to generate the target sentence and its corresponding tree structure [128, 129]. Gū et al. [127] proposed to use a single model to perform translation and parsing at the same time. Yang et al. [133] introduced a latent variable model to capture the co-dependence between syntax and semantics. Yang et al. [134] trained a neural model to predict the soft template of the target sentence conditioning only on the source sentence and then incorporated the predicted template into the NMT model via a separate template encoder.

2.8. Interpretability and Robustness

Despite the remarkable progress, it is hard to interpret the internal workings of NMT models. All internal information in NMT is represented as high-dimensional real-valued vectors or matrices. Therefore, it is challenging to associate these hidden states with language structures. The lack of interpretability has made it very difficult for researchers to understand the translation process of NMT models.

In addition to interpretability, the lack of robustness is a severe challenge for NMT systems as well. With small perturbations in source inputs (also referred to as *adversarial examples*), the translations of NMT models may lead to significant erroneous changes [135, 136]. The lack of robustness of NMT limits its application on tasks that require robust performance on noisy inputs. Therefore, improving the robustness of NMT has gained increasing attention in the NMT community.

2.8.1. Interpretability

Efforts have been devoted to improving the interpretability of NMT systems in recent works. Ding et al. [137] proposed to

visualize the internal workings of the RNNSearch [6] architecture. With layer-wise relevance propagation [138], they computed and visualized the contribution of each contextual word to arbitrary hidden states in RNNSearch. Bau et al. [139] share similar motivations with Ding et al. [137]. Their basic assumption is that the same neuron in different NMT models captures similar syntactic and semantic information. They proposed to use several types of correlation coefficients to measure the importance of each neuron. As a result, by identifying important neurons and controlling their activation, the translation process of NMT systems can be controlled. Strobelt et al. [140] also put effort into visualizing the working process of RNNSearch. The highlights of their work lie in the utilization of training data. When an NMT system decodes some words, their visualization system provides the most relevant training corpora by using the nearest neighbor search. In case of translation errors, the system can locate the erroneous outputs directly in the training set by showing its origin cause. As a result, this function provides better assistants and makes it easy for developers to adjust the model and the data.

With the tremendous success of the Transformer architecture [15], the NMT community have shown increasing interest in understanding and interpreting Transformer. He et al. [141] generalized the idea of layer-wise relevance to word importance by attributing the NMT output to every input word through a gradient-based method. The calculated word importance illustrates the influence of each source words, which also serves as an implication of under-translation errors. Raganato and Tiedemann [142] analyzed the internal representations of Transformer encoder. Utilizing the attention weights in each layer, they extract relation among each word in the source sentence. They designed four types of probing tasks to analyze the syntactic and semantic information encoded by each layer representation and test their transferability. Voita et al. [143] also proposed to analyze the bottom-up evolution of representations in Transformer with canonical correlation analysis (CCA). By estimating mutual information, they studied how information flows in Transformer. Stahlberg et al. [144] proposed an operation sequence model to interpret NMT. Based on the translation outputted by the Transformer system, they proposed explicit modeling of the word reordering process and provided explicit word alignment between the reordered target-side sentence and the source sentence. As a result, one can track the reordering process of each word's information as they are explicitly aligned with the source side. Recent work [145] also provided a theoretical understanding of Transformer by proving that Transformer networks are universal approximators of sequence-to-sequence functions.

2.8.2. Robustness

Belinkov and Bisk [135] first investigated the robustness of NMT. They pointed out that both synthetic and natural noise can severely harm the performance of NMT models. They experimented with four types of synthetic noise and leveraged structure-invariant representation and adversarial training to improve the robustness of NMT. Similarly, Zhao et al. [146] proposed to map the input sentence to a latent space with generative adversarial networks (GAN) and search for adversarial examples in that space. Their approach can produce semantically and syntactically coherent sentences that have negative impacts on the performance of NMT models.

Ribeiro et al. [147] proposed semantic-preserving adversarial rules to explicitly induce adversarial examples. This approach provides a better guarantee for the adversarial examples to satisfy semantically equivalence property. Cheng et al. [148] proposed two types of approaches to generating adversarial examples by perturbing the source sentence or the internal representation of the encoder. By integrating the effect of adversarial examples into the loss function, the robustness of neural machine translation is improved by adversarial training.

Ebrahimi et al. [149] proposed a character-level white-box attack for character-level NMT. They proposed to model the operations of character insertion, deletion, and swapping with vector computations so that the generation of adversarial examples can be formulated with differentiable string-edit operations. Liu et al. [88] proposed to jointly utilize textual and phonetic embedding in NMT to improve robustness. They found that to train a more robust model, more weights should be put on the phonetic rather than textual information. Cheng et al. [136] proposed doubly adversarial inputs to improve the robustness of NMT. Concretely, they proposed to both attack the translation model with adversarial source examples and defend the translation model with adversarial target inputs for model robustness. Zou et al. [150] utilized reinforcement learning to generate adversarial examples, producing stable attacks with semantic-preserving adversarial examples. Cheng et al. [151] proposed a novel adversarial augmentation method that minimizes the vicinal risk over virtual sentences sampled from a smoothly interpolated embedding space around the observed training sentence pairs. The adversarial data augmentation method substantially outperforms other data augmentation methods and achieves significant improvements in translation quality and robustness. For the better exploration of robust NMT, Michel and Neubig [152] proposed an MTNT dataset, source sentences of which are collected from Reddit discussion, and contain several types of noise. Target referenced translations for each source sentence, in contrast, are clear from noise. Experiments showed that current NMT models perform badly on the MTNT dataset. As a result, this dataset can serve as a testbed for NMT robustness analysis.

3. Resources

3.1. Parallel Data

Bilingual parallel corpora are the most important resources for NMT. There are several publicly available corpora, such as the datasets provided by WMT[1], IWSLT[2], and WAT [3]. Table 3 lists the available domains and language pairs in these workshops.

[1] http://www.statmt.org/wmt20/index.html
[2] http://iwslt.org/doku.php
[3] http://lotus.kuee.kyoto-u.ac.jp/WAT/WAT2020/index.html

Workshop	Domain	Language Pair
WMT20	News	zh-en, cz-en, fr-de, de-en, iu-en, km-en, ja-en, ps-en, pl-en, ru-en, ta-en
	Biomedical	en-eu, en-zh, en-fr, en-de, en-it, en-pt, en-ru, en-es
	Chat	en-de
IWSLT20	TED Talks	en-de
	e-Commerce	zh-en, en-ru
	Open Domain	zh-ja
WAT20	Scientific Paper	en-ja, zh-ja
	Business Scene Dialogue	en-ja
	Patent	zh-ja, ko-ja, en-ja
	News	ja-en, ja-ru
	IT and Wikinews	hi-en, th-en, ms-en, id-en

Table 3: Domain and language pairs provided by WMT20, IWSLT20, WAT20.

Source	Fr-En	Es-En	De-En	Pt-En	Ru-En	Ar-En	Zh-En	Ja-En	Hi-En
OPUS [153]	200.6M	172.0M	93.3M	77.7M	75.5M	69.2M	31.2M	6.2M	1.7M

Table 4: Number of sentences that available at OPUS for major languages to English.

Besides the aforementioned machine translation workshops, we also recommend OPUS [4] to search resources for training NMT models, which gathers parallel data for a large number of language pairs. We list the number of sentence pairs that are available for major languages to Enligsh in Table 4. OPUS also provides the OPUS-100 corpus for multilingual machine translation research [154], which is an English-centric multilingual corpus covering over 100 languages.

3.2. Monolingual Data

Monolingual data are also valuable resources for NMT. The Common Crawl Foundation [5] provides open access to high quality crawled data for over 40 languages. The CCNET toolkit [6] [155] can be used to download and clean Common Crawl texts. Wikipedia provides database dump [7] that can be used to extract monolingual data, which can be download using WIKIEXTRACTOR [8]. WMT 2020 also provides several monolingual training data, which consists of data collected from NewsCrawl, NewsDicussions, Europarl, NewsCommentary, CommonCrawl, and WikiDumps.

4. Tools

With the rapid advances of deep learning, many open-source deep learning frameworks have emerged, with TensorFlow [156] and PyTorch [157] as representative examples. At the same time, we have also witnessed the rapid development of open-source NMT toolkits, which significantly boosted the research progress of NMT. In this section, we will give a summarization of popular open-source NMT toolkits. Besides, we also introduce tools that are useful for evaluation, analysis, and data pre-processing.

4.1. Open-source NMT Toolkits

We summarize some popular open-source NMT toolkits on GitHub in Table 5. The users can get the source codes of these toolkits directly from GitHub. We shall give a brief description of these projects.

Tensor2Tensor. TENSOR2TENSOR [158] is a library of deep learning models and datasets based on TensorFlow [156]. The library was mainly developed by the Google Brain team. TENSOR2TENSOR provides implementation of several NMT architectures (e.g., Transformer) for the translation task. The users can run TENSOR2TENSOR easily on CPU, GPU, and TPU, either locally or on Cloud.

FairSeq. FAIRSEQ [159] is a sequence modeling toolkit developed by Facebook AI Research. The toolkit is based on Pytorch [157] and allows the users to train custom models for the translation task. FAIRSEQ implements traditional RNN-based models and Transformer models. Besides, it also includes CNN-based translation models (e.g., LightConv and DynamicConv).

Nmt. NMT [160] is a toolkit developed by Google Research. The toolkit implements the GNMT architecture [8]. Besides, the NMT project also provides a nice tutorial for building a competitive NMT model from scratch. The codebase of NMT is high-quality and lightweight, which is friendly for users to add customized models.

OpenNMT. OPENNMT is an open-source NMT toolkit developed by the collaboration of Harvard University and SYSTRAN. The toolkit currently maintains two implementations: OPENNMT-PY and OPENNMT-TF. OPENNMT is proven to be research-friendly and production-ready. The OpenNMT project also provides CTRANSLATE2 as a fast inference engine that supports both CPU and GPU.

[4] http://opus.nlpl.eu
[5] https://commoncrawl.org/
[6] https://github.com/facebookresearch/cc_net
[7] https://dumps.wikimedia.org
[8] https://github.com/attardi/wikiextractor

Name	Language	Framework	Status
TENSOR2TENSOR	Python	TensorFlow	Deprecated
FAIRSEQ	Python	PyTorch	Active
NMT	Python	TensorFlow	Deprecated
OPENNMT	Python/C++	PyTorch/TensorFlow	Active
SOCKEYE	Python	MXNet	Active
NEMATUS	Python	Tensorflow	Active
MARIAN	C++	-	Active
THUMT	Python	PyTorch/TensorFlow	Active
NMT-KERAS	Python	Keras	Active
NEURAL MONKEY	Python	TensorFlow	Active

Table 5: Popular Open-source NMT toolkits on GitHub, the ordering is determined by the number of stars as the date of December 2020.

Sockeye. SOCKEYE [161] is a versatile sequence-to-sequence toolkit that is based on MXNet [162]. SOCKEYE is maintained by Amazon and powers machine translation services such as Amazon Translate. The toolkit features state-of-the-art machine translation models and fast CPU inference, which is useful for both research and production.

Nematus. NEMATUS is an NMT toolkit developed by the NLP Group at the University of Edinburgh. The toolkit is based on TensorFlow and supports RNN-based NMT architectures as well as the TRANSFORMER architecture. In addition to the toolkits, NEMATUS also released high-performing NMT models covering 13 translation directions.

Marian. MARIAN [163] is an efficient and self-contained NMT framework currently being developed by the Microsoft Translator team. The framework is written entirely in C++ with minimal dependencies. Marian is widely deployed by many companies and organizations. For example, Microsoft Translator currently adopts Marian as its neural machine translation engine.

THUMT. THUMT [164] is an open-source toolkit for neural machine translation developed by the NLP Group at Tsinghua University. The toolkit includes TensorFlow, and Pytorch implementations. It supports vanilla RNN-based and Transformer models and is easy for users to build new models. Furthermore, THUMT provides visualization analysis using layer-wise relevance propagation [137].

NMT-Keras. NMT-KERAS [165] is a flexible toolkit for neural machine translation developed by the Pattern Recognition and Human Language Technology Research Center at Polytechnic University of Valencia. The toolkit is based on Keras which uses Theano or TensorFlow as the backend. NMT-KERAS emphasizes the development of advanced applications for NMT systems, such as interactive NMT and online learning. It also has been extended to other tasks including image and video captioning, sentence classification, and visual question answering.

Neural Monkey NEURAL MONKEY is an open-source neural machine translation and general sequence-to-sequence learning system. The toolkit is built on the TensorFlow library and provides a high-level API tailored for fast prototyping of complex architectures.

4.2. Tools for Evaluation and Analysis

Manual evaluation of MT outputs is not only expensive but also impractical to scaling for large language pairs. On the contrary, automatic MT evaluation is inexpensive and language-independent, with BLEU [166] as the representative automatic evaluation metric. Besides evaluation, there is also a need for analyzing MT outputs. We recommend the following tools for evaluating and analyzing MT output.

SACREBLEU. SACREBLEU [9] [167] is a toolkit to compute shareable, comparable, and reproducible BLEU scores. SACREBLEU computes BLEU scores on detokenized outputs, using WMT standard tokenization. As a result, the scores are not affected by different processing tools. Besides, it can produce a short version string that facilitates cross-paper comparisons.

COMPARE-MT. COMPARE-MT[10] [168] is a program to compare the outputs of multiple systems for language generation. In order to provide high-level analysis of outputs, it enables analysis of accuracy of generation of particular types of words, bucketed histograms of sentence accuracies or counts based on salient characteristics, and so on.

MT-COMPAREVAL. MT-COMPAREVAL[11] is also a tool for comparison and evaluation of machine translations. It allows users to compare translations according to automatic metrics or quality comparison from the aspects of n-grams.

4.3. Other Tools

Asides from the above mentioned tools, we found the following toolkits are very useful for NMT research and deployment.

MOSES. MOSES[12][169] is a self-contained statistical machine translation toolkit. Besides SMT-related components, MOSES provides a large number of tools to clean and preprocess texts, which are also useful for training NMT models. MOSES also contains several easy-to-use scripts to analyze and evaluate MT outputs.

[9] https://github.com/mjpost/sacrebleu
[10] https://github.com/neulab/compare-mt
[11] https://github.com/ondrejklejch/MT-ComparEval
[12] https://github.com/moses-smt/mosesdecoder

SUBWORD-NMT. SUBWORD-NMT[13] is an open-source toolkit for unsupervised word segmentation for neural machine translation and text generation. It adopts the Byte-Pair Encoding (BPE) algorithm proposed by [111] and BPE dropout proposed by [113]. It is the most commonly used toolkit to alleviate the out-of-vocabulary problem in NMT.

SENTENCEPIECE. SENTENCEPIECE[14] is a powerful unsupervised text segmentation toolkit. SENTENCEPIECE is written in C++ and provides APIs for other languages such as Python. SENTENCEPIECE implements the BPE algorithm [111] and unigram language model [112]. Unlike SUBWORD-NMT, SENTENCEPIECE can learn to segment raw texts without additional pre-processing. As a result, SENTENCEPIECE is a suitable choice to segment multilingual texts.

5. Conclusion

Neural machine translation has become the dominant approach to machine translation in both research and practice. This article reviewed the widely used methods in NMT, including modeling, decoding, data augmentation, interpretation, as well as evaluation. We then summarize the resources and tools that are useful for NMT research.

Despite the great success achieved by NMT, there are still many problems to be explored. We list some important and challenging problems for NMT as follows:

- **Understanding NMT.** Although there are many attempts to analyze and interpret NMT, our understandings about NMT are still limited. Understanding how and why NMT produces its translation result is important to figure out the bottleneck and weakness of NMT models.
- **Designing better architectures**. Designing a new architecture that better than Transformer is beneficial for both NMT research and production. Furthermore, designing a new architecture that balances translation performance and computational complexity is also important.
- **Making full use of monolingual data.** Monolingual data are valuable resources. Despite the remarkable progress, we believe that there is still much room for NMT to make use of abundant monolingual data.
- **Integrating prior knowledge.** Incorporating human knowledge into NMT is also an important problem. Although there is some progress, the results are far from satisfactory. How to convert discrete and continuous representations into each other is a problem of NMT that needs further exploration.

Acknowledgements

This work was supported by the National Key R&D Program of China (No. 2017YFB0 202204), National Natural Science Foundation of China (No. 61925601, No. 61761166 008, No. 61772302), Beijing Academy of Artificial Intelligence, Huawei Noah's Ark Lab, and the NExT++ project supported by the National Research Foundation, Prime Ministers Office, Singapore under its IRC@Singapore Funding Initiative. We thank Kehai Chen and Xiangwen Zhang for their kindly suggestions.

[13] https://github.com/rsennrich/subword-nmt

[14] https://github.com/google/sentencepiece

References

[1] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. Lafferty, R. L. Mercer, P. S. Roossin, A statistical approach to machine translation, Computational linguistics 16 (1990) 79–85.

[2] P. Koehn, F. J. Och, D. Marcu, Statistical phrase-based translation, Technical Report, UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST, 2003.

[3] N. Kalchbrenner, P. Blunsom, Recurrent continuous translation models, in: Proceedings of EMNLP, 2013, pp. 1700–1709.

[4] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder–decoder for statistical machine translation, in: Proceedings of EMNLP, 2014, pp. 1724–1734.

[5] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Proceedings of NeurIPS, 2014, pp. 3104–3112.

[6] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: Proceedings of ICLR, 2015.

[7] M. Junczys-Dowmunt, T. Dwojak, H. Hoang, Is neural machine translation ready for deployment? a case study on 30 translation directions, arXiv preprint arXiv:1610.01108 (2016).

[8] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).

[9] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li, et al., Achieving human parity on automatic chinese to english news translation, arXiv preprint arXiv:1803.05567 (2018).

[10] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[11] K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, arXiv preprint arXiv:1409.1259 (2014).

[12] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. (1994).

[13] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, K. Kavukcuoglu, Neural machine translation in linear time, arXiv preprint arXiv:1610.10099 (2016).

[14] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y. N. Dauphin, Convolutional sequence to sequence learning, arXiv preprint arXiv:1705.03122 (2017).

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of NeurIPS, 2017, pp. 5998–6008.

[16] A. Graves, G. Wayne, I. Danihelka, Neural turing machines, arXiv preprint arXiv:1410.5401 (2014).

[17] M.-T. Luong, H. Pham, C. D. Manning, Effective approaches to attention-based neural machine translation, arXiv preprint arXiv:1508.04025 (2015).

[18] H. T. Siegelmann, E. D. Sontag, On the computational power of neural nets, Journal of computer and system sciences (1995).

[19] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation (1997).

[20] J. L. Ba, J. R. Kiros, G. E. Hinton, Layer normalization, arXiv preprint arXiv:1607.06450 (2016).

[21] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of CVPR, 2016, pp. 770–778.

[22] J. Zhou, W. Xu, End-to-end learning of semantic role labeling using recurrent neural networks, in: Proceedings of ACL, 2015, pp. 1127–1137.

[23] L. Kaiser, A. N. Gomez, F. Chollet, Depthwise separable convolutions for neural machine translation, arXiv preprint arXiv:1706.03059 (2017).

[24] F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, M. Auli, Pay less attention with lightweight and dynamic convolutions, arXiv preprint arXiv:1901.10430 (2019).

[25] L. Liu, M. Utiyama, A. Finch, E. Sumita, Agreement on target-bidirectional neural machine translation, in: Proceedings of NAACL-HLT, 2016, pp. 411–416.

[26] C. D. V. Hoang, G. Haffari, T. Cohn, Towards decoding as continuous optimisation in neural machine translation, in: Proceedings of EMNLP, 2017, pp. 146–156.

[27] X. Zhang, J. Su, Y. Qin, Y. Liu, R. Ji, H. Wang, Asynchronous bidirectional decoding for neural machine translation, in: Proceedings of AAAI, 2018, pp. 5698–5705.

[28] L. Zhou, J. Zhang, C. Zong, Synchronous bidirectional neural machine translation, TACL (2019).

[29] J. Gu, J. Bradbury, C. Xiong, V. O. Li, R. Socher, Non-autoregressive neural machine translation, in: Proceedings of ICLR, 2018.

[30] Z. Zheng, H. Zhou, S. Huang, L. Mou, X. Dai, J. Chen, Z. Tu, Modeling past and future for neural machine translation, Transactions of the Association for Computational Linguistics 6 (2018) 145–157.

[31] Z. Zheng, S. Huang, Z. Tu, X.-Y. Dai, C. Jiajun, Dynamic past and future for neural machine translation, in: Proceedings of EMNLP-IJCNLP, 2019, pp. 930–940.

[32] B. Zhang, D. Xiong, J. Su, J. Luo, Future-aware knowledge distillation for neural machine translation, IEEE/ACM Transactions on Audio, Speech, and Language Processing 27 (2019) 2278–2287.

[33] J. Zhang, C. Zong, Neural machine translation: Challenges, progress and future, arXiv preprint arXiv:2004.05809 (2020).

[34] Z. Yang, L. Chen, M. Le Nguyen, Regularizing forward and backward decoding to improve neural machine translation, in: Proceedings of International Conference on Knowledge and Systems Engineering (KSE), 2018, pp. 73–78.

[35] Z. Zhang, S. Wu, S. Liu, M. Li, M. Zhou, T. Xu, Regularizing neural machine translation by target-bidirectional agreement, in: Proceedings of AAAI, volume 33, 2019, pp. 443–450.

[36] R. Sennrich, B. Haddow, A. Birch, Edinburgh neural machine translation systems for wmt 16, in: Proceedings of WMT, 2016, pp. 371–376.

[37] J. Su, X. Zhang, Q. Lin, Y. Qin, J. Yao, Y. Liu, Exploiting reverse target-side contexts for neural machine translation via asynchronous bidirectional decoding, Artificial Intelligence 277 (2019) 103168.

[38] L. Zhou, J. Zhang, C. Zong, H. Yu, Sequence generation: from both sides to the middle, in: Proceedings of IJCAI, 2019, pp. 5471–5477.

[39] J. Zhang, L. Zhou, Y. Zhao, C. Zong, Synchronous bidirectional inference for neural sequence generation, Artificial Intelligence 281 (2020) 103234.

[40] R. Sennrich, A. Birch, A. Currey, U. Germann, B. Haddow, K. Heafield, A. V. M. Barone, P. Williams, The university of edinburgh's neural mt systems for wmt17, in: Proceedings of WMT, 2017, pp. 389–399.

[41] S. Mehri, L. Sigal, Middle-out decoding, in: Advances in NeurIPS, 2018, pp. 5518–5529.

[42] Y. Kim, A. M. Rush, Sequence-level knowledge distillation, in: Proceedings of EMNLP, 2016, pp. 1317–1327.

[43] C. Zhou, J. Gu, G. Neubig, Understanding knowledge distillation in non-autoregressive machine translation, in: Proceedings of ICLR, 2019.

[44] J. Lee, E. Mansimov, K. Cho, Deterministic non-autoregressive neural sequence modeling by iterative refinement, in: Proceedings of EMNLP, 2018, pp. 1173–1182.

[45] J. Libovickỳ, J. Helcl, End-to-end non-autoregressive neural machine translation with connectionist temporal classification, in: Proceedings of EMNLP, 2018, pp. 3016–3021.

[46] M. Ghazvininejad, O. Levy, Y. Liu, L. Zettlemoyer, Mask-predict: Parallel decoding of conditional masked language models, in: Proceedings of EMNLP-IJCNLP, 2019, pp. 6114–6123.

[47] C. Wang, J. Zhang, H. Chen, Semi-autoregressive neural machine translation, in: Proceedings of EMNLP, 2018, pp. 479–488.

[48] J. Guo, X. Tan, D. He, T. Qin, L. Xu, T.-Y. Liu, Non-autoregressive neural machine translation with enhanced decoder input, in: Proceedings of AAAI, volume 33, 2019, pp. 3723–3730.

[49] C. Shao, Y. Feng, J. Zhang, F. Meng, X. Chen, J. Zhou, Retrieving sequential information for non-autoregressive neural machine translation, in: Proceedings of ACL, 2019, pp. 3013–3024.

[50] Y. Wang, F. Tian, D. He, T. Qin, C. Zhai, T.-Y. Liu, Non-autoregressive machine translation with auxiliary regularization, in: Proceedings of AAAI, volume 33, 2019, pp. 5377–5384.

[51] M. Stern, W. Chan, J. Kiros, J. Uszkoreit, Insertion transformer: Flexible sequence generation via insertion operations, in: Proceedings of ICML, 2019, pp. 5976–5985.

[52] B. Wei, M. Wang, H. Zhou, J. Lin, X. Sun, Imitation learning for non-autoregressive neural machine translation, in: Proceedings of ACL, 2019, pp. 1304–1312.

[53] N. Akoury, K. Krishna, M. Iyyer, Syntactically supervised transformers for faster neural machine translation, in: Proceedings of ACL, 2019, pp. 1269–1281.

[54] J. Gu, Q. Liu, K. Cho, Insertion-based decoding with automatically inferred generation order, TACL 7 (2019) 661–676.

[55] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of NAACL-HLT, 2019, pp. 4171–4186.

[56] M. Ranzato, S. Chopra, M. Auli, W. Zaremba, Sequence level training with recurrent neural networks, arXiv preprint arXiv:1511.06732 (2015).

[57] M. Ranzato, S. Chopra, M. Auli, W. Zaremba, Sequence level training with recurrent neural networks, in: Proceedings of ICLR, 2016.

[58] L. Wu, F. Tian, T. Qin, J. Lai, T.-Y. Liu, A study of reinforcement learning for neural machine translation, in: Proceedings of EMNLP, 2018, pp. 3612–3621.

[59] S. Shen, Y. Cheng, Z. He, W. He, H. Wu, M. Sun, Y. Liu, Minimum risk training for neural machine translation, in: Proceedings of ACL, 2016, pp. 1683–1692.

[60] L. Choshen, L. Fox, Z. Aizenbud, O. Abend, On the weaknesses of reinforcement learning for neural machine translation, in: Proceedings of ICLR, 2020.

[61] S. Edunov, M. Ott, M. Auli, D. Grangier, M. Ranzato, Classical structured prediction losses for sequence to sequence learning, in: Proceedings of NAACL-HLT, 2018, pp. 355–364.

[62] Z. Yang, Y. Cheng, Y. Liu, M. Sun, Reducing word omission errors in neural machine translation: A contrastive learning approach, in: Proceedings of ACL, 2019, pp. 6191–6196.

[63] J. Wieting, T. Berg-Kirkpatrick, K. Gimpel, G. Neubig, Beyond BLEU:training neural machine translation with semantic similarity, in: Proceedings of ACL, 2019, pp. 4344–4355.

[64] L. Chen, Y. Zhang, R. Zhang, C. Tao, Z. Gan, H. Zhang, B. Li, D. Shen, C. Chen, L. Carin, Improving sequence-to-sequence learning via optimal transport, in: Proceedings of ICLR, 2019.

[65] S. Kumar, Y. Tsvetkov, Von mises-fisher loss for training sequence to sequence models with continuous outputs, in: Proceedings of ICLR, 2019.

[66] B. Zoph, D. Yuret, J. May, K. Knight, Transfer learning for low-resource neural machine translation, arXiv preprint arXiv:1604.02201 (2016).

[67] C. Gulcehre, O. Firat, K. Xu, K. Cho, Y. Bengio, On integrating a language model into neural machine translation, Computer Speech & Language 45 (2017) 137–148.

[68] R. Sennrich, B. Haddow, A. Birch, Improving neural machine translation models with monolingual data, in: Proceedings of ACL, 2016, pp. 86–96.

[69] A. Poncelas, D. Shterionov, A. Way, G. de Buy Wenniger, P. Passban, Investigating backtranslation in neural machine translation, arXiv preprint arXiv:1804.06189 (2018).

[70] A. Karakanta, J. Dehdari, J. van Genabith, Neural machine translation for low-resource languages without parallel corpora, Machine Translation 32 (2018) 167–189.

[71] K. Imamura, A. Fujita, E. Sumita, Enhancement of encoder and attention using target monolingual corpora in neural machine translation, in: Proceedings of the 2nd Workshop on Neural Machine Translation and Generation, 2018, pp. 55–63.

[72] S. Edunov, M. Ott, M. Auli, D. Grangier, Understanding back-translation at scale, arXiv preprint arXiv:1808.09381 (2018).

[73] I. Caswell, C. Chelba, D. Grangier, Tagged back-translation, WMT 2019 (2019) 53.

[74] S. Wang, Y. Liu, C. Wang, H. Luan, M. Sun, Improving back-translation with uncertainty-based confidence estimation, arXiv preprint arXiv:1909.00157 (2019).

[75] J. Zhang, C. Zong, Exploiting source-side monolingual data in neural

machine translation, in: Proceedings of EMNLP, 2016, pp. 1535–1545.
[76] V. C. D. Hoang, P. Koehn, G. Haffari, T. Cohn, Iterative back-translation for neural machine translation, in: Proceedings of the 2nd Workshop on Neural Machine Translation and Generation, 2018, pp. 18–24.
[77] Y. Cheng, W. Xu, Z. He, W. He, H. Wu, M. Sun, Y. Liu, Semi-supervised learning for neural machine translation, in: Proceedings of ACL, 2016, pp. 1965–1974.
[78] D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T.-Y. Liu, W.-Y. Ma, Dual learning for machine translation, in: Advances in NeurIPS, 2016, pp. 820–828.
[79] Z. Zheng, H. Zhou, S. Huang, L. Li, X.-Y. Dai, J. Chen, Mirror-generative neural machine translation, in: Proceedings of ICLR, 2020.
[80] K. Song, X. Tan, T. Qin, J. Lu, T.-Y. Liu, Mass: Masked sequence to sequence pre-training for language generation, arXiv preprint arXiv:1905.02450 (2019).
[81] Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, L. Zettlemoyer, Multilingual denoising pre-training for neural machine translation, arXiv preprint arXiv:2001.08210 (2020).
[82] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, arXiv preprint arXiv:1802.05365 (2018).
[83] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, OpenAI Blog 1 (2019) 9.
[84] K. Clark, U. Khandelwal, O. Levy, C. D. Manning, What does bert look at? an analysis of bert's attention, in: Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, 2019, pp. 276–286.
[85] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, arXiv preprint arXiv:1910.13461 (2019).
[86] S. Edunov, A. Baevski, M. Auli, Pre-trained language model representations for language generation, arXiv preprint arXiv:1903.09722 (2019).
[87] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. Zhou, H. Li, T.-Y. Liu, Incorporating bert into neural machine translation, arXiv preprint arXiv:2002.06823 (2020).
[88] H. Liu, M. Ma, L. Huang, H. Xiong, Z. He, Robust neural machine translation with joint textual and phonetic embedding, in: Proceedings of ACL, 2019, pp. 3044–3049.
[89] M. Zhang, Y. Liu, H. Luan, M. Sun, Adversarial training for unsupervised bilingual lexicon induction, in: Proceedings of ACL, 2017, pp. 1959–1970.
[90] M. Artetxe, G. Labaka, E. Agirre, Learning bilingual word embeddings with (almost) no bilingual data, in: Proceedings of ACL, 2017, pp. 451–462.
[91] A. Conneau, G. Lample, M. Ranzato, L. Denoyer, H. Jégou, Word translation without parallel data, arXiv preprint arXiv:1710.04087 (2017).
[92] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of ICML, 2008, pp. 1096–1103.
[93] G. Lample, A. Conneau, L. Denoyer, M. Ranzato, Unsupervised machine translation using monolingual corpora only, arXiv preprint arXiv:1711.00043 (2017).
[94] M. Artetxe, G. Labaka, E. Agirre, K. Cho, Unsupervised neural machine translation, arXiv preprint arXiv:1710.11041 (2017).
[95] Z. Yang, W. Chen, F. Wang, B. Xu, Unsupervised neural machine translation with weight sharing, arXiv preprint arXiv:1804.09057 (2018).
[96] M. Artetxe, G. Labaka, E. Agirre, Unsupervised statistical machine translation, arXiv preprint arXiv:1809.01272 (2018).
[97] G. Lample, M. Ott, A. Conneau, L. Denoyer, M. Ranzato, Phrase-based & neural unsupervised machine translation, arXiv preprint arXiv:1804.07755 (2018).
[98] S. Ren, Z. Zhang, S. Liu, M. Zhou, S. Ma, Unsupervised neural machine translation with smt as posterior regularization, in: Proceedings of the AAAI, volume 33, 2019, pp. 241–248.
[99] H. Sun, R. Wang, K. Chen, M. Utiyama, E. Sumita, T. Zhao, Unsupervised bilingual word embedding agreement for unsupervised neural machine translation, in: Proceedings of ACL, 2019, pp. 1235–1245.
[100] J. Wu, X. Wang, W. Y. Wang, Extract and edit: An alternative to back-translation for unsupervised neural machine translation, arXiv preprint arXiv:1904.02331 (2019).
[101] S. Ren, Y. Wu, S. Liu, M. Zhou, S. Ma, A retrieve-and-rewrite initialization method for unsupervised machine translation, in: Proceedings of ACL, 2020, pp. 3498–3504.
[102] M. Artetxe, G. Labaka, E. Agirre, An effective approach to unsupervised machine translation, arXiv preprint arXiv:1902.01313 (2019).
[103] G. Lample, A. Conneau, Cross-lingual language model pretraining, arXiv preprint arXiv:1901.07291 (2019).
[104] J. Chung, K. Cho, Y. Bengio, A character-level decoder without explicit segmentation for neural machine translation, arXiv preprint arXiv:1603.06147 (2016).
[105] J. Lee, K. Cho, T. Hofmann, Fully character-level neural machine translation without explicit segmentation, Transactions of the Association for Computational Linguistics 5 (2017) 365–378.
[106] M.-T. Luong, C. D. Manning, Achieving open vocabulary neural machine translation with hybrid word-character models, arXiv preprint arXiv:1604.00788 (2016).
[107] P. Passban, Q. Liu, A. Way, Improving character-based decoding using target-side morphological information for neural machine translation, arXiv preprint arXiv:1804.06506 (2018).
[108] H. Chen, S. Huang, D. Chiang, X. Dai, J. Chen, Combining character and word information in neural machine translation using a multi-level attention, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018, pp. 1284–1293.
[109] Y. Gao, N. I. Nikolov, Y. Hu, R. H. Hahnloser, Character-level translation with self-attention, arXiv preprint arXiv:2004.14788 (2020).
[110] C. Cherry, G. Foster, A. Bapna, O. Firat, W. Macherey, Revisiting character-based neural machine translation with capacity and compression, arXiv preprint arXiv:1808.09943 (2018).
[111] R. Sennrich, B. Haddow, A. Birch, Neural machine translation of rare words with subword units, in: Proceedings of ACL, 2016.
[112] T. Kudo, Subword regularization: Improving neural network translation models with multiple subword candidates, arXiv preprint arXiv:1804.10959 (2018).
[113] I. Provilkov, D. Emelianenko, E. Voita, Bpe-dropout: Simple and effective subword regularization, arXiv preprint arXiv:1910.13267 (2019).
[114] C. Wang, K. Cho, J. Gu, Neural machine translation with byte-level subwords, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 9154–9160.
[115] Z. Tu, Z. Lu, Y. Liu, X. Liu, H. Li, Modeling coverage for neural machine translation, in: Proceedings of ACL, 2016.
[116] J. Zhang, Y. Liu, H. Luan, J. Xu, M. Sun, Prior knowledge integration for neural machine translation using posterior regularization, in: Proceedings of ACL, 2017, pp. 1514–1523.
[117] M. Morishita, J. Suzuki, M. Nagata, Improving neural machine translation by incorporating hierarchical subword features, in: Proceedings of COLING, 2018, pp. 618–629.
[118] X. Liu, D. F. Wong, Y. Liu, L. S. Chao, T. Xiao, J. Zhu, Shared-private bilingual word embeddings for neural machine translation, in: Proceedings of ACL, 2019, pp. 3613–3622.
[119] K. Chen, R. Wang, M. Utiyama, E. Sumita, Content word aware neural machine translation, in: Proceedings of ACL, 2020, pp. 358–364.
[120] M. Weller-Di Marco, A. Fraser, Modeling word formation in english–german neural machine translation, in: Proceedings of ACL, 2020, pp. 4227–4232.
[121] A. Eriguchi, K. Hashimoto, Y. Tsuruoka, Tree-to-sequence attentional neural machine translation, in: Proceedings of ACL, 2016, pp. 823–833.
[122] R. Sennrich, B. Haddow, Linguistic input features improve neural machine translation, in: Proceedings of WMT, 2016, pp. 83–91.
[123] J. Hao, X. Wang, S. Shi, J. Zhang, Z. Tu, Multi-granularity self-attention for neural machine translation, in: Proceedings of EMNLP-IJCNLP, 2019, pp. 886–896.
[124] E. Bugliarello, N. Okazaki, Enhancing machine translation with dependency-aware self-attention, in: Proceedings of ACL, 2020, pp. 1618–1627.
[125] A. Eriguchi, Y. Tsuruoka, K. Cho, Learning to parse and translate improves neural machine translation, in: Proceedings of ACL, 2017, pp. 72–78.
[126] L. H. Baniata, S. Park, S.-B. Park, A multitask-based neural machine translation model with part-of-speech tags integration for arabic dialects,

Applied Sciences 8 (2018) 2502.
[127] J. Gū, H. S. Shavarani, A. Sarkar, Top-down tree structured decoding with syntactic connections for neural machine translation and parsing, in: Proceedings of EMNLP, 2018, pp. 401–413.
[128] X. Wang, H. Pham, P. Yin, G. Neubig, A tree-based decoder for neural machine translation, in: Proceedings of EMNLP, 2018, pp. 4772–4777.
[129] S. Wu, D. Zhang, N. Yang, M. Li, M. Zhou, Sequence-to-dependency neural machine translation, in: Proceedings of ACL, 2017, pp. 698–707.
[130] R. Aharoni, Y. Goldberg, Towards string-to-tree neural machine translation, in: Proceedings of ACL, 2017, pp. 132–140.
[131] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, K. Sima'an, Graph convolutional encoders for syntax-aware neural machine translation, in: Proceedings of EMNLP, 2017, pp. 1957–1967.
[132] X. Li, L. Liu, Z. Tu, S. Shi, M. Meng, Target foresight based attention for neural machine translation, in: Proceedings of NAACL-HLT, 2018, pp. 1380–1390.
[133] X. Yang, Y. Liu, D. Xie, X. Wang, N. Balasubramanian, Latent part-of-speech sequences for neural machine translation, in: Proceedings of EMNLP-IJCNLP, 2019, pp. 780–790.
[134] J. Yang, S. Ma, D. Zhang, Z. Li, M. Zhou, Improving neural machine translation with soft template prediction, in: Proceedings of WMT, 2020, pp. 5979–5989.
[135] Y. Belinkov, Y. Bisk, Synthetic and natural noise both break neural machine translation, in: Proceedings of ICLR, 2018.
[136] Y. Cheng, L. Jiang, W. Macherey, Robust neural machine translation with doubly adversarial inputs, in: Proceedings of ACL, 2019, pp. 4324–4333.
[137] Y. Ding, Y. Liu, H. Luan, M. Sun, Visualizing and understanding neural machine translation, in: Proceedings of ACL, 2017, pp. 1150–1159.
[138] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, W. Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, PloS one 10 (2015) e0130140.
[139] A. Bau, Y. Belinkov, H. Sajjad, N. Durrani, F. Dalvi, J. Glass, Identifying and controlling important neurons in neural machine translation, in: Proceedings of ICLR, 2019.
[140] H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, A. M. Rush, Seq2seq-vis: A visual debugging tool for sequence-to-sequence models, IEEE transactions on visualization and computer graphics 25 (2019) 353–363.
[141] S. He, Z. Tu, X. Wang, L. Wang, M. Lyu, S. Shi, Towards understanding neural machine translation with word importance, in: Proceedings of EMNLP-IJCNLP, 2019, pp. 953–962.
[142] A. Raganato, J. Tiedemann, An analysis of encoder representations in transformer-based machine translation, in: Proceedings of EMNLP Workshop, 2018, pp. 287–297.
[143] E. Voita, R. Sennrich, I. Titov, The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives, in: Proceedings of EMNLP-IJCNLP, 2019, pp. 4396–4406.
[144] F. Stahlberg, D. Saunders, B. Byrne, An operation sequence model for explainable neural machine translation, in: Proceedings of EMNLP Workshop, 2018, pp. 175–186.
[145] C. Yun, S. Bhojanapalli, A. S. Rawat, S. Reddi, S. Kumar, Are transformers universal approximators of sequence-to-sequence functions?, in: Proceedings of ICLR, 2020.
[146] Z. Zhao, D. Dua, S. Singh, Generating natural adversarial examples, in: Proceedings of ICLR, 2018.
[147] M. T. Ribeiro, S. Singh, C. Guestrin, Semantically equivalent adversarial rules for debugging nlp models, in: Proceedings of ACL, 2018, pp. 856–865.
[148] Y. Cheng, Z. Tu, F. Meng, J. Zhai, Y. Liu, Towards robust neural machine translation, in: Proceedings of ACL, 2018, pp. 1756–1766.
[149] J. Ebrahimi, D. Lowd, D. Dou, On adversarial examples for character-level neural machine translation, in: Proceedings of COLING, 2018, pp. 653–663.
[150] W. Zou, S. Huang, J. Xie, X. Dai, J. Chen, A reinforced generation of adversarial examples for neural machine translation, in: Proceedings of ACL, 2020, pp. 3486–3497.
[151] Y. Cheng, L. Jiang, W. Macherey, J. Eisenstein, AdvAug: Robust adversarial augmentation for neural machine translation, in: Proceedings of ACL, 2020, pp. 5961–5970.
[152] P. Michel, G. Neubig, Mtnt: A testbed for machine translation of noisy text, in: Proceedings of EMNLP, 2018, pp. 543–553.
[153] J. Tiedemann, Opus–parallel corpora for everyone, Baltic Journal of Modern Computing (2016) 384.
[154] B. Zhang, P. Williams, I. Titov, R. Sennrich, Improving massively multilingual neural machine translation and zero-shot translation, arXiv preprint arXiv:2004.11867 (2020).
[155] G. Wenzek, M.-A. Lachaux, A. Conneau, V. Chaudhary, F. Guzmán, A. Joulin, É. Grave, Ccnet: Extracting high quality monolingual datasets from web crawl data, in: Proceedings of The 12th Language Resources and Evaluation Conference, 2020, pp. 4003–4012.
[156] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 2016, pp. 265–283.
[157] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in NeurIPS, 2019, pp. 8026–8037.
[158] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, J. Uszkoreit, Tensor2Tensor for neural machine translation, in: Proceedings of AMTA, 2018, pp. 193–199.
[159] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, M. Auli, fairseq: A fast, extensible toolkit for sequence modeling, in: Proceedings of NAACL-HLT (Demonstrations), 2019, pp. 48–53.
[160] M. Luong, E. Brevdo, R. Zhao, Neural machine translation (seq2seq) tutorial, https://github.com/tensorflow/nmt (2017).
[161] F. Hieber, T. Domhan, M. Denkowski, D. Vilar, A. Sokolov, A. Clifton, M. Post, Sockeye: A toolkit for neural machine translation, arXiv preprint arXiv:1712.05690 (2017).
[162] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, in: Proceedings of NeurIPS, Workshop, 2016.
[163] M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. F. Aji, N. Bogoychev, A. F. T. Martins, A. Birch, Marian: Fast neural machine translation in C++, in: Proceedings of ACL, System Demonstrations, 2018, pp. 116–121.
[164] Z. Tan, J. Zhang, X. Huang, G. Chen, S. Wang, M. Sun, H. Luan, Y. Liu, THUMT: An open-source toolkit for neural machine translation, in: Proceedings of AMTA, 2020, pp. 116–122.
[165] A. Peris, F. Casacuberta, Nmt-keras: a very flexible toolkit with a focus on interactive nmt and online learning, The Prague Bulletin of Mathematical Linguistics 111 (2018) 113–124.
[166] K. Papineni, S. Roukos, T. Ward, W. Zhu, Bleu: A method for automatic evaluation of machine translation, in: Proceedings of ACL, 2002.
[167] M. Post, A call for clarity in reporting bleu scores, arXiv preprint arXiv:1804.08771 (2018).
[168] G. Neubig, Z.-Y. Dou, J. Hu, P. Michel, D. Pruthi, X. Wang, compare-mt: A tool for holistic comparison of language generation systems, in: Proceedings of NAACL-HLT (Demonstrations), 2019, pp. 35–41.
[169] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al., Moses: Open source toolkit for statistical machine translation, in: Proceedings of ACL on interactive poster and demonstration sessions, 2007, pp. 177–180.

Relation Extraction

Luke Zettlemoyer

CSE 517

Winter 2013

[with slides adapted from many people, including Bill MacCartney, Dan Jurafsky, Rion Snow, Jim Martin, Chris Manning, William Cohen, and others]

Goal: “machine reading”

- Acquire structured knowledge from unstructured text

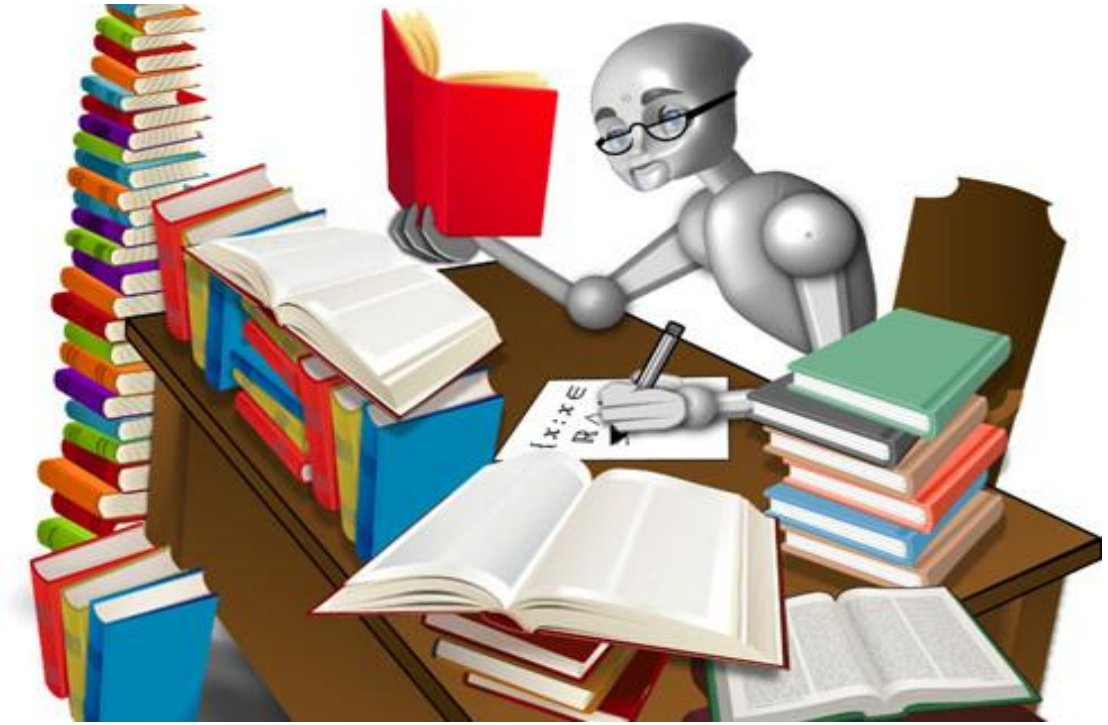


illustration from DARPA

Information extraction

- IE = extracting information from text
- Sometimes called *text analytics* commercially
- Extract entities
  - People, organizations, locations, times, dates, prices, ...
  - Or sometimes: genes, proteins, diseases, medicines, ...
- Extract the relations between entities
  - Located in, employed by, part of, married to, ...
- Figure out the larger events that are taking place

Machine-readable summaries

IE

Subject	Relation	Object
p53	**is_a**	protein
Bax	**is_a**	protein
p53	has_function	apoptosis
Bax	has_function	induction
apoptosis	involved_in	cell_death
Bax	is_in	mitochondrial outer membrane
Bax	is_in	cytoplasm
apoptosis	related_to	caspase activation
...	...	...

textual abstract:
summary for human

structured knowledge extraction:
summary for machine

More applications of IE

- Building & extending knowledge bases and ontologies
- Scholarly literature databases: Google Scholar, CiteSeerX
- People directories: Rapleaf, Spoke, Naymz
- Shopping engines & product search
- Bioinformatics: clinical outcomes, gene interactions, …
- Patent analysis
- Stock analysis: deals, acquisitions, earnings, hirings & firings
- SEC filings
- Intelligence analysis for business & government

Named Entity Recognition (NER)

The task:

1. find names in text
2. classify them by type, usually {ORG, PER, LOC, MISC}

```
The [European Commission ORG] said on Thursday it
disagreed with [German MISC] advice.
Only [France LOC] and [Britain LOC] backed
[Fischler PER] 's proposal .

"What we have to be extremely careful of is how
other countries are going to take [Germany LOC]
's lead", [Welsh National Farmers ' Union ORG]
( [NFU ORG] ) chairman [John Lloyd Jones PER]
said on [BBC ORG] radio .
```

Named Entity Recognition (NER)

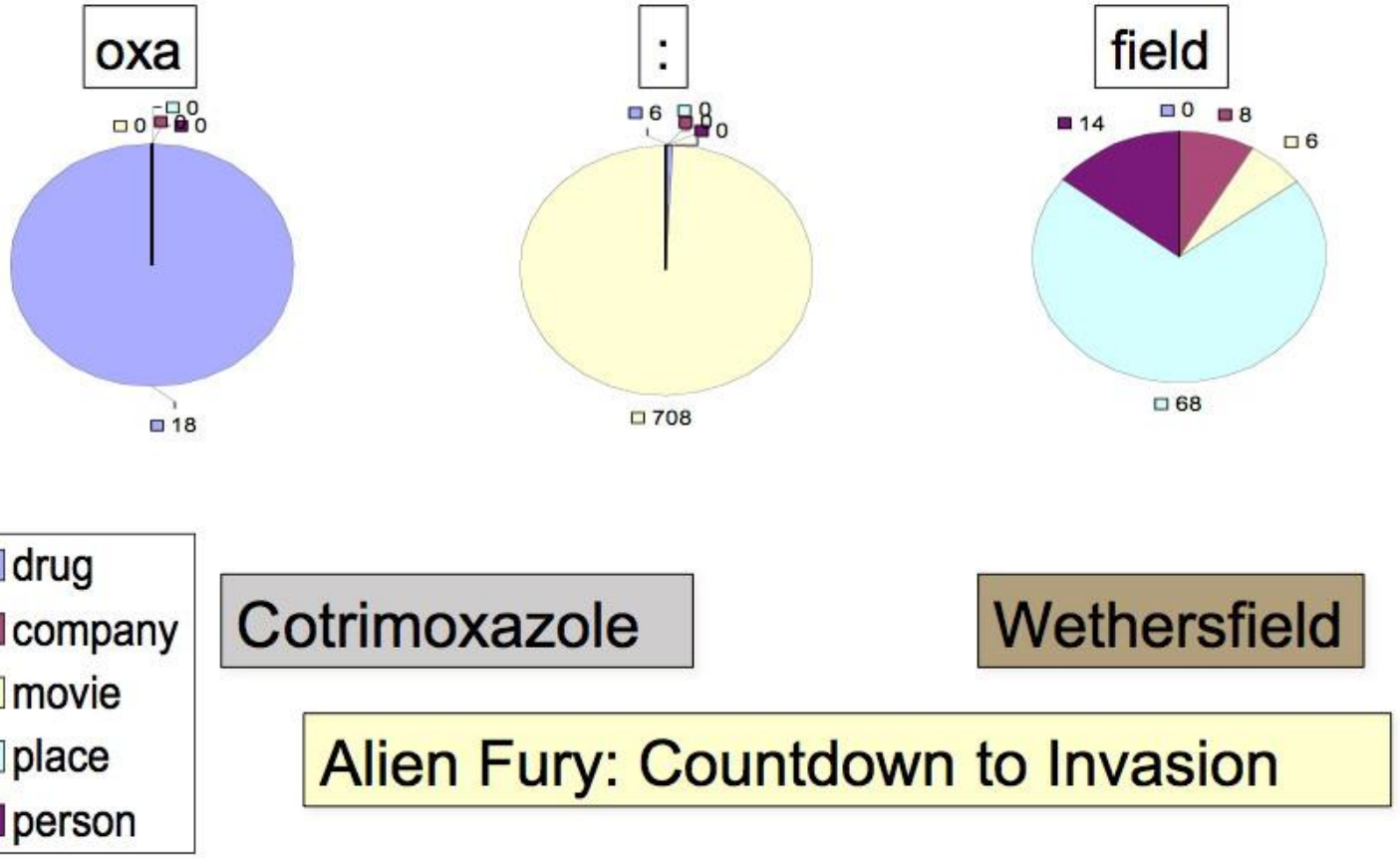
- It's a tagging task, similar to part-of speech (POS) tagging
- So, systems use sequence classifiers: HMMs, MEMMs, CRFs
- Features usually include words, POS tags, word shapes, orthographic features, gazetteers, etc.
- Accuracies of >90% are typical — but depends on genre!
- NER is commonly thought of as a "solved problem"
- A building block technology for relation extraction
- E.g., http://nlp.stanford.edu/software/CRF-NER.shtml

Orthographic features for NER

oxa : field

slide adapted from Chris Manning

Orthographic features for NER



Relation extraction example

CHICAGO (AP) — Citing high fuel prices, United Airlines said Friday it has increased fares by $6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said. United, a unit of UAL, said the increase took effect Thursday night and applies to most routes where it competes against discount carriers, such as Chicago to Dallas and Atlanta and Denver to San Francisco, Los Angeles and New York.

Question: What relations should we extract?

example from Jim Martin

Relation extraction example

CHICAGO (AP) — Citing high fuel prices, United Airlines said Friday it has increased fares by $6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said. United, a unit of UAL, said the increase took effect Thursday night and applies to most routes where it competes against discount carriers, such as Chicago to Dallas and Atlanta and Denver to San Francisco, Los Angeles and New York.

Subject	Relation	Object
American Airlines	subsidiary	AMR
Tim Wagner	employee	American Airlines
United Airlines	subsidiary	UAL

example from Jim Martin

Relation types

For generic news texts ...

Relations		Examples	Types
Affiliations			
	Personal	*married to, mother of*	PER → PER
	Organizational	*spokesman for, president of*	PER → ORG
	Artifactual	*owns, invented, produces*	(PER | ORG) → ART
Geospatial			
	Proximity	*near, on outskirts*	LOC → LOC
	Directional	*southeast of*	LOC → LOC
Part-Of			
	Organizational	*a unit of, parent of*	ORG → ORG
	Political	*annexed, acquired*	GPE → GPE

slide adapted from Jim Martin

Relation types from ACE 2003

ROLE: relates a person to an organization or a geopolitical entity

subtypes: member, owner, affiliate, client, citizen

PART: generalized containment

subtypes: subsidiary, physical part-of, set membership

AT: permanent and transient locations

subtypes: located, based-in, residence

SOCIAL: social relations among persons

subtypes: parent, sibling, spouse, grandparent, associate

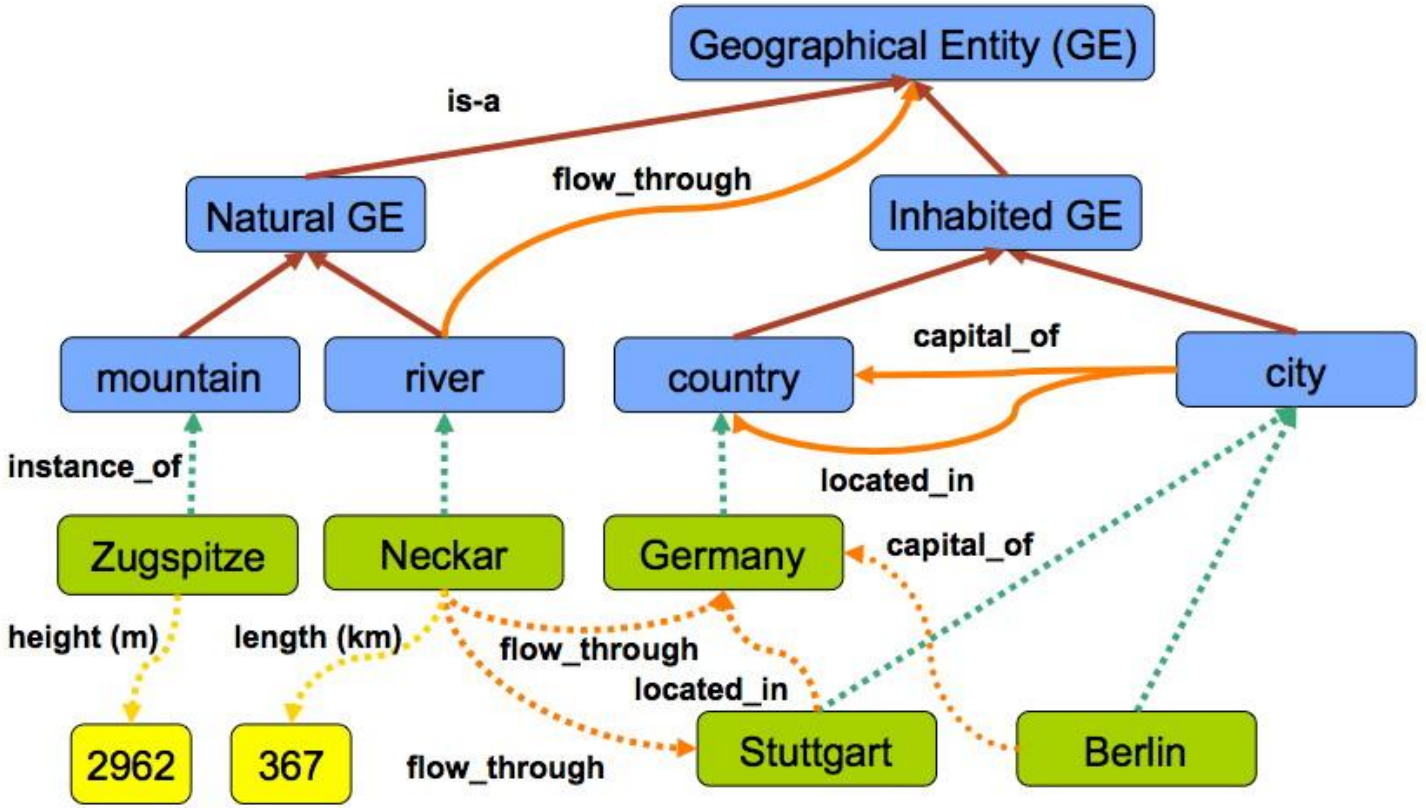
slide adapted from Doug Appelt

Relation types: Freebase

23 Million Entities, thousands of relations

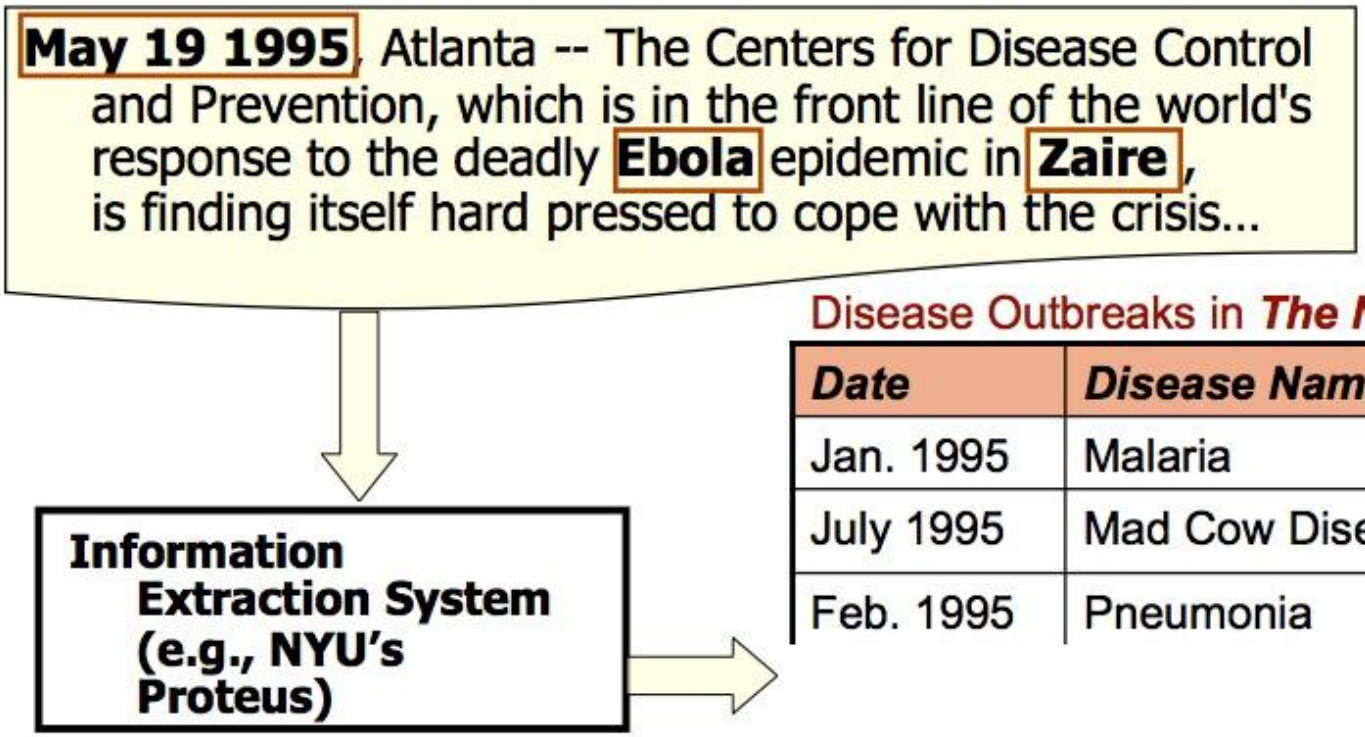
Relation name	Size	Example
/people/person/nationality	281,107	John Dugard, South Africa
/location/location/contains	253,223	Belgium, Nijlen
/people/person/profession	208,888	Dusa McDuff, Mathematician
/people/person/place_of_birth	105,799	Edwin Hubble, Marshfield
/dining/restaurant/cuisine	86,213	MacAyo's Mexican Kitchen, Mexican
/business/business_chain/location	66,529	Apple Inc., Apple Inc., South Park, NC
/biology/organism_classification_rank	42,806	Scorpaeniformes, Order
/film/film/genre	40,658	Where the Sidewalk Ends, Film noir
/film/film/language	31,103	Enter the Phoenix, Cantonese
/biology/organism_higher_classification	30,052	Calopteryx, Calopterygidae
/film/film/country	27,217	Turtle Diary, United States
/film/writer/film	23,856	Irving Shulman, Rebel Without a Cause
/film/director/film	23,539	Michael Mann, Collateral
/film/producer/film	22,079	Diane Eskenazi, Aladdin
/people/deceased_person/place_of_death	18,814	John W. Kern, Asheville
/music/artist/origin	18,619	The Octopus Project, Austin
/people/person/religion	17,582	Joseph Chartrand, Catholicism
/book/author/works_written	17,278	Paul Auster, Travels in the Scriptorium
/soccer/football_position/players	17,244	Midfielder, Chen Tao
/people/deceased_person/cause_of_death	16,709	Richard Daintree, Tuberculosis
/book/book/genre	16,431	Pony Soldiers, Science fiction
/film/film/music	14,070	Stavisky, Stephen Sondheim
/business/company/industry	13,805	ATS Medical, Health care

Relation types: geographical



slide adapted from Paul Buitelaar

More relations: disease outbreaks



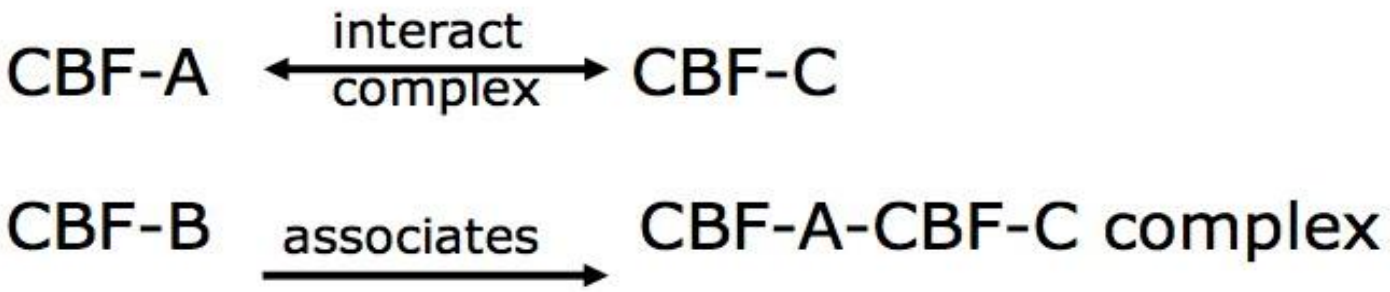
Disease Outbreaks in ***The New York Times***

Date	*Disease Name*	*Location*
Jan. 1995	Malaria	Ethiopia
July 1995	Mad Cow Disease	U.K.
Feb. 1995	Pneumonia	U.S.

slide adapted from Eugene Agichtein

More relations: protein interactions

„We show that CBF-A and CBF-C interact with each other to form a CBF-A-CBF-C complex and that CBF-B does not interact with CBF-A or CBF-C individually but that it associates with the CBF-A-CBF-C complex."



slide adapted from Rosario & Hearst

Relations between word senses

- NLP applications need word meaning!
  - Question answering
  - Conversational agents
  - Summarization
- One key meaning component: word relations
  - Hyponymy: San Francisco is an instance of a city
  - Antonymy: acidic is the opposite of basic
  - Meronymy: an alternator is a part of a car

WordNet is incomplete

Ontological relations are missing for many words:

In WordNet 3.1	Not in WordNet 3.1
insulin progesterone	leptin pregnenolone
combustibility navigability	affordability reusability
HTML	XML
Google, Yahoo	Microsoft, IBM

Esp. for specific domains: restaurants, auto parts, finance

Relation extraction: 5 easy methods

1. Hand-built patterns
2. Bootstrapping methods
3. Supervised methods
4. Distant supervision
5. Unsupervised methods

Relation extraction: 5 easy methods

1. Hand-built patterns
2. Bootstrapping methods
3. Supervised methods
4. Distant supervision
5. Unsupervised methods

A hand-built extraction rule

```
;;; For <company> appoints <person> <position>

(defpattern appoint
    "np-sem(C-company)?  rn?  sa?  vg(C-appoint) np-sem(C-person) ´,´?
     to-be?  np(C-position) to-succeed?:
     company-at=1.attributes, sa=3.span, lv=4.span, person-at=5.attributes
     position-at=8.attributes |
...
```

```
(defun when-appoint (phrase-type)
     (let ((person-at (binding ´person-at))
           (company-entity (entity-bound ´company-at))
           (person-entity (essential-entity-bound ´person-at ´C-person))
           (position-entity (entity-bound ´position-at))
           (predecessor-entity (entity-bound ´predecessor-at))
           new-event)
      (not-an-antecedent position-entity)
      ;; if no company is specified for position, use agent
...
```

NYU Proteus system (1997)

Patterns for learning hyponyms

- Intuition from Hearst (1992)

  Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.

- What does *Gelidium* mean?

- How do you know?

Patterns for learning hyponyms

- Intuition from Hearst (1992)

  Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.

- What does *Gelidium* mean?

- How do you know?

Hearst's lexico-syntactic patterns

Y such as X ((, X)* (, and/or) X)

such Y as X…

X… or other Y

X… and other Y

Y including X…

Y, especially X…

Hearst, 1992. Automatic Acquisition of Hyponyms.

Examples of the Hearst patterns

Hearst pattern	Example occurrences
X and other Y	...temples, treasuries, and other important civic buildings.
X or other Y	bruises, wounds, broken bones or other injuries...
Y such as X	The bow lute, such as the Bambara ndang...
such Y as X	...such authors as Herrick, Goldsmith, and Shakespeare.
Y including X	...common-law countries, including Canada and England...
Y, especially X	European countries, especially France, England, and Spain...

Patterns for learning meronyms

- Berland & Charniak (1999) tried it
- Selected initial patterns by finding all sentences in a corpus containing *basement* and *building*

whole NN[-PL] 's POS part NN[-PL]	... building's basement ...
part NN[-PL] of PREP {the|a} DET mods [JJ|NN]* whole NN	... basement of a building ...
part NN in PREP {the|a} DET mods [JJ|NN]* whole NN	... basement in a building ...
parts NN-PL of PREP wholes NN-PL	... basements of buildings ...
parts NN-PL in PREP wholes NN-PL	... basements in buildings ...

- Then, for each pattern:
  1. found occurrences of the pattern
  2. filtered those ending with *-ing*, *-ness*, *-ity*
  3. applied a likelihood metric — poorly explained
- Only the first two patterns gave decent (though not great!) results

Problems with hand-built patterns

- Requires hand-building patterns for each relation!
  - hard to write; hard to maintain
  - there are zillions of them
  - domain-dependent
- Don’t want to do this for all possible relations!
- Plus, we’d like better accuracy
  - Hearst: 66% accuracy on hyponym extraction
  - Berland & Charniak: 55% accuracy on meronyms

Relation extraction: 5 easy methods

1. Hand-built patterns
2. Bootstrapping methods
3. Supervised methods
4. Distant supervision
5. Unsupervised methods

Bootstrapping approaches

- If you don't have enough annotated text to train on …
- But you do have:
  - some seed instances of the relation
  - (or some patterns that work pretty well)
  - and lots & lots of unannotated text (e.g., the web)
- … can you use those seeds to do something useful?
- Bootstrapping can be considered *semi-supervised*

Bootstrapping example

- Target relation: *burial place*
- Seed tuple: [*Mark Twain*, *Elmira*]
- Grep/Google for “Mark Twain” and “Elmira”

  “Mark Twain is buried in Elmira, NY.”

  → X is buried in Y

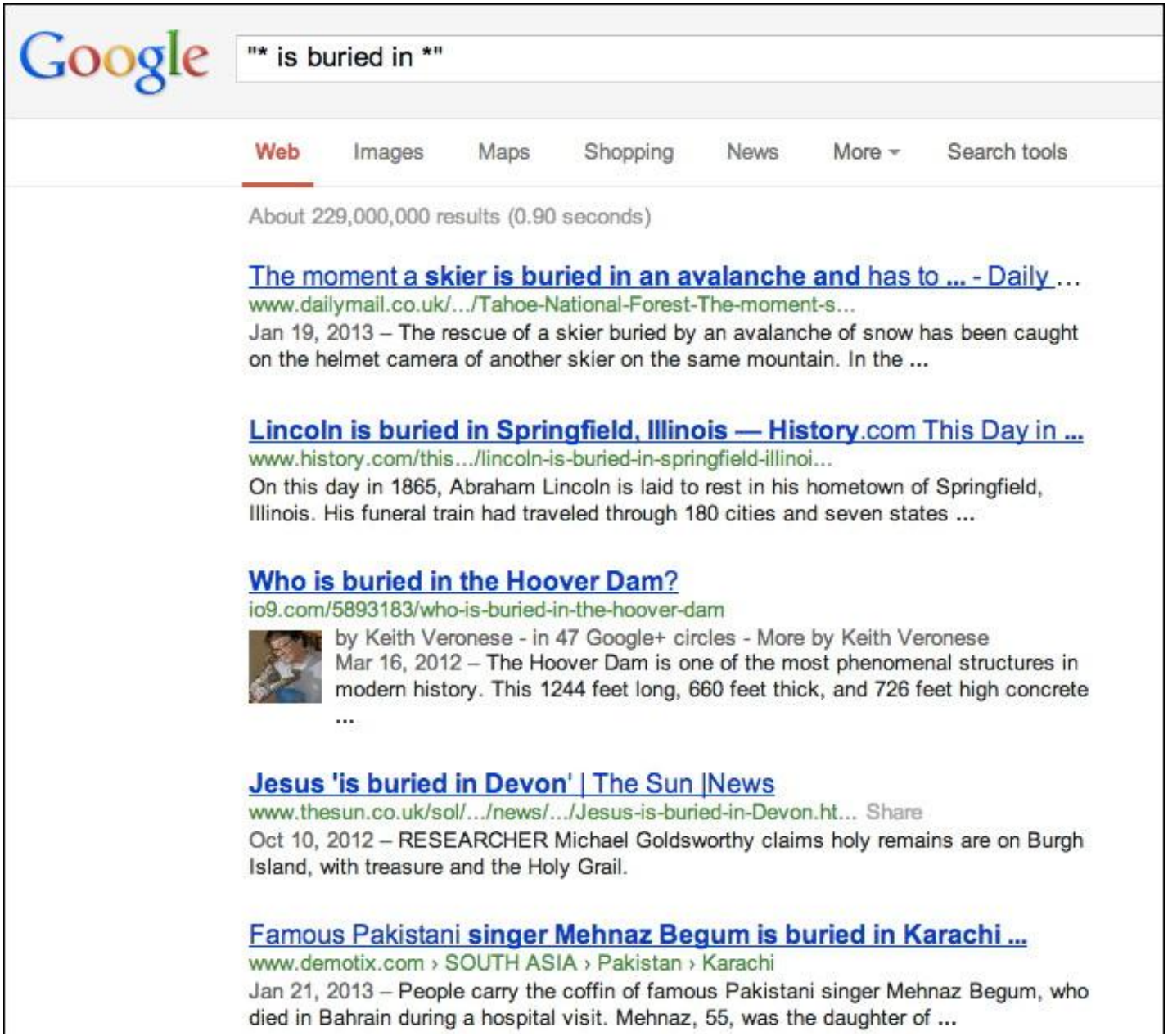
  “The grave of Mark Twain is in Elmira”

  → The grave of X is in Y

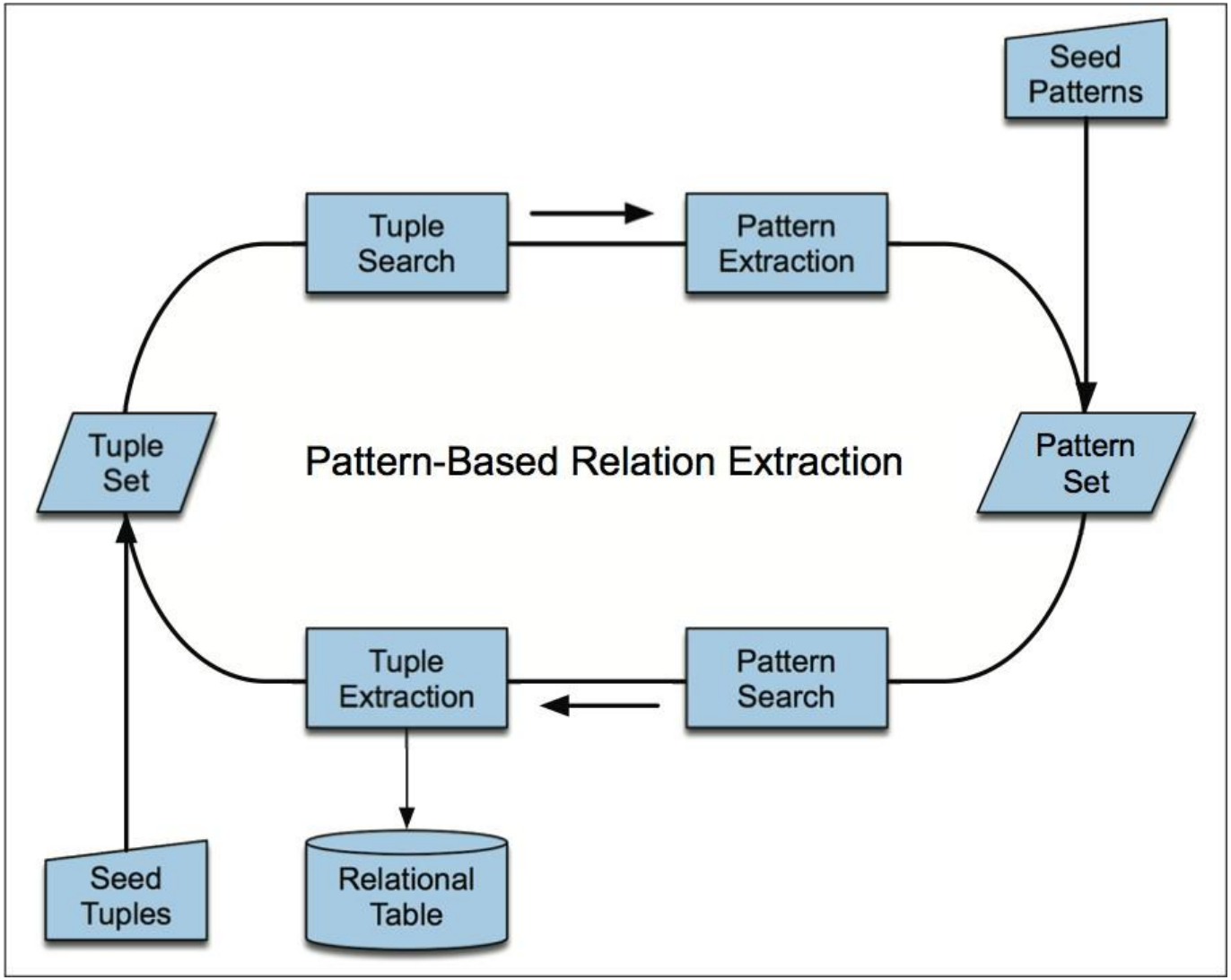
  “Elmira is Mark Twain’s final resting place”

  → Y is X’s final resting place
- Use those patterns to search for new tuples

Bootstrapping example



Bootstrapping relations



slide adapted from Jim Martin

DIPRE (Brin 1998)

Extract (author, book) pairs

Start with these 5 seeds:

Author	Book
Isaac Asimov	The Robots of Dawn
David Brin	Startide Rising
James Gleick	Chaos: Making a New Science
Charles Dickens	Great Expectations
William Shakespeare	The Comedy of Errors

Learn these patterns:

URL Prefix	Text Pattern
`www.sff.net/locus/c.*`	<LI><B>*title*</B> by *author* (
`dns.city-net.com/~lmann/awards/hugos/1984.html`	*title* by *author* (
`dolphin.upenn.edu/~dcummins/texts/sf-award.htm`	*author* || *title* || (

Iterate: use patterns to get more instances & patterns…

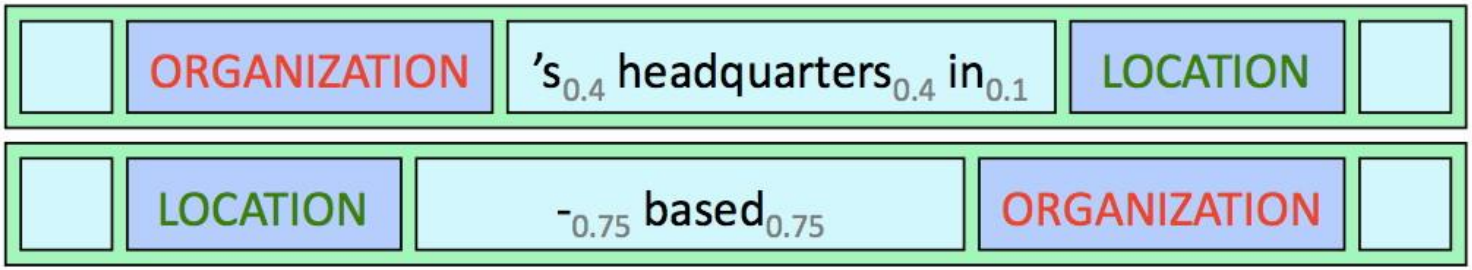
Results: after three iterations of bootstrapping loop, extracted 15,000 author-book pairs with 95% accuracy.

Snowball (Agichtein & Gravano 2000)

New ideas:

- require that X and Y be named entities
- add heuristics to score extractions, select best ones

Organization	Location of Headquarters
Microsoft	Redmond
Exxon	Irving
IBM	Armonk
Boeing	Seattle
Intel	Santa Clara



Snowball Results!

Conf	*middle*	*right*
1	<based, 0.53> <in, 0.53>	<, , 0.01>
0.69	<', 0.42> <s, 0.42> < headquarters, 0.42> <in, 0.12>	
0.61	<(, 0.93>	<), 0.12>

Table 2: Actual patterns discovered by *Snowball*. **(For each pattern the** *left* **vector is empty,** *tag1* **=** *ORGANIZATION*, **and** *tag2* **=** *LOCATION*.**)**

			Type of Error			
	Correct	*Incorrect*	*Location*	*Organization*	*Relationship*	P_{Ideal}
DIPRE	74	26	3	18	5	90%
Snowball (all tuples)	52	48	6	41	1	88%
Snowball ($\tau_t = 0.8$)	93	7	3	4	0	96%
Baseline	25	75	8	62	5	66%

5: Manually computed precision estimate, derived from a random sample of 100 tuples from each e

Bootstrapping problems

- Requires that we have seeds for each relation
  - Sensitive to original set of seeds
- Big problem of semantic drift at each iteration
- Precision tends to be not that high
- Generally have lots of parameters to be tuned
- No probabilistic interpretation
  - Hard to know how confident to be in each result

Relation extraction: 5 easy methods

1. Hand-built patterns
2. Bootstrapping methods
3. Supervised methods
4. Distant supervision
5. Unsupervised methods

Supervised relation extraction

The supervised approach requires:

- Defining an inventory of output labels
  - Relation detection: true/false
  - Relation classification: located-in, employee-of, inventor-of, …
- Collecting labeled training data: MUC, ACE, …
- Defining a feature representation: words, entity types, …
- Choosing a classifier: Naïve Bayes, MaxEnt, SVM, …
- Evaluating the results

ACE 2008: relations

Type	Subtype
ART (artifact)	User-Owner-Inventor-Manufacturer
GEN-AFF (General affiliation)	Citizen-Resident-Religion-Ethnicity, Org-Location
METONYMY*	*None*
ORG-AFF (Org-affiliation)	Employment, Founder, Ownership, Student-Alum, Sports-Affiliation, Investor-Shareholder, Membership
PART-WHOLE (part-to-whole)	Artifact, Geographical, Subsidiary
PER-SOC* (person-social)	Business, Family, Lasting-Personal
PHYS* (physical)	Located, Near

ACE 2008: data

Source	Training epoch	Approximate size
English Resources		
Broadcast News	3/03 – 6/03	55,000 words
Broadcast Conversations	3/03 – 6/03	40,000 words
Newswire	3/03 – 6/03	50,000 words
Weblog	11/04 – 2/05	40,000 words
Usenet	11/04 – 2/05	40,000 words
Conversational Telephone Speech	11/04-12/04 (differentiated by topic vs. eval)	40,000 words
Arabic Resources		
Broadcast News	10/00 – 12/00	30,000+ words
Newswire	10/00 – 12/00	55,000+ words
Weblog	11/04 – 2/05	20,000+ words

Features

- Lightweight features — require little pre-processing
  - Bags of words & bigrams between, before, and after the entities
  - Stemmed versions of the same
  - The types of the entities
  - The distance (number of words) between the entities
- Medium-weight features — require base phrase chunking
  - Base-phrase chunk paths
  - Bags of chunk heads
- Heavyweight features — require full syntactic parsing
  - Dependency-tree paths
  - Constituent-tree paths
  - Tree distance between the entities
  - Presence of particular constructions in a constituent structure

Let's take a closer look at features used in Zhou et al. 2005

Features: words

American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said.

Bag-of-words features

WM1 = {American, Airlines}, WM2 = {Tim, Wagner}

Head-word features

HM1 = Airlines, HM2 = Wagner, HM12 = Airlines+Wagner

Words in between

WBNULL = false, WBFL = NULL, WBF = a, WBL = spokesman,
WBO = {unit, of, AMR, immediately, matched, the, move}

Words before and after

BM1F = NULL, BM1L = NULL, AM2F = said, AM2L = NULL

Word features yield good precision (69%), but poor recall (24%)

Features: NE type & mention level

American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said.

Named entity types (ORG, LOC, PER, etc.)
ET1 = ORG, ET2 = PER, ET12 = ORG-PER

Mention levels (NAME, NOMINAL, or PRONOUN)
ML1 = NAME, ML2 = NAME, ML12 = NAME+NAME

Named entity type features help recall a lot (+8%)
Mention level features have little impact

Features: overlap

American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said.

Number of mentions and words in between

#MB = 1, #WB = 9

Does one mention include in the other?

M1>M2 = false, M1<M2 = false

Conjunctive features

ET12+M1>M2 = ORG-PER+false

ET12+M1<M2 = ORG-PER+false

HM12+M1>M2 = Airlines+Wagner+false

HM12+M1<M2 = Airlines+Wagner+false

These features hurt precision a lot (-10%), but also help recall a lot (+8%)

Features: base phrase chunking

American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said.

Parse using the Stanford Parser, then apply Sabine Buchholz's chunklink.pl:

```
 0 B-NP    NNP   American        NOFUNC      Airlines      1 B-S/B-S/B-NP/B-NP
 1 I-NP    NNPS  Airlines        NP          matched       9 I-S/I-S/I-NP/I-NP
 2 O       COMMA COMMA           NOFUNC      Airlines      1 I-S/I-S/I-NP
 3 B-NP    DT    a               NOFUNC      unit          4 I-S/I-S/I-NP/B-NP/B-NP
 4 I-NP    NN    unit            NP          Airlines      1 I-S/I-S/I-NP/I-NP/I-NP
 5 B-PP    IN    of              PP          unit          4 I-S/I-S/I-NP/I-NP/B-PP
 6 B-NP    NNP   AMR             NP          of            5 I-S/I-S/I-NP/I-NP/I-PP/B-NP
 7 O       COMMA COMMA           NOFUNC      Airlines      1 I-S/I-S/I-NP
 8 B-ADVP  RB    immediately     ADVP        matched       9 I-S/I-S/B-ADVP
 9 B-VP    VBD   matched         VP/S        matched       9 I-S/I-S/B-VP
10 B-NP    DT    the             NOFUNC      move         11 I-S/I-S/I-VP/B-NP
11 I-NP    NN    move            NP          matched       9 I-S/I-S/I-VP/I-NP
12 O       COMMA COMMA           NOFUNC      matched       9 I-S
13 B-NP    NN    spokesman       NOFUNC      Wagner       15 I-S/B-NP
14 I-NP    NNP   Tim             NOFUNC      Wagner       15 I-S/I-NP
15 I-NP    NNP   Wagner          NP          matched       9 I-S/I-NP
16 B-VP    VBD   said            VP          matched       9 I-S/B-VP
17 O       .     .               NOFUNC      matched       9 I-S
```

[$_{NP}$ American Airlines], [$_{NP}$ a unit] [$_{PP}$ of] [$_{NP}$ AMR], [$_{ADVP}$ immediately] [$_{VP}$ matched] [$_{NP}$ the move], [$_{NP}$ spokesman Tim Wagner] [$_{VP}$ said].

Features: base phrase chunking

[$_{NP}$ American Airlines], [$_{NP}$ a unit] [$_{PP}$ of] [$_{NP}$ AMR], [$_{ADVP}$ immediately] [$_{VP}$ matched] [$_{NP}$ the move], [$_{NP}$ spokesman Tim Wagner] [$_{VP}$ said].

Phrase heads before and after

CPHBM1F = NULL, CPHBM1L = NULL, CPHAM2F = said, CPHAM2L = NULL

Phrase heads in between

CPHBNULL = false, CPHBFL = NULL, CPHBF = unit, CPHBL = move

CPHBO = {of, AMR, immediately, matched}

Phrase label paths

CPP = [NP, PP, NP, ADVP, VP, NP]

CPPH = NULL

These features increased both precision & recall by 4-6%

Features: syntactic features

Features of mention dependencies

ET1DW1 = ORG:Airlines

H1DW1 = matched:Airlines

ET2DW2 = PER:Wagner

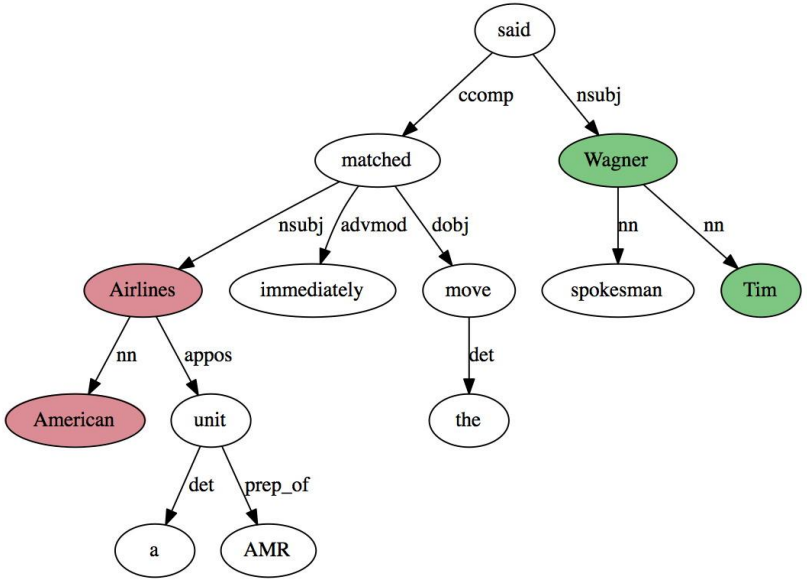
H2DW2 = said:Wagner

Features describing entity types and dependency tree

ET12SameNP = ORG-PER-false

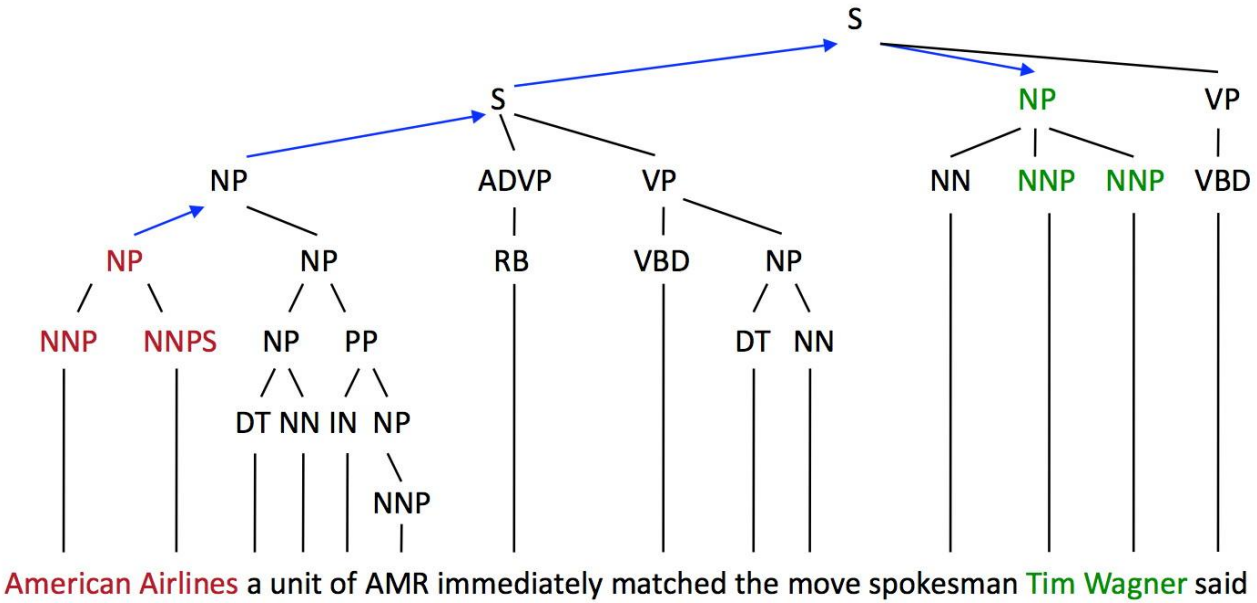
ET12SamePP = ORG-PER-false

ET12SameVP = ORG-PER-false



These features had disappointingly little impact!

Features: syntactic features



Phrase label paths

PTP = [NP, S, NP]

PTPH = [NP:Airlines, S:matched, NP:Wagner]

These features had disappointingly little impact!

Relation extraction classifiers

Now use any (multiclass) classifier you like:

- SVM
- MaxEnt (aka multiclass logistic regression)
- Naïve Bayes
- etc.

[Zhou et al. 2005 used a one-vs-many SVM]

Zhou et al. 2005 results

Features	P	R	F
Words	69.2	23.7	35.3
+Entity Type	67.1	32.1	43.4
+Mention Level	67.1	33.0	44.2
+Overlap	57.4	40.9	47.8
+Chunking	61.5	46.5	53.0
+Dependency Tree	62.1	47.2	53.6
+Parse Tree	62.3	47.6	54.0
+Semantic Resources	63.1	49.5	55.5

Table 2: Contribution of different features over 43 relation subtypes in the test data

Zhou et al. 2005 results

Type	Subtype	#Testing Instances	#Correct	#Error	P	R	F
AT		**392**	**224**	**105**	**68.1**	**57.1**	**62.1**
	Based-In	85	39	10	79.6	45.9	58.2
	Located	241	132	120	52.4	54.8	53.5
	Residence	66	19	9	67.9	28.8	40.4
NEAR		**35**	**8**	**1**	**88.9**	**22.9**	**36.4**
	Relative-Location	35	8	1	88.9	22.9	36.4
PART		**164**	**106**	**39**	**73.1**	**64.6**	**68.6**
	Part-Of	136	76	32	70.4	55.9	62.3
	Subsidiary	27	14	23	37.8	51.9	43.8
ROLE		**699**	**443**	**82**	**84.4**	**63.4**	**72.4**
	Citizen-Of	36	25	8	75.8	69.4	72.6
	General-Staff	201	108	46	71.1	53.7	62.3
	Management	165	106	72	59.6	64.2	61.8
	Member	224	104	36	74.3	46.4	57.1
SOCIAL		**95**	**60**	**21**	**74.1**	**63.2**	**68.5**
	Other-Professional	29	16	32	33.3	55.2	41.6
	Parent	25	17	0	100	68.0	81.0

Table 4: Performance of different relation types and major subtypes in the test data

Supervised RE: summary

- Supervised approach can achieve high accuracy
  - At least, for *some* relations
  - If we have lots of hand-labeled training data
- But has significant limitations!
  - Labeling 5,000 relations (+ named entities) is expensive
  - Doesn't generalize to different relations
- Next: beyond supervised relation extraction
  - Distantly supervised relation extraction
  - Unsupervised relation extraction