**14**

**Machine Learning Architecture**

In the previous chapter, you learned about ingesting and processing big data and getting insights to understand your business. In the traditional way of running a business, the organization's decision-maker looks at past data and uses their experience to plot the future course of company direction. It's not just about setting up the business vision but also improving the end-user experience by predicting their needs and delighting them or automating day-to-day decision-making activities such as loan approval.

However, with the sheer amount of data availability, now, it's become difficult for the human brain to process all data and predict the future. That's where **machine learning** (**ML**) helps us predict future courses of action by looking at a large amount of historical data. Most enterprises are either investing in ML today or planning to do so. It is fast becoming the technology that helps companies differentiate themselves—by creating new products, services, and business models to innovate and gain a competitive advantage.

ML is great for solving business problems because of the countless use cases possible in different lines of business across a company and the high degree of impact these use cases can make. Commercial and government industries can benefit from deploying ML tools that help them achieve better outcomes in less time. AI/ML is a great way to solve problems across different lines of business, for example, build a new level of customer service with call center intelligence, or help marketing teams deliver on their personalization objectives by using a ML-based personalized marketing campaign.

In this chapter, you will learn about the following topics to handle and manage your ML needs:

* What is machine learning?
* Data science and machine learning
* ML model overfitting versus underfitting
* Supervised and unsupervised ML model
* Machine learning architecture
* MLOps
* Deep learning
* Natural language processing
* Design principles for ML architecture

By the end of this chapter, you will know how to design ML architecture. You will learn about the various ML models and ML workflow. You will understand the process of creating an ML model pipeline through feature engineering, model training, inference, and model evaluation.

**What is machine learning?**

ML drives better customer experiences, more efficient business operations, and faster, more accurate decision making. With the rise in compute power and the proliferation of data, ML has moved from the periphery to be a core differentiator for businesses and organizations across industries. ML use cases can apply to most businesses, like personalized product and content recommendations, contact center intelligence, virtual identity verification, and intelligent document processing. And there are customized use cases built for a specific industry—like clinical trials in pharma or assembly line quality control in manufacturing.

Let's say your company wants to send marketing offers to potential customers for a new toy launch, and you have been tasked to develop a system to identify who to target for the marketing campaign. Your customer base could be millions of users to which you need to apply predictive analytics, and ML can help you solve such a complex problem.

ML uses technology to discover trends and patterns and compute mathematical predictive models based on past factual data. ML can help to solve complex problems such as the following:

* When you may not know how to create complex code rules to make a decision; for example, if you want to recognize people's emotions in image and speech, there are no easy ways to code the logic to achieve that.
* When you need to analyze a large amount of data for decision making, the volume of data is too large for a human to do it efficiently. For example, while a human can do it with spam detection, the amount of data makes it impractical to do this quickly.
* When relevant information may only become available dynamically when you need to adapt and personalize user behavior based on individual data; examples are individualized product recommendations or website personalization.
* When there are many tasks with a lot of data available, you cannot track the information fast enough to make a rule-based decision—for example, fraud detection and natural language processing.

Humans handle data prediction based on the results of their analyses and their experience. Using ML, you can train a computer to provide expertise based on available data and get a prediction based on new data. Some of the industry ML use cases are listed below:

* **Predictive maintenance**: Predict if a component will fail before failure based on sensor data. Example applications include predicting failure and **remaining useful life** (**RUL**) of automotive fleets, manufacturing equipment, and IoT sensors. The key value is increased vehicle and equipment up-time and cost savings. This use case is widely used in the automotive and manufacturing industries.
* **Demand forecasting**: Use historical data to forecast key demand metrics faster and make more accurate business decisions around production, pricing, inventory management, and purchasing/re-stocking. The key value is meeting customer demand, reducing inventory carrying costs by reducing surplus inventory, and reducing waste. This use case is used mainly in financial services, manufacturing, retail, and **consumer packaged goods** (**CPG**) industries.
* **Fraud detection**: Automate the detection of potentially fraudulent activity and flag it for review. The key value is reducing costs associated with fraud and maintaining customer trust. This use case is used mainly in financial services and online retail industries.
* **Credit risk prediction**: Explain individual predictions from a credit application to predict whether the credit will be paid back or not (often called a *credit default*). The key value is identifying bias and satisfying regulatory requirements. This use case is used mainly in financial services and online retail industries.
* **Extract & analyze data from documents**: Understand text in written and digital documents and forms, extract information, and use it to classify items and make decisions. This use case is commonly used in the healthcare, financial services, legal, mechanical and electrical, and education industries.
* **Personalized recommendations**: Make customized recommendations based on historical trends. Common in the mechanical and electrical, retail, and education (most likely recommending classes) industries.
* **Churn prediction**: Predict customer likelihood to churn; often used in retail, education, and **software as a service** (**SaaS**).

The main idea behind ML is to make available a training dataset to an ML algorithm and have it predict something from a new dataset, for example, feeding some historical stock market trend data to an ML model and having it predict how the market will fluctuate in the next 6 months to 1 year.

While developing ML solutions, data and code must come together carefully and should evolve in a controlled way, toward the common goal of a robust and scalable ML system; data for training, testing, and inference will change over time, across different sources, and needs to be met with changing code. Without a systematic approach, there can be divergence in how code and data evolve that causes problems in production, gets in the way of smooth deployment, and leads to results that are hard to trace or reproduce. Let's learn how data science goes hand in hand with ML in the next section.

**Working with data science and ML**

ML is all about working with data. The quality of the training data and labels is crucial to the success of an ML model. High-quality data leads to a more accurate ML model and the right prediction. Often in the real world, your data has multiple issues such as missing values, noise, bias, outliers, and so on. Part of data science is the cleaning and preparing of your data to get it ready for ML.

The first thing about data preparation is to understand business problems. Data scientists are often eager to jump into the data directly, start coding, and produce insights. However, without a clear understanding of the business problem, any insights you develop have a high chance of becoming a solution that cannot address the problem at hand. It makes much more sense to start with a straightforward user story and business objectives before getting lost in the data. After building a solid understanding of the business problem, you can begin to narrow down the ML problem categories and determine whether ML will be suitable to solve your particular business problem.

Data science includes data collection, analysis, preprocessing, and feature engineering. Exploring the data provides us with necessary information such as data quality and cleanliness, interesting patterns in the data, and likely paths forward once you start modeling.

Data prep is the first step of building an ML model. It is time-consuming and constitutes up to 80% of the time spent in ML development. Data preparation has always been considered tedious and resource-intensive due to the inherent nature of data being "dirty" and not ready for ML in its raw form. "Dirty" data could include missing or erroneous values, outliers, and so on. Feature engineering is often needed to transform the inputs to deliver more accurate and efficient ML models.

Data preparation often needs multiple steps. While most "standalone data preparation tools" provide data transformation and feature engineering and visualization, few tools offer built-in model validation. And all of these data preparation steps are considered separate from ML. What's needed is a framework that provides all these capabilities in one place and is tightly integrated with the rest of the ML pipeline. Therefore, data preparation modules need curation and integration before they are deployed in production.

As shown in the following diagram, data preprocessing and learning to create an ML model are interconnected—your data preparation will heavily influence your model, while the model you choose heavily influences the type of data preparation you will do. Finding the correct balance is highly iterative and is very much an art (or trial and error):

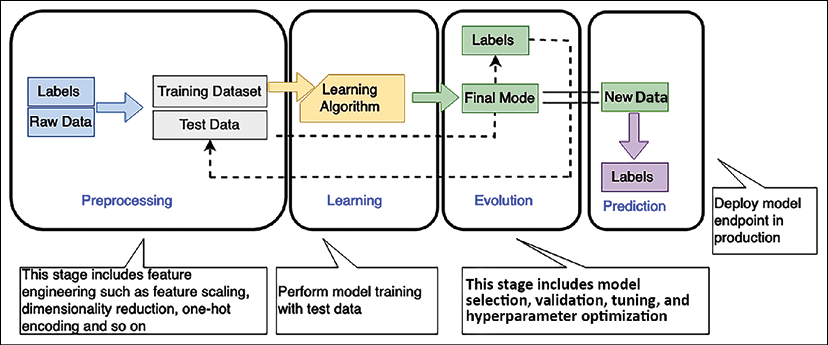


Figure 14.1: ML workflow

As shown in the preceding diagram, the ML workflow includes the following phases:

* **Preprocessing**: In this phase, the data scientist preprocesses the data and divides it into training, validation, and testing datasets. Your ML model gets trained with the training dataset to fit the model and is evaluated using the validation dataset. Once the model is ready, you can test it using a testing dataset. Considering the amount of data and your business case, you need to divide the data into training, testing, and validation sets, perhaps keeping 70% of the data for training, 10% for validation, and 20% for testing. Features are independent attributes of your dataset that may or may not influence the outcome. Feature engineering involves finding the right feature, which can help to achieve model accuracy. The label is your target outcome, which is dependent on feature selection. You can apply dimensionality reduction to choose the right feature, which filters and extracts the most compelling feature for your data.
* **Learning**: You select the appropriate ML algorithm as per the business use case and data in the learning phase. The learning phase is the core of the ML workflow, where you train your ML model on your training dataset. To achieve model accuracy, you need to experiment with various hyperparameters and perform model selection.
* **Evaluation**: Once your ML model gets trained in the learning phase, you want to evaluate the accuracy with a known dataset. You use the validation dataset kept aside during the preprocessing step to assess your model. Required model tuning needs to be performed as per the evaluation result if your model prediction accuracy is not up to the exceptions as determined by validation data.
* **Prediction**: Prediction is also known as inference. In this phase, you deployed your model and started making a prediction. These predictions can be made in real time or in batches.

As per your data input, the ML model often has overfitting or underfitting issues, which you must consider to get the right outcome.

**Evaluating ML models – overfitting versus underfitting**

In overfitting, your model fails to generalize. You will determine an overfitting model that performs well on the training set but poorly on the test set.

This typically indicates that the model is too flexible for the amount of training data, and this flexibility allows it to *memorize* the data, including noise. Overfitting corresponds to high variance, where small changes in the training data result in significant changes to the results.

In underfitting, your model fails to capture essential patterns in the training dataset. Typically, underfitting indicates the model is too simple or has too few explanatory variables. An underfitting model is not flexible enough to model real patterns and corresponds to high bias, indicating that the results show a systematic lack of fit in a certain region.

The following graph illustrates the clear difference between overfitting and underfitting as they correspond to a model with a good fit:

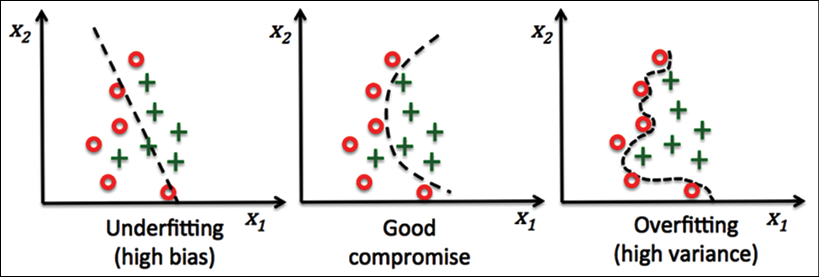


Figure 14.2: ML model overfitting versus underfitting

The ML model categorizes two data point categories illustrated by the preceding graphs' red points and green crosses. The ML model tries to determine whether a customer will buy a given product or not. The chart shows predictions from three different ML models. You can see an overfitted model (on the right) traversing through all red data points in training and failing to generalize the algorithm for real-world data outside of the training dataset. On the other hand, the underfitted model (on the left) leaves out several data points and produces an inaccurate result. A good model (shown in the middle) provides clear data point predictions in most cases. Creating a good ML model is like creating art, and you can find the right fit with model tuning.

The ML algorithm is categorized into supervised and unsupervised learning at the heart of the overall ML workflow.

**Supervised and unsupervised machine learning algorithms**

In supervised learning, the algorithm is given a set of training examples where the data and target are known. It can then predict the target value for new datasets containing the same attributes. For supervised algorithms, human intervention and validation are required, such as photo classification and tagging.

In unsupervised learning, the algorithm is provided with massive amounts of data, and it must find patterns and relationships between the data. It can then draw inferences from datasets.

Human intervention is not required in unsupervised learning, for example, auto-classification of documents based on context. It addresses the problem where correct output is not available for training examples, and the algorithm must find patterns in data using clustering.

Reinforcement learning is another category where you don't tell the algorithm what action is correct but give it a reward or penalty after each action in a sequence.

The following are the popular ML algorithm types used for ML:

* **Linear regression**: Let's use the price of houses as a simple example to explain linear regression. Say we have collected many data points representing the prices of homes and their sizes on the market, and we plot them on a two-dimensional graph. Now we try to find a line that best fits these data points and use it to predict the price of a house of a new size.
* **Logistic regression**: Estimates the probability of the input belonging to two classes, positive and negative.
* **Neural networks**: The ML model acts like the human brain, where layers of nodes are connected in a neural network. Each node is one multivariate linear function with a univariate nonlinear transformation. The neural network can represent any nonlinear function and address problems generally hard to interpret, such as image recognition. Neural networks are expensive to train but fast to predict.
* **K-nearest neighbors**: It chooses the number of *k* neighbors. It finds the *k-*nearest neighbors of the new observation that you want to classify and assigns the class label by majority vote. For example, you want to categorize your data into five clusters, so your *k* value will be five.
* **Support Vector Machines** (**SVMs**): Support vectors are a popular approach in research, but not so much in the industry. SVMs maximize the margin, the distance between the decision boundary (hyperplane) and the support vectors (the training examples closest to the boundary).

SVMs are not memory efficient because they store the support vectors, which grow with the size of the training data.

* **Decision trees**: In a decision tree, nodes are split based on features to have the most significant **Information Gain** (**IG**) between the parent node and its split nodes. The decision tree is easy to interpret and flexible; not many feature transformations are required.
* **Random forests and ensemble methods**: A random forest is an ensemble method in which multiple models are trained, and their results are combined, usually via majority vote or averaging. A random forest is a set of decision trees. Each tree learns from a different randomly sampled subset. Randomly selected features are applied to each tree from the original feature sets. Random forests increase diversity by randomly selecting the training dataset and a subset of features for each tree, reducing variance through averaging.
* **K-means clustering**: It uses unsupervised learning to find data patterns. K-means iteratively separates data into *k* clusters by minimizing the sum of distances to the center of the closest cluster. It first assigns each instance to the nearest center and then re-computes each center from assigned instances. Users must determine or provide the *k* number of clusters.

Zeppelin, RStudio, and Jupyter notebooks are the most common environments for data engineers doing data discovery, cleansing, enrichment, labeling, and preparation for ML model training. Spark provides the Spark ML library, which implements many standard high-level estimator algorithms such as regressions, page rank, k-means, and more.

For algorithms that leverage neural networks, data scientists use frameworks such as TensorFlow and MXNet, or higher-level abstractions such as Keras, Gluon, or PyTorch. Those frameworks and common algorithms can be found in the Amazon SageMaker service, which provides a full ML model development, training, and hosting environment. As the cloud is becoming a go-to platform for ML model training, let's learn about some available ML cloud platforms.

**Machine learning in the cloud**

ML development is a complex and costly process. There are barriers to adoption at each step of the ML workflow, from collecting and preparing data, which is time-consuming and undifferentiated, to choosing the right ML algorithm, which is often done by trial and error, to lengthy training times, which leads to higher costs. Then there is model tuning, which can be a very long cycle and requires adjusting thousands of different combinations. Once you've deployed a model, you must monitor it and then scale and manage its production.

To solve these challenges, all major public cloud vendors provide an ML platform that facilitates ease of training, tuning, and deploying ML models anywhere at a low cost. For example, Amazon SageMaker is one of the most popular platforms that provides end-to-end ML services. SageMaker provides users with an integrated workbench of tools brought together in one place through SageMaker Studio. Users can launch Jupyter Notebook and JupyterLab environments instantly through SageMaker Studio. SageMaker also provides complete experiment management, data preparation, and pipeline automation and orchestration to help make data scientists more productive. SageMaker also provides a fully managed RStudio platform, which is one of the most popular IDEs among R developers for ML and data science projects.

SageMaker provides fully managed servers in the cloud to make this easy for data scientists and developers. But even beyond notebooks, SageMaker provides other managed infrastructure capabilities as well. From distributed training jobs, data processing jobs, and even model hosting, SageMaker takes care of all of the scaling, patching, high availability, and so on associated with building, training, and hosting models. Similarly, GCP provides the Google Cloud AI platform with different services to perform ML experiments, and Microsoft Azure offers Azure Machine Learning Studio.

In addition to the managed ML platform, cloud vendors also provide ready-to-use **artificial intelligence** (**AI**) services. AI services allow developers to add intelligence to any application without needing ML skills easily. The pre-trained models provide ready-made intelligence for your applications and workflows to help you do things like personalizing the customer experience, forecasting business metrics, translating conversations, extracting meaning from documents, and more. For example, AWS provides the Amazon Comprehend AI service, which has pre-trained models that support entity detection, key phrase detection, and sentiment analysis natively in multiple languages.

Data scientists leverage the managed cloud environment to do data preparation and set up a model training cluster to start their training job. When complete, they can one-click-deploy the model and begin serving inferences over HTTP as you learn about algorithms and ML workflow to build a ML pipeline. Let's learn more about some of the important things to consider when you are designing ML architecture.

**Building machine learning architecture**

Creating an ML pipeline consists of multiple phases and requires iterative improvement. Building a robust and scalable workflow from a loose collection of code is a complex and time-consuming process, and many data scientists don't have experience building workflows. An ML workflow can be defined as an orchestrated sequence that involves multiple steps. Data scientists and ML developers first need to package numerous code recipes and then specify the order they should execute, keeping track of code, data, and model parameter dependencies between each step.

Added complexity to ML workflows warrants monitoring changes in data used for training and predictions because changes in the data could introduce bias, leading to inaccurate predictions. In addition to monitoring the data, data scientists and ML developers also need to monitor model predictions to ensure they are accurate and don't become skewed toward particular results over time. As a result, it can take several months of custom coding to get the individual code recipes to execute in the correct sequence and as expected.

ML architectures need to protect model artifacts and require self-service capabilities for model development and training. Your ML architecture needs to be an automated end-to-end evidence capture of the entire model development lifecycle across development, training, and deployment. ML application should use a **continuous integration and continuous deployment** (**CI/CD**) pipeline integrated with change control systems for model management and deployment. The environments require pre-defined security configurations. The following are the ML architecture components with examples from the **Amazon Web Services** (**AWS**) ML platform to understand it better.

**Prepare and label**

To make data ready for ML, you need to run your data processing workloads, such as feature engineering, data validation, model evaluation, and model interpretation. The feature also preprocesses datasets to convert the input datasets into a format expected by the ML algorithm you're using. You can use the various tools and techniques mentioned in the previous section, on processing data and performing analytics, to wrangle data as per your ML needs. A managed ML platform like **Amazon SageMaker** also provides a data wrangler and feature store capability to make the data processing job easier for you. Amazon SageMaker is a fully managed service that offers the ability to build, train, and deploy ML models quickly. Other ML platforms are Azure ML Studio, H2O.ai, SAS, Databricks, and the Google AI platform.

During data processing, you often need to label your data, and it becomes laborious in the case of image processing. Data labeling helps you build and manage highly accurate training datasets quickly. You can use third-party vendors who help you to label the image, such as Labelbox, CrowdAI, Docugami, and Scale. You can also automate the labeling process using AI services such as **SageMaker Ground Truth**, which continuously learns from labels provided by humans to improve annotation quality. Automatic annotations significantly lower labeling costs; once your data is ready, the next step is to select the suitable algorithm and build the model.

**Select and build**

While creating a ML model, you first want to understand business problems clearly, which will help you select the right algorithm. As explained in the previous section, you can choose from a list of algorithms and ML frameworks, both *supervised and unsupervised machine learning algorithms*. Once you select the suitable algorithm for your use case to build an ML model, you need a platform to train and develop your model.

Jupyter notebooks and RStudio are the most popular platforms among data scientists to build ML models. You can use cloud platforms such as Amazon SageMaker to spin up Jupyter notebooks or RStudio Workbench. AWS provides SageMaker Studio and RStudio a web-based visual interface where you can perform all ML development steps.

To select your model, you can choose several built-in ML algorithms that you can use for various problem types or get hundreds of algorithms and pre-trained models available in the cloud marketplace, making it easy to get started quickly. The next step is to train and tune the model. Let's learn more about it.

**Train and tune**

It would be best to get a distributed compute cluster, perform the training, and output the result that applications can consume for training. Model tuning is also known as hyperparameter tuning, which is a critical aspect of achieving result accuracy. You need to find the best model version by running multiple training jobs on the dataset using the algorithm and ranges of hyperparameters. Then it would be best if you chose the correct hyperparameter values that result in a model that performs the best, as measured by a metric that you prefer.

While you are tuning the model, you need to have the ability to debug the model, which helps to capture real-time metrics during training, such as training and validation, confusion matrices, and learning gradients, to help improve model accuracy. You need to capture the input parameters, configurations, and results and store them as experiments so that you can search for previous experiments by their characteristics, review previous experiments with their results, and compare experiment results visually. Most managed ML platforms, such as Amazon SageMaker, provide all these features like model autotuning, experiment, and debugger.

Amazon SageMaker also provides Autopilot, which automatically looks at raw data and applies feature processors. It picks the best set of algorithms, trains, tunes multiple models, tracks their performance, and ranks the models based on performance. Once your model is ready, you need to deploy it and manage it in production to get helpful insights.

**Deploy and manage**

You need to deploy your trained model into production to start generating predictions for real-time or batch data. You need to apply auto-scaling for ML instances across multiple locations for high redundancy and set up the restful HTTPS endpoint for your application. Your application needs to have an API call to an ML endpoint to achieve low latency and high throughput. This type of architecture allows you to integrate your new models into your application quickly because model changes no longer require application code changes.

Data can change quickly based on seasonality or unpredicted events, making it essential to monitor your model for accuracy and business relevance and remediate concept drift. Today, one of the significant factors that can affect the accuracy of deployed models is if the data that is used to generate predictions differs from the data used to train the model. For example, changing economic conditions could drive new interest rates affecting home purchasing predictions. This is called concept drift, whereby the patterns the model uses to make predictions no longer apply. You need to automatically detect concept drift in deployed models and provide detailed alerts that help identify the source of the problem.

In most deep learning applications, making predictions using a trained model—a process called inference—can be a significant factor in the compute costs of the application. A whole GPU instance may be oversized for model inference. In addition, it can be challenging to optimize the GPU, CPU, and memory needs of your deep learning application. You need to solve these problems by adding the right GPU-powered inference acceleration to production instances with no code changes.

Model compatibility is another crucial factor during deployment. Once a model has been built and trained using MXNet, TensorFlow, PyTorch, or XGBoost, you can choose your target hardware platform from Intel, NVIDIA, or ARM. You need to compile your trained ML models to run optimally and efficiently deploy compiled models to Edge devices and provide high performance and low-cost inference. You should have the ability to run large-scale ML inference applications like image recognition, speech recognition, natural language processing, personalization, and fraud detection as you learn various stages of building and deploying ML models. Let's look at a reference architecture to connect all components.

**Machine learning reference architecture**

The following architecture depicts a bank loan approval workflow based on customer data built on the AWS cloud platform.

Here, customer data ingested into the cloud and ML framework decides on the customer loan application.

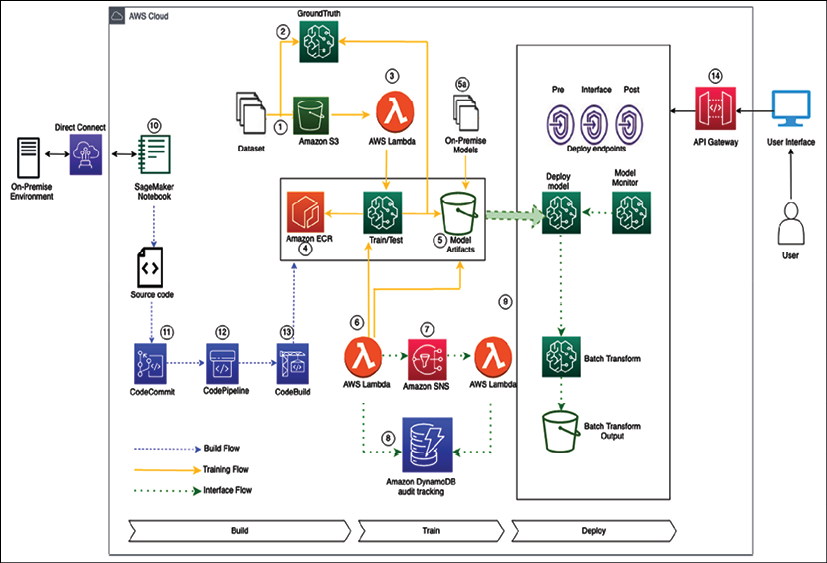


Figure 14.3: ML architecture in the AWS cloud

In designing the above architecture, some fundamental design principles to consider as a guide are:

* Training workflow:
  1. Datasets enter the process flow using S3. This data may be raw input data or preprocessed from on-premises datasets.
  2. Ground Truth is used to build a high-quality training labeled dataset for ML models. If required, the data can use the Ground Truth service to label the data.
  3. AWS Lambda can be used for data integration, preparation, and cleaning before datasets are passed to SageMaker.
  4. Data scientists will interface with SageMaker to train and test their models. The Docker images used by SageMaker are stored in ECR and can be custom images with custom toolsets that have been created through the build flow steps below or use one of the pre-built Amazon images.
  5. Model artifacts to be used as part of the deployment phase are output to S3. The output from the SageMaker model can also be used to label data using Ground Truth. Models that have been pre-built and trained on-premises or other platforms can be deposited into the model artifacts S3 bucket and deployed using SageMaker.
  6. AWS Lambda can trigger an approval workflow based on a new model artifact being deposited into the S3 bucket.
  7. Amazon Simple Notification Service can be used to provide an automatic or manual approval workflow based on human intervention to deploy the final model. The supporting Lambda function takes the output from SNS to deploy the model.
  8. DynamoDB is used to store all model metadata, actions, and other associated data for audit tracking.
  9. To host the final model, we deploy the endpoint with associated configuration as part of the final step in the workflow.
* Build flow:
  1. SageMaker notebook instances are used to prepare and process data and to train and deploy ML models. These notebooks can be accessed via a VPC endpoint for the SageMaker service.
  2. CodeCommit provides the repository for the source code to trigger the build jobs required for any custom Docker images used by SageMaker.
  3. The CodePipeline service manages the end-to-end build pipeline for the custom Docker images and uses the CodeBuild service for the build/test phase.
  4. CodeBuild will build and perform unit testing of the custom Docker image and push it to Amazon ECR (this process can be managed centrally or by business functions requiring the tools).
* Inference flow:
  1. As SageMaker endpoints are private, Amazon API Gateway exposes the model endpoint to end-users for inference.
  2. Batch transform jobs are typically used to get inferences for an entire dataset. Using a trained model and dataset, the output from the batch job is stored in S3.
  3. SageMaker Model Monitor is used to monitor production models to alert them to any quality issues.

This section taught you how to build a ML architecture with a CI/CD pipeline following ML architecture design principles. Earlier in this book, you learned about DevOps to automate and operationalize your development workload. As ML is becoming mainstream, MLOps become important to learn ML at scale in production. Let's explore more details on operationalizing the ML workload with MLOps.

**Machine learning operations**

An ML workflow is a set of operations developed and executed to produce a mathematical model, which eventually is designed to solve a real-world problem. But there is no value of these models until they are deployed in production, other than proofs of concept. ML models almost always require deployment to a production environment to provide business value.

At the core, **Machine Learning Operations** (**MLOps**) takes an experimental ML model into a production system. MLOps is an emerging practice different from traditional DevOps because the ML development lifecycle and ML artifacts are different. The ML lifecycle involves using patterns from training data, making the MLOps workflow sensitive to data changes, volumes, and quality. Additionally, matured MLOps should support monitoring both ML lifecycle activities and production model monitoring.

MLOps framework implementation makes it simple for organizations to feel confident in building a mature MLOps framework, eliminating extensive coding. Like any other workload, you want to develop MLOps by applying best practices such as security, reliability, high availability, performance, and cost for the deployment phase of the ML lifecycle. Let's look at some MLOps principles.

**MLOps principles**

Any changed code, data, or model should trigger the build process in the ML development pipeline. An ML pipeline should follow the below MLOps principles while developing ML systems:

1. **Automation**: Deployment of ML models in production should be automated. The MLOps team should automate the end-to-end ML workflow from data engineering to model interference in production without any manual intervention. The MLOps pipeline can trigger model training and deployment based on events such as calendar scheduling, messaging, monitoring, data changes, model training code changes, and application code changes.
2. **Versioning**: Versioning is an essential aspect of MLOps, the same as DevOps. Every ML model and related scripts version should be maintained in a version control system such as GitHub to make the models reproducible and auditable.
3. **Testing**: ML systems require extensive testing and monitoring. Each ML system should have at least the following three scopes for testing:
   * Features and data tests include validating data quality and selecting the right features for your ML model
   * Model development tests include business metric tests, model staleness tests, and model performance validation tests
   * ML infrastructure tests include ML API usage tests, full ML pipeline integration tests, and training and production server availability tests
4. **Reproducibility**: Every phase of a ML workflow should be reproducible, which means that data processing, ML model training, and ML model deployment should produce identical results given the same input. It will ensure a robust ML system.
5. **Deployment**: MLOps is an ML engineering culture that includes CI/CD and **Continuous Training/Continuous Monitoring** (**CT/CM**). Automated deployment/testing helps discover problems quickly and in the early stages. This enables the fast fixing of errors and learning from mistakes.
6. **Monitoring**: Model performance may degrade in production for reasons such as data drift. This means new models must be shipped into production constantly to address performance decline or improve model fairness. Once the ML model has been deployed, it needs to be monitored to ensure that the ML model performs as expected.

Having learned about MLOps design principles in this section, let's consider some best practices to apply MLOps in your machine leaning workload.

**MLOps best practices**

Due to many moving parts (data, model, or code) and challenges in solving business problems using ML, MLOps can be a challenging task.

Based on the principles outlined in the previous section, as follows are the best practices that ML engineers/full-stack data scientists should practice while deploying the ML solutions in production, which will help reduce the "technical debts" and "maintenance overhead" in ML projects and drive most business value out of it:

1. **Design considerations**: To develop a maintainable ML system, the architecture/system design should be modular and, as much as possible, loosely coupled.

Having a loosely coupled architecture allows the teams to work independently, without relying on other teams for support and services, enabling them to work quickly and deliver value to the organization.

1. **Data validation**: Data validation is very crucial for a successful ML system. In production, data may create a variety of issues. If the statistical properties of data are different from training data properties, the training data or the sampling process were faulty. **Data drift** might cause statistical properties to change for successive batches of data. Data drift may cause model performance to degrade over time as input data properties change in comparison to the data used during ML model training.
2. **Model validation**: Reusing models is different from reusing software. You need to tune models to fit each new scenario. Validating models before promoting them into production is very important. To establish the adequate performance of the model on live data, you should perform online and offline data validation.
3. **Model experiment tracking**: Always keep track of ML model experiments. Experimenting may involve trying out different code combinations (preprocessing, training, and evaluation methods), data, and hyperparameters. Each unique combination produces metrics that you need to compare to your other experiments.
4. **Code quality check**: Every ML model specification (ML training code that creates an ML model) should go through a code review phase. It's good practice to include this code quality check as the first step of a pipeline triggered by a pull request.
5. **Naming conventions**: Following a standard naming convention (like *PEP8* for Python programming) in your ML code helps mitigate the challenge of the **Changing Anything Changes Everything** (**CACE**) principle. It also helps team members establish familiarity with your project quickly.
6. **Model predictive service performance monitoring**: Other than project metrics (such as RMSE and AUC-ROC) that evaluate a model's performance in relation to business objectives, operational metrics such as latency, scalability, and service update are also crucial to monitor to avoid business losses.
7. **Continuous Training**(**CT**) and**Continuous Monitoring**(**CM**)**process**: Model performance may degrade in production for reasons such as data drift. It means new models must be deployed into production constantly to improve model fairness. This calls for CT/CM.
8. **Resource utilization**: Understanding the requirements of your system during the training and deployment phases helps your team optimize the cost of your experiments.

MLOps plays a crucial role in the industrialization of AI. MLOps combines ML, DevOps, and data engineering with the goal of reliably and efficiently building, deploying, and maintaining ML systems in production. Deep learning is now the go-to mechanism to solve complex ML problems. Let's learn some more details about deep learning.

**Deep learning**

ML is not just about forecasting numbers but also solving complex problems using neural language processing. These use cases include complex scenarios processed by the human brain, such as building an automated chatbot impersonating humans, reading handwritten text, image recognition, transcribing videos/audios, and converting text to audio and vice versa. **Deep learning** has the ability to solve such use cases by mimicking the human brain.

While ML needs a pre-defined set of labeled data using supervised learning, deep learning uses a neural network for unsupervised learning to simulate human brain behaviors by using a large amount of data to develop learning capabilities for machines. Deep learning is a neural network of multiple layers where you don't need to do data labeling upfront. However, you can use both labeled data and unlabeled data with deep learning, depending upon your use case. The following diagram shows a simple deep learning model:

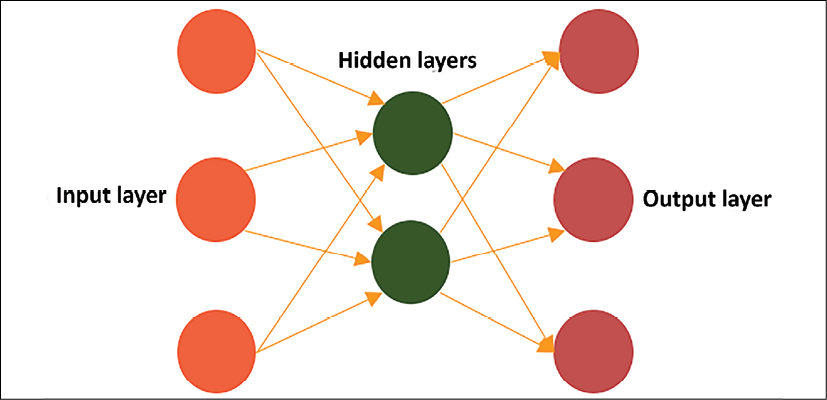


Figure 14.4: A overview of deep learning layers

In the preceding diagram, a deep learning model has interconnected nodes where input layers provide data input through various nodes. This data goes through multiple hidden layers to calculate the output and deliver final model inference through the output node layer. The input and output layers are visible layers, and learning happens in the middle layer through weights and bias, as shown in the diagram below:

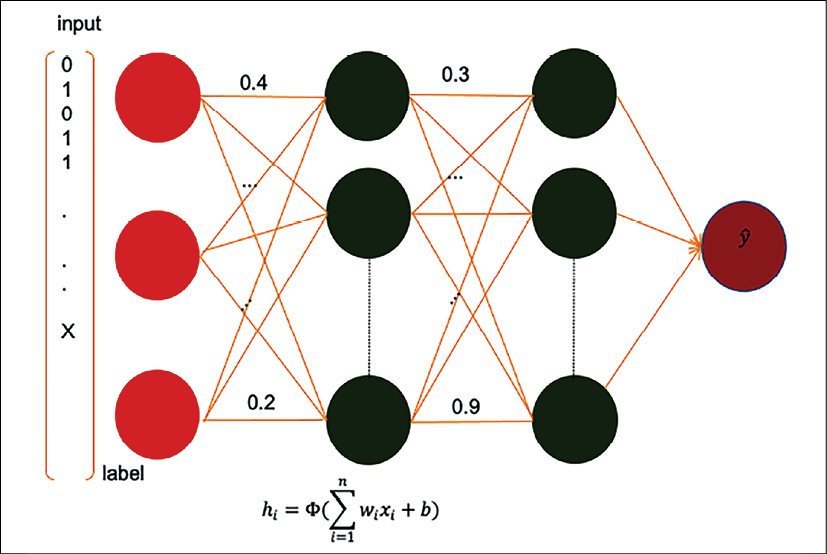


Figure 14.5: Deep learning neural network model

As shown in the preceding diagram, you can see a series of hidden layers in between where each layer applies some weight functions to these interconnected nodes to learn the pattern the same as the human brain, to provide the desired outcome. You can see **label** data is coming as input and going through neural network nodes with their weight (**0.2**, **0.4**, **0.3**, and **0.9**) indicated between vertices.

Weight is a neural network parameter that transforms input data within the hidden layers. Weight decides how much influence the input will have on the output. It represents the strength of the connection between nodes. If the weight from node A to node B has a greater degree, it means that neuron A has greater influence over neuron B. Weights near zero means changing this input will not have any effect on the output. If weights are negative, this means increasing the input will decrease the output and vice versa.

The above learning method is called **forward propagation**, where data flows from the input to the output layer. Another technique called **back propagation** uses algorithms to calculate errors in predictions and then adjust the weights and biases of the function by moving backward through the layers to train the model. With the help of forwarding and backward propagation, you can build a neural network to make predictions and correct errors and gradually become more accurate with the training algorithm.

Deep learning has different types of neural networks. The two most common are **Convolution Neural Networks** (**CNNs**), used on computer vision and image classifications, and **Recurrent Neural Networks** (**RNNs**), used for natural language processing and speech recognition. Some of the most popular frameworks to build neural network models are:

* **TensorFlow**: It is an open-source software library for ML. TensorFlow's main API is written in Python and has experimental support for other languages. It has built-in support for many neural network architectures.
* **MXNet**: MXNet is also an open-source software library for deep learning natively implemented in C++ and has built-in support for many network architectures. Its API is available in multiple languages such as Python, Scala, Clojure, R, Julia, Perl, and Java (inference only).

In addition to the above, other popular deep learning frameworks are PyTorch, Chainer, Caffe2, ONNX, Keras, Gluon, and so on. The idea of this section is to provide you with a high-level view of deep learning. It is a complex topic and requires an entire book to cover the basics. You will find multiple books available on each of the frameworks. Deep learning model training requires a large amount of processing power and could be very costly. However, public cloud providers such as AWS, GCP, and Azure make it easy to make available high-powered GPU-based instances to train these models with the pay-as-you-go method.

Now, ML is applicable everywhere, which includes solving customer problems such as predictive maintenance, providing accurate forecasting for businesses, or building personalized recommendations for end-users. ML use cases are not only limited to customer problems, but also help you to handle your IT applications by optimizing your workload with predictive scaling, identifying log patterns, fixing errors before they cause issues in production, or budget forecasting for IT infrastructure. So, it's important for solution architects to be aware of ML use cases and associated technology.

Overall, ML and AI are very vast topics and warrant multiple books to understand them in more detail. In this chapter, you just learned an overview of ML models, types, and the ML workflow.

**Summary**

In this chapter, you learned about ML architecture and components for a ML workflow. You learned about how data and ML go hand in hand. It is essential to get high-quality data with feature engineering to build the right ML model.

You learned about ML model validation by recognizing model overfit versus underfit situations. You also learned about various supervised and unsupervised ML algorithms. As the cloud is becoming a go-to platform for ML model training and deployment, you learned about ML platforms in popular public cloud providers.

Further, you learned about the ML workflow, including data preprocessing, modeling, evaluation, and prediction. Also, you learned about building ML architecture with a detailed reference architecture built in AWS cloud platforms. MLOps is essential for putting ML models in production. You learned about MLOps principles and best practices. Further, you got an overview of deep learning, which helps solve complex problems by mimicking the human brain.

There are millions of small devices connected to the internet, referred to collectively as the IoT. You need to understand the various components available in the cloud to collect, process, and analyze IoT data to produce meaningful insights. In the next chapter, you will learn more details about IoT use cases and solve them. You will learn about challenges with IoT systems and the techniques used to scale them.