NetApp

White Paper

Building a data pipeline for deep learning

Take your AI project from pilot to production

Santosh Rao, NetApp
March 2019 | WP-7299

Abstract

This white paper describes the considerations for taking a deep learning (DL) project from initial conception to production, including understanding your business and data needs and designing a multistage data pipeline to ingest, prep, train, validate, and serve an artificial intelligence (AI) model.

TABLE OF CONTENTS

Intended audience **4**

Introduction **4**

Challenges to a successful AI deployment 4

What is a data pipeline? **5**

Understanding your business needs **6**

Understanding your data needs **7**

Data needs for various industry use cases 9

Ingest data and move data from edge to core **10**

Streaming data movement 11

Batch data movement 11

Prepare data for training **11**

Accelerate data labeling 13

Deliver data to the training platform **13**

Copy data into the training platform 13

The training platform accesses data in place 14

Tiering data into the training platform 14

Train a deep learning model **14**

Types of neural networks 15

Popular deep learning frameworks 17

Depp learning software platforms 17

Model validation and evaluation 17

Model serving and deployment **18**

Platform options 18

Version history **19**

LIST OF TABLES

Table 1) Common data types in DL. 8

Table 2) Common data preparation steps for various data types. 12

Table 3) Common neural networks and associated use cases. 16

LIST OF FIGURES

Figure 1) Most of the time needed for a DL project is spent on data-related tasks. 4

Figure 2) Stages in the data pipeline for DL. 5
Figure 3) Popular AI use cases in different industries. 7
Figure 4) Data often flows from edge devices to core data centers or the cloud for training. 10
Figure 5) Copying data into the training platform from a data lake or individual data sources. 13
Figure 6) Training platform accessing data in place. 14
Figure 7) Simplified illustration of a deep neural network. 15

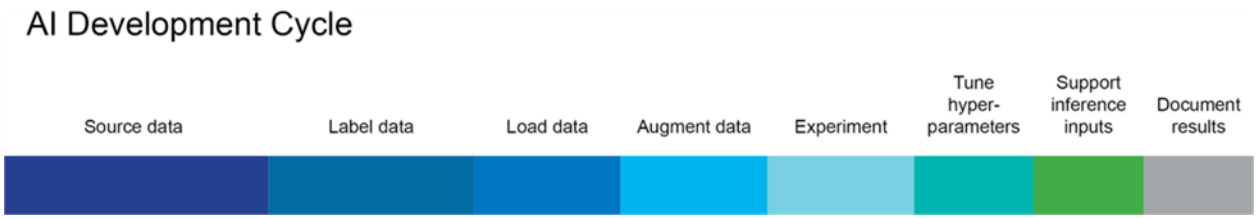
Intended audience

This white paper is primarily intended for data engineers, infrastructure engineers, big data architects, and line of business consultants who are exploring or engaged in deep learning (DL). It should also be helpful for infrastructure teams that want to understand and address the requirements of data scientists as artificial intelligence (AI) projects move from pilot to production.

Introduction

There are many ingredients for AI success, from selecting the best initial use case, to assembling a team with the right skills, to choosing the best infrastructure. Given the complexity, it's easy to underestimate the critical role that data plays in the process. However, if you look at the timeline for a typical AI project, as illustrated in Figure 1, most of the time is spent on data-related tasks such as gathering, labeling, loading, and augmenting data.

Figure 1) Most of the time needed for a DL project is spent on data-related tasks.



This is where the concept of a data pipeline comes in. A data pipeline is the collection of software and supporting hardware that you need to efficiently collect, prepare, and manage all the data to train, validate, and operationalize an AI algorithm.

The need for a well-designed data pipeline may not be immediately evident in the early stages of AI planning and development, but its importance grows as data volumes increase and the trained model moves from prototype to production. Ultimately, your success may hinge on how effective your pipeline is. If you don't start thinking about how to accommodate data needs early enough, you are likely to end up doing some painful rearchitecting.

This white paper is intended to help you understand the elements of an effective data pipeline for AI:

- What are the most common options in the software stack in each stage?
- When should various software options be applied?
- How do the software and hardware work together?

Although the focus of this paper is on building a data pipeline for DL, much of what you'll learn is also applicable to other machine learning (ML) use cases and big data analytics.

Challenges to a successful AI deployment

Any AI project may face challenges in a variety of areas.

Data engineering. Much of the time for an AI project is likely to be spent on data-related tasks like labeling, normalization, tokenization, and more.

Computation. Training a DL algorithm requires large numbers of cores. Graphics processing units (GPUs) with thousands of cores and purpose-built hardware interconnects between GPUs are necessary for AI training with large datasets.

Algorithm choices. Training is mostly done in parallel across many GPUs.

- If the model fits inside GPU memory, then copies of the model are deployed on many GPUs and a subset of data is fed to each GPU in parallel for training. GPUs need to communicate with each other either synchronously or asynchronously, with a trade-off between better convergence versus lower communication costs.
- If the model cannot fit in the memory of a single GPU, then the model itself has to be split and run across many GPUs.

Infrastructure challenges. GPUs, CPUs, network switches, and fast storage all have to be carefully architected to achieve the optimal configuration and avoid bottlenecks. This applies to both training the model and inferencing once a model has been trained.

Operational challenges. These challenges are divided into ScienceOps and DevOps:

- ScienceOps involves data provenance, experiment management, hyperparameter tuning, and other tasks necessary to deliver a well-trained DL model.
- DevOps involves tasks such as how AI training jobs are submitted (batch or interactive), monitoring of jobs, scheduling, and so on. Once a model is completed, further DevOps tasks are required to incorporate that model with code and to manage deployments over time as the model is retrained, enhanced, or the software using the model output evolves.

What is a data pipeline?

The main purpose of a data pipeline is to enable you to gather and manage the datasets needed for model training. This includes getting the data into a form that your model can digest and understand. The underlying architecture of your pipeline will vary depending on the sources and data types you are drawing from. Data types and data sources are discussed in section 5, Understanding Your Data Needs.

A data pipeline is logically divided into stages, as shown in Figure 2

Figure 2) Stages in the data pipeline for DL.

Here's what happens in each stage in a typical DL use case:

- **Data ingest.** Ingestion often occurs at the edge—for example, capturing data from cars, point-of-sale devices, or cameras on an assembly line. Depending on your use case, you may need to stream data as it comes in, or you may cache the data at the edge and forward it periodically.
- **Data prep.** Some form of preprocessing is almost always necessary to prepare data before training. Preprocessing often takes place in a data lake.
- **Training.** During training, datasets may be copied from the data lake into a training cluster, which can have a tremendous appetite for I/O.
- **Validation.** Before a model can be deployed, it has to be validated to ensure that the results it delivers are valid. Several cycles of data prep, training, and validation may be required to identify the features and parameters that yield the best results.
- **Deployment.** The resulting model is moved to software development and then to production. Depending on the use case, the model might be deployed back to edge operations. Real-world results of the model are monitored, and feedback in the form of new data flows back into the data lake, along with new incoming data to iterate on the process and periodically retrain the model.
- **Archive.** Data used in each training run may be saved indefinitely. Many AI teams archive cold data to object storage in either a private or a public cloud.

All of these stages are important, but the transitions between stages are often just as crucial. Data has to flow efficiently from ingest to data prep to training to validation. Later chapters of this paper examine the stages in detail, including important transitions.

Software 1.0 versus software 2.0

In conventional software development, developers do coding work on laptops or small workstations and the completed software is deployed on high-end production servers in a well-understood workflow:

Development > Deployment > Production

By comparison, in AI development the "development server" is often a GPU supercomputer (training cluster) and the trained model is deployed on a smaller system that is GPU-enabled or that contains other specialized hardware for inferencing. Deployment of a model from the training system to inference is called serving. The workflow for AI therefore looks like this:

Training > Serving > Inference.

Understanding your business needs

AI is becoming a foundational technology with benefits for every industry. It is transforming industries such as agriculture, manufacturing, automotive, pharmaceuticals, and financial services. In many industries, ML and DL are already essential to competitiveness and long-term viability.

The first step in any AI project is to analyze your business needs and choose the best use case for your organization to tackle. Many teams begin with a use case that is prevalent in their industry, particularly if it's their first AI project.

It can be helpful to consider a "land and expand" strategy, picking a use case that has a good chance of producing results in a relatively short period, and then expanding on that use case in other areas of the business. For example, Liberty Mutual Insurance started with a digital assistant (or chatbot) for its internal employees, a relatively safe use case with no direct customer exposure. Success allowed Liberty Mutual to pilot a similar technology approach in its customer call center and also to commercialize the software through its Workgrid Software subsidiary.

Figure 3 summarizes some common use cases for a variety of industry sectors.

Figure 3) Popular AI use cases in different industries.

Understanding your data needs

"Powerful compute resources and machine learning frameworks can be used only when a company has relevant data assets and has taken steps to prepare and transform data to be used to train models."

–Implementing AI: From Exploration to Execution

After you've chosen an AI use case to pursue, the next step is to critically assess the data you have to support that use case. If you don't have the necessary dataset, you'll have to figure out how to get the data you're missing. In some cases, you may have to table the use case until you can put in place the processes and infrastructure to gather the necessary data.

If your organization is new to big data and AI, you may want to start thinking about all of your company's data assets now. Having the right governance policies in place helps to ensure that datasets are available, usable, and consistent when your data science teams need them. It's also important to ensure that key regulations for data protection and data privacy are met.

An important realization when thinking about your data is that not all datasets can be treated the same; there are many different data types. Your use case may require only one data type, or it could encompass multiple types, and the data types you are using may determine the software tools needed in each stage of the data pipeline.

A number of common data types along with examples and sources are shown in Table 1.

Table 1) Common data types in DL.

Data type	Examples	Sources
Images	• Assembly line cameras • Medical imaging • Geospatial Imagery • Geologic images • Thermal Imaging • Lidar 3D Point Cloud	• NFS • Lustre • GPFS
Video	• Security cameras • Autonomous vehicles • Drones • Cobots	
Audio	• Voicemail • Customer service calls • Acoustic data	
Time-Series Data	• Internet of Things (IoT) • Securities prices • Scientific data	• NoSQL databases (Cassandra, AeroSpike)
Text	• Log data • Unstructured text • Documents	• Splunk, ELK, etc. • NFS, Hadoop/HDFS • NoSQL databases (MongoDB)
Graph	• Social media data • GPS navigation	• Graph databases

Each data type may be stored in a unique way. As Table 1 also shows, you may have to incorporate data from a variety of sources, from Hadoop to NoSQL databases to file systems such as NFS and Lustre. You may need to work with data that has already been collected, as well as streaming data from video cameras, sensors, applications, ecommerce transactions, and so on. (Streaming data is covered in Chapter 6, “Ingest Data and Move Data from Edge to Core to Cloud.”)

The algorithm you are training may only require data from internal sources, but you may also need externally sourced data, such as weather data, demographics, or social media posts.

For each source (internal and external) you need to make sure that you have a right to the data, that you’re not violating compliance guidelines or regulations, and that you can access the data in a consistent format in the necessary timeframe.

Why the three versus matter?

AI learns from the features and attributes of your data. The three Vs of big data—volume, variety, and veracity—therefore have a direct influence on the accuracy of your AI model.

Volume. As a rule, the greater the volume of data the better the performance of the DL system.

Variety. Variety means having diverse attributes and features in the dataset. The greater the variety, the more accurately a DL model can generalize.

Veracity. In computer vision and other models, learning requires labeled data. Correct labeling of data (veracity) is crucial to accurate models. Hand-labeled training datasets are expensive and time consuming to create. Therefore new techniques are being developed to programmatically generate training datasets..

Data needs for various industry use cases

“In contrast with a traditional software product, it is data rather than code that is of primary importance in a deep learning system.”

–Allegro.ai

If the preceding discussion seems a bit esoteric, a few use cases should help clarify the data needed for AI.

Retail: Inventory optimization

A global retailer, such as a convenience store, needs to take many factors into account to optimize the inventory in each location. A retail environment can be extremely dynamic, with perishable items needing to be restocked daily, while beverages and other items are restocked just one to three times per week.

The necessary data includes:

- **Localization.** What products are popular in the area where the store is located?
- **Product trends.** What is the sales trend for each product stocked?
- **Historic trends.** What did product sales look like last month? Last year?
- **Promotions.** What is the impact of coupons or other promotions?
- **Demographics.** Who lives in the area surrounding this location?
- **Weather.** What is the weather forecast for the forecast period, and how will that affect overall sales and sales of individual products?

Healthcare: Smart inhaler

In the United States alone, there are 25 million asthma sufferers—1 in every 13 people. Adding a few sensors and Bluetooth connectivity to asthma inhalers can provide a wealth of useful data. Because

patients often use inhalers in response to symptoms, there's a huge opportunity to correlate usage and location data. Combining patient data with information such as weather, air quality, and pollen counts has the potential to help patients avoid triggers in real time, minimizing risks and improving their overall health.

Computer vision

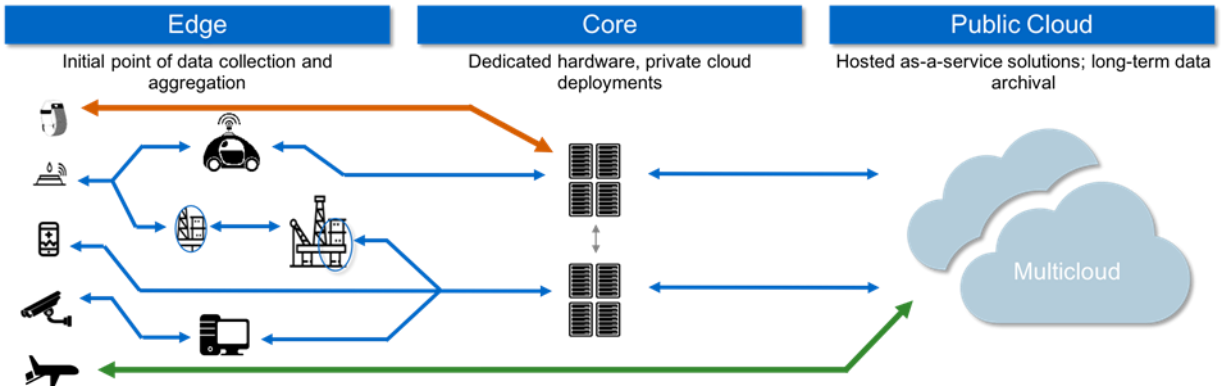
Before you reach the mistaken assumption that every DL use case requires correlations across diverse datasets and data types, let's look at computer vision, which has applications across almost all industries and often involves only images or video. For example, manufacturers often use computer vision to detect anomalies in finished parts, and it is also applied in healthcare to interpret CT, MRI, and other diagnostic scans. If you've ever used your mobile phone to deposit a check to your bank account, that's a computer vision application that has become so commonplace in the last few years that it's easy to overlook.

NetApp partner Allegro.ai specializes in the use of DL in computer vision with a deep understanding of data needs. Allegro.ai frames the central challenge of computer vision this way: "A neural network for computer vision will only perform well in the field if it is thoroughly trained with a dataset that effectively mimics the environment it will see in production. While this requires large amounts of data, it is not the sheer volume that determines success but the degree to which the training images and videos reflect the specific reality your edge devices will be exposed to."

Ingest data and move data from edge to core

Data ingest is an important consideration for every DL use case, but it's particularly crucial for use cases that involve data sources at the edge. Data ingest is likely to be much less of a concern when you're sourcing data from an image archive than—for example—when you're gathering IoT data from sensors distributed across a dozen production and distribution facilities. Figure 4 illustrates how data flows from edge to core to cloud.

Figure 4) Data often flows from edge devices to core data centers or the cloud for training.



Some data types lend themselves to a batch approach to data ingest, in which you store and periodically forward data, while others require a streaming approach. With either method, the destination is likely to be a data lake, where you can explore and transform the data as needed.

As a rule, a mapping exists between the type of data you're ingesting and the method of data movement that is needed. For example, text and numeric data sources tend to use streaming data movement, while image, video, and audio sources may use streaming data movement or batch data movement.

Once you train and deploy a DL model, you may decide to transition from streaming to batch data movement. For example, you might stream all images needed to initially train a computer vision model. However, once you have a trained model deployed at the edge, you can transition to storing data locally and forwarding only "interesting" images—for instance, ones that the model has trouble with.

Streaming data movement

Streaming data movement is often accomplished by using the Apache open source tools Kafka, Flume, and NiFi:

- Kafka is a well-known and widely used message broker that allows data "consumers" to subscribe to streams from "producers." Strengths include reliability and scalability. Limitations include a need for custom coding and message size limits.
- Flume is well suited for moving high-volume streaming data into Hadoop. Limitations include the chance of data loss in some failure modes and message size limits.
- NiFi provides real-time control that makes it possible to manage data movement between any source and destination. It is well suited for mission-critical data movement with security and compliance requirements and has no message size limit. NiFi lacks the data replication capability of Kafka.

For help choosing among these tools, use the links in the preceding list. For a useful comparison of the three, see Big Data Ingestion: Flume, Kafka, and NiFi.

If your use case requires IoT data from industrial equipment and sensors, you may also need to accommodate one of several specialized protocols used in the space. IoT traces its roots back to the 1980s and 1990s and has evolved its own protocols and standards to support a variety of low-memory, low-power devices. Common protocols include:

- **MQTT.** Message Queue Telemetry Transport
- **DDS.** Data Delivery Service
- **AMQP.** Advanced Message Queuing Protocol

For wireless data transmission, Zigbee is popular, and you'll also see more familiar protocols, including Bluetooth, WiFi, and cellular networks. The forthcoming 5G cellular technology is expected to play a big role in IoT in the future.

Batch data movement

If you have the ability to store data near the edge and forward it periodically, a variety of Linux commands can do the job, such as rsync, or even cp or mv. However, use of these tools does require some scripting. You can also use REST API calls, such as object puts, to move data items directly into an object store such as Amazon S3.

There are vendor tools that are well suited to this purpose, eliminating the need to maintain scripts or programs while integrating encryption for security, as well as data efficiency technologies such as deduplication and compression to utilize network bandwidth and storage capacity efficiently.

- **NetApp SnapMirror.** Provides data aggregation and data movement from edge to core with data reduction and encryption for efficiency and security.
- **NetApp Cloud Sync.** Provides seamless and secure data synchronization from edge to cloud.

Prepare data for training

The data prep stage of the AI data pipeline encapsulates a number of functions, the purpose of which is to create a dataset that is suitable for training and validating a DL model. Keep in mind that data prep isn't necessarily a discrete function. You may do some preprocessing on ingest and some prior to training, or you may do all data prep in line with the training process.

Data prep can include:

- **Exploring the data.** What is the hypothesis for your model, and what features of the data are likely to be predictive?

- **Cleaning up data types and data formats.** Training is likely to go more smoothly if your data is consistent; however, you don't want it to be too much more consistent than the live traffic the model will receive.
- **Adjusting the training dataset.** You need to make sure that the feature you are training the model on is adequately represented. For example, you can't train an effective model for anomaly detection if your dataset consists only of images of "good" parts.
- **Labeling datasets.** For supervised learning, you need datasets that are labeled.
- **Splitting the dataset into training, validation, and testing sets.** You need enough data to provide a training set, a separate validation set that is used during training (but that the model is not trained on), and a testing set to assess the performance of the trained model.

Data prep, especially in early phases of model development, is frequently an iterative, exploratory process in itself, aimed at understanding which processes deliver the best results. Table 2 shows common data preparation activities for various data types.

Table 2) Common data preparation steps for various data types.

Data type	Common data preparation steps
Images	• Make sure that all images are the same size and resolution. • Make sure that all images are black and white or all color. • Label features in images. • Correct any data imbalances (using oversampling, undersampling, data augmentation, class weights).
Video	• Extract JPEGs or BMPs of each frame. • Scale image size up or down as needed. • Correct any data imbalances (using oversampling, undersampling, data augmentation, class weights).
Audio	• Choose sampling rate. • Transform signal from time domain to frequency domain. • Use magnitude compression.
Time-Series Data	• Normalization (all values between 0 and 1). • Standardization (rescaling values so mean is 0 and standard deviation is 1).
Text	• Normalization (eliminate case and punctuation, convert numbers to text, etc.). • Tokenization (split text into "tokens" that represent words, sentences, or paragraphs). • Noise reduction (remove headers and footers, HTML, metadata, etc.).

Accelerate data labeling

Because data preparation is so time consuming, a number of companies specialize in helping streamline the process. In particular, many DL use cases involve supervised learning, which requires labeled data. Data labeling is therefore a big issue for many DL projects.

It's rare for labeled datasets to exist with the specificity you require, so data labeling is often a labor-intensive manual process. Business process outsourcing companies can provide cheap labor to carry out manual data labeling.

NetApp is collaborating with the following companies that are innovating in data preparation and data labeling automation to accelerate data labeling tasks:

- Figure Eight is developing a platform that can transform text, image, audio, and video data into customized training data, including machine-learning-assisted data labeling. Figure Eight technology learns from labeled examples that you create and then labels your remaining data.
- The Hive provides a data labeling solution targeted specifically for computer vision use cases.

Deliver data to the training platform

Once your dataset is prepared and it's time to train your model, the next step is to deliver the dataset to the training platform. For large DL models, the training platform frequently consists of a cluster of systems with multiple GPUs running in parallel.

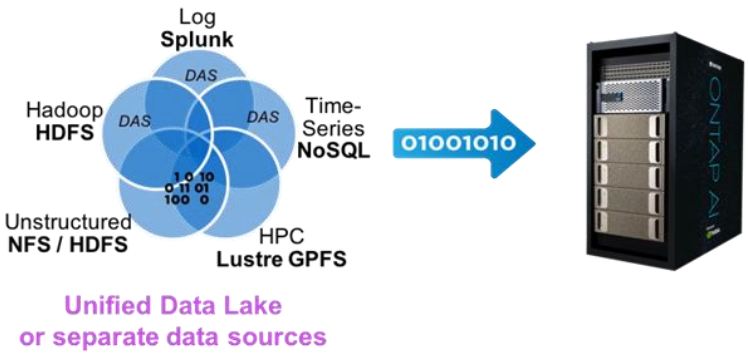
- There are three ways to get data to the training platform:
- Copy the data into the training platform
- Allow the training platform to access the data in place
- Tier the data to the training platform

Which method you choose can depend on a number of factors, including where your data is stored, how big your dataset is, and the capabilities of your training platform.

Copy data into the training platform

The most common method for getting data into the training platform, used in 80% to 90% of cases, is simply to copy the training dataset into the training platform, as illustrated in Figure 5.

Figure 5) Copying data into the training platform from a data lake or individual data sources.



The dataset is commonly staged in a data lake and copied into the training platform. If you don't have a central data lake, you may choose to copy data from the various data sources into the training platform as needed. If this is impractical because of the amount of data relative to the memory and/or storage

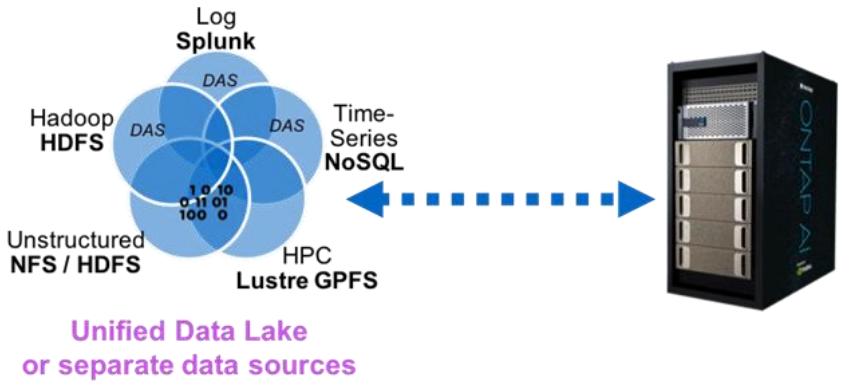
capacity of your training platform, another variation is to cache data from the various data sources on fast storage near the training platform. NetApp FlexCache is one solution that makes this possible.

Another variation is to copy data directly into the training platform on ingest. Streaming data brokers such as Kafka make this approach simple; a data producer can have multiple data consumers, so you can send ingested data simultaneously to a data lake and to your training platform.

The training platform accesses data in place

A less common alternative for training is to have the platform access data in place, as shown in Figure 6. Again, this access is commonly from a data lake; or it could be directly from other data sources. This is a good option when the dataset is stored on high-performance storage and the dataset size exceeds the capacity of the training platform. Adding a high-performance cache is an alternative for data sources that don't deliver adequate performance on their own.

Figure 6) Training platform accessing data in place.



In theory, this makes it possible to share a single copy of the same data for big data analytics, ML, and DL. However, in practice you will probably need to prepare and transform the data for DL in unique ways, requiring a separate copy. An alternative to a full physical copy is a clone of the data, which consumes additional space only as changes are made. For example, with NetApp FlexClone®, you can make as many clones as you need while only consuming capacity incrementally and with no impact on performance.

Tiering data into the training platform

A final option for delivering datasets for training is to use a tiering solution. This can be a great option when you are pulling in data from cold storage. In that situation, you can spend a lot of time and effort just getting data where it needs to be. Tiering solutions move data from cold storage transparently when it is accessed, eliminating the need for manual or scripted data movement. As a model matures and is repeatedly retrained, tiering is a good way to maintain access to previous versions of training datasets.

For example, NetApp FabricPool can tier data between on-premises and cloud object storage. Depending on your training platform, you can tier data directly into the training platform or near it.

Train a deep learning model

When your dataset has been prepared, it's time to begin training your DL model. Almost without exception, training depends on trial and error to deliver good results (and minimize computational overhead).

The recipe for training DL models involves three main ingredients:

- Search for the best model architecture
- Scale computation
- Accommodate very large training datasets

Training can involve a lot of experimentation. Data scientists may want to try the latest and most innovative model architectures. Infrastructure engineers may test novel parallelization methods—or source better GPUs—to scale compute. Data engineers work to prepare new and larger datasets.

This chapter helps you understand a few basics about the different types of neural networks, as well as the DL software frameworks that you'll probably be using. It also introduces a few software platforms that can integrate and streamline more of the DL process.

Addressing DL computation and I/O requirements

The datasets used to train DL models are incessantly increasing in size. Larger datasets require more compute during training. However, compute requirements for DL are empirically predictable, so it's possible to apply DL on small datasets to choose the model architecture and then refine the model with larger datasets to attain the desired accuracy.

As DL proceeds toward bigger and more complex models, training can run into compute bottlenecks, mostly limited by the amount of GPU memory. Alternatives to these challenges include upgrading to the latest GPU architectures (with increased memory) or migrating to GPU "supercomputers" that combine large numbers of interconnected GPUs acting as a single giant GPU.

To improve training times, you can explore parallelization techniques such as data parallel, model parallel, and pipeline parallelism. The tradeoff is having to revisit the model architecture in order to make these approaches work.

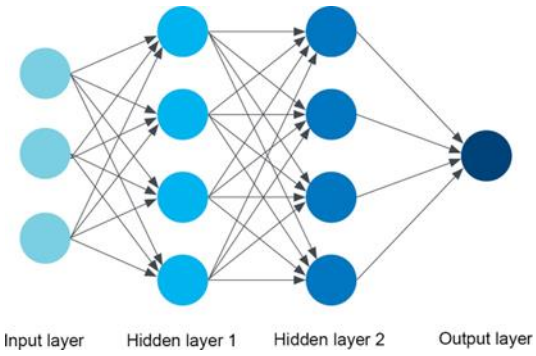
Computational limits can prevent data scientists from using models that are too compute intensive. This may mean trading compute performance for statistical accuracy. In use cases such as medical imaging, very high accuracy is essential, but for other use cases—such as recommender systems— a faster response may be preferable to greater accuracy.

Expect to see continued rapid evolution in approaches to address the problems of dataset size and training time.

Types of neural networks

If you're new to DL, it's helpful to know about the different classes of artificial neural networks and where they might be applied**.** Although it's not necessary to understand how each type of neural network is architected to use it, it's helpful to be familiar with a few basic concepts. A neural network consists of an input layer and an output layer, separated by a number of hidden layers.

Figure 7) Simplified illustration of a deep neural network.



The nodes in the hidden layers of a neural network mostly execute simple mathematical calculations, called activation functions. The way the nodes are interconnected, the mathematical functions they use, and the breadth and width of the network all contribute to the architecture of the deep neural network and determine what type of network you have.

Table 3 shows some of the most commonly used network types, but keep in mind this list is by no means exhaustive and new approaches are emerging all the time. Hybrid approaches that combine elements from different network types are also in use.

Table 3) Common neural networks and associated use cases.

Neural network type	Common use cases
Convolutional neural network (CNN) Classify, cluster, and identify	• Image and video recognition • Recommendations • NLP
Recurrent neural network (RNN) Long/short-term memory (LSTM) Recognize patterns in sequences of data; can link cause and effect	• NLP • Video • Text • Speech • Time-series
Generative adversarial network (GAN) Suited to creative text and image tasks	• Text to image • Image-to-image translation • Video learning and synthesis
Deep reinforcement learning (DRL) Can learn how to attain a goal, such as playing a game	• Traffic scheduling in mobile networks • Recovery controller for mobile robots • Large-scale fleet management • Medical imaging

For schematics and more in-depth descriptions of these and other network types, check out "The mostly complete chart of Neural Networks, explained."

From a practical standpoint, designing and training a neural network involves a number of considerations, including choosing the right connectivity (network type), optimizers, loss functions, and activation functions, as well as tuning hyperparameters.

Popular deep learning frameworks

For most teams embarking on a DL project, neural network specifics are likely to be a bigger consideration than the framework to be used. The framework decision is usually based on experience—assuming that at least one person has prior experience with any of the frameworks—the specific use case you are undertaking, and language support.

Popular DL frameworks include:

- TensorFlow
- Keras
- PyTorch
- Caffe and Caffe2
- Microsoft Cognitive Toolkit (CNTK)
- MXNet
- DeepLearning4j
- Chainer
- Neural Network Libraries
- PaddlePaddle

Depp learning software platforms

A number of software companies are in business to streamline the process of AI model development to help companies gain more value from data more quickly. NetApp is collaborating with several of these companies, including the following.

Deep learning

- Allegro.ai. Allegro.ai is developing a DL platform that is optimized for computer vision and focuses on data preparation, including data labeling, training, and deployment. Allegro.ai supports AI frameworks including TensorFlow, PyTorch, Keras, and Caffe.
- Element AI. Canada-based Element AI was founded to help enterprises succeed with AI. Industry focus areas include financial services, retail, cybersecurity, transportation, and logistics.

Machine learning

- H2O.ai. The proliferation of ML and DL tools can make the field challenging to navigate, even for experts. H2O.ai's mission is to democratize access to AI. Its H2O platform is an open-source ML platform that simplifies the use of common data science and ML algorithms.

Model validation and evaluation

The most crucial part of the training process for the data scientist is determining whether or not the model has sufficiently learned from the data.

- **Model validation** determines whether a model can scale computationally. Will it crash? Does it have the necessary compute, network, storage, and security resources? If a model validation fails, then an operations engineer needs to revisit the computational requirements of the model.

- **Model evaluation** determines a model's statistical performance. How accurate is the model on live data? How accurate was the model for a subset of data? If the model evaluation fails, then the data scientist has to go back to the design and look at data provenance and model architecture.

You may want to maintain a single dedicated cluster for model validation/evaluation and model serving /deployment, or two separate clusters.

Model serving and deployment

This is the stage in the pipeline where your DL model transitions from development to production. At this point, the process becomes more about lifecycle management and less about data science.

Once the initial training has been completed and the model validated and evaluated to your team's satisfaction, it has to be operationalized. This process often includes:

- Optimizing the model for inferencing performance (using NVIDIA TensorRT for example).
- Serving the model. Depending on your use case, you may deploy your model on inferencing systems at the edge. For instance, autonomous cars have inferencing hardware on board.
- Periodic retraining with the latest data to ensure that the model remains current.
- Integration of model output and custom software.
- Careful management of model versions and training, validation, and testing datasets.

The scale of the activities and requirements often change at this stage. New requirements include:

- **Multitenant environment.** More than one DL model may be in the software pipeline. Access control is needed to make sure that everyone can't access everything.
- **Versioning of datasets.** You need careful version control of all training, validation, and testing datasets.
- **Model management:**
  - Each time the model is retrained, a new version is created.
  - You may need different versions of the same model, optimized for different inferencing hardware.
- **Automation.** These processes should be automated as much as possible to avoid bottlenecks and errors.

Platform options

As with other parts of the pipeline, a number of AI vendors are building platforms that can help you optimize these aspects of the DL pipeline.

- Allegro.ai. NetApp is partnering with Allegro.ai to offer a unified platform that handles all aspects of the dataset lifecycle, eliminating concerns about tooling and infrastructure.
- Datatron offers a platform that focuses on versioning and deployment, monitoring, and management.
- Algorithmia automates DevOps for ML. Models are deployed as scalable microservices, using a serverless model.
- Skymind. The Skymind Intelligence Layer (SKIL) is designed to help enterprise IT teams manage, deploy, and retrain ML models at scale. SKIL can import models from various frameworks.
- Dimensional Mechanics' NeoPulse Framework supports the creation, deployment, and distribution of custom AI models on the premises or in the cloud.

Version history

Version	Date	Document version history
Version 2.0	May 2022	Reformatted and rebranded.
Version 1.0	March 2019	Initial release.

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

WP-7299-0522

NetApp