Introduction to Advanced Information Technology

Bert-Jan Butijn

1 Introduction

Over the years Information Systems (IS) have become increasingly complex and are difficult to grapple. The complexity of recent novel technologies like blockchain (BCT), artificial intelligence (AI) and cloud computing constitutes a genuine challenge to IT-auditors tasked with auditing these IS to provide assurance. Recognizing this challenge this book aims to aid IT-auditors in their audit of such complex IS. This book provides novel insights into these complex IS by demonstrating how control frameworks can be applied to these technologies using several real-life case studies. The chapters that follow hereafter each discuss a different technology.

Each of the aforementioned IS complex, and therefore particularities of the technologies discussed in this book may not be well understood. This chapter discusses the inner-workings, intricacies, and concepts related to these technologies to provide the background necessary to perform an audit using the frameworks presented in the chapters hereafter. In Sect. 2 background is provided about blockchain technology. Section 3 expounds on artificial intelligence, more specifically how it can be perceived and how it is practically used. Similar to the outline of this book, the final technology discussed in the chapter in Sect. 4 is cloud computing. It is strongly recommended to read this chapter before continuing to read the other chapters.

B.-J. Butijn (✉)
Erasmus University Rotterdam, Rotterdam, The Netherlands
e-mail: butijn@ese.eur.nl

E. Berghout et al. (eds.), *Advanced Digital Auditing*, Progress in IS,
https://doi.org/10.1007/978-3-031-11089-4_3

2 Blockchain Technology

The concept of blockchain technology was first published in an anonymous paper by an author called Satoshi Nakamoto in 2008.[1] Blockchain technology incorporates several technologies previously developed for initiatives like Adam Back's Hash Cash (Back, 2002), Digi Cash proposed by David Chaum (1979), and Bit Gold created by Nick Szabo (2005).[2] In 2009, the Bitcoin network was established when the first (genesis) block was mined by Satoshi Nakamoto. Although Bitcoin is often mentioned in the same breath as BCT, there is an important distinction: BCT is the technology that underpins the Bitcoin making it possible to perform transactions without a trusted third party. Bitcoin on the other hand is a cryptocurrency that represents value similar to normal currency that is made possible by the technology.

Since the initial conception of BCT it has gained immense worldwide attention from organizations. BCT has many favorable characteristics and currently many prominent firms like JP Morgan Chase, Maersk, and KLM have started to explore how they can leverage the potential of the technology to their advantage. One of the key features of BCT is that it allows for transactions between parties without requiring a trusted intermediary (e.g., a bank) to safeguard the safety of their transaction. This remarkable feature is made possible by a sophisticated combination of technologies.

BCT is a specific form of distributed ledger technology where the ledger is deployed on a Peer-to-Peer network (P2P). On the P2P network all data about transactions is replicated, shared, and synchronously distributed across multiple peers. Transactions are processed following a strict consensus protocol that is operated by specific nodes to ensure the validity of the transactions requested by other peers in the network, and to synchronize all shared copies of the distributed ledger. During the execution of the consensus protocol, the data of valid transactions, along with other required metadata concerning the network, and the hash of the previous block are bundled by these specific nodes into a block using hashing functions. The essential and key property reflecting BCT architectures is that each block contains the hash of their predecessor, therefore linking all prior transactions to newly appended transactions; the blocks therefore form a chain with the aim of establishing a tamper-proof historical record. This property is depicted in Fig. 1.

As can be noted, BCT is a complex technology that itself encompasses a combination of several other technologies. Let us now further explore how these interrelated technologies interact with one another and constitute to a blockchain system. To exemplify how the technology works we will further discuss the initial BCT underpinning the Bitcoin from the perspective of the trustless transactions it enables. In Sect. 2.1, we first discuss the basic notions of blockchain technology. Then, smart contracts an important concept related to blockchain technology is

[1]Until date nothing is known about the identity or whereabouts of the original author(s).

[2]For further reading about the origins of blockchain technology, we recommend "On the Origins and Variations of Blockchain Technologies" by Sherman et al. (2019).

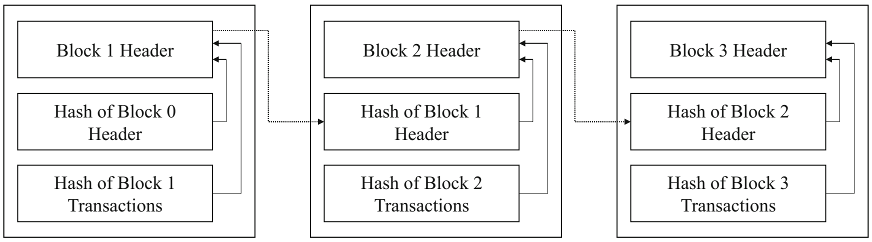


Fig. 1 Graphical depiction of blocks in a blockchain. Note how the combination of the previous block hash and the hash of current transactions form the blockheader

discussed in Sect. 2.2. The last section, Sect. 2.3 presents an overview of a typical blockchain architecture that explains the relation between some over the overarching concepts.

2.1 *Basic Notions of Blockchain Technology*

Owners of a Bitcoin can commit a transaction to the network by digitally signing a hash of the previous transaction and combining it with the public key of the requested recipient. Within a blockchain network public keys are used as the addresses of agents that make use the blockchains' services. The combination of the hash of the previous transaction and public key of the recipient are added to the end of the coin. Therefore, crypto coins can be considered as a chain of digital signatures. This chain of signatures allows anyone to audit and verify the transaction history of a coin. Albeit that the chain of signatures allows anyone to verify ownership claims, this technique does not prevent current owners to double spend a coin. *Double-spending* refers to the act of spending the same coin twice in two different transactions yet at the same time.

One of the unique features of BCT is that it prevents double-spending by introducing a distributed ledger that is shared among peers. Traditional transaction processors like a bank maintain a centrally kept ledger that records all transactions made and especially when they were made. This centralized ledger allows the transaction processor to verify whether transactions have already taken place.

BCT achieves these objectives in a different manner: (1) Transactions are publicly announced to all peers that are part of the P2P network that thereafter record them on their own copy of the distributed ledger. These peers are oftentimes referred to as nodes in blockchain nomenclature. It is important to note that nodes are physical or virtual machines connected to other nodes via a P2P network. Nodes can have one or many human owners, and someone can own several nodes. (2) Because there is no centralized ledger the nodes in the network need to reach a consensus about the history of the transactions on the ledger, and more specifically how to correctly chronologically order them. In principle this approach effectively

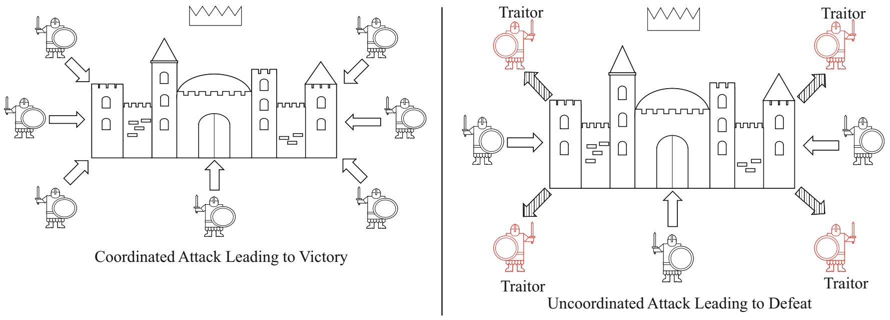

Coordinated Attack Leading to Victory
Traitor
Traitor
Traitor
Traitor
Uncoordinated Attack Leading to Defeat

Fig. 2 The Byzantine Generals Problem. When all nodes behave honest and work together the system works otherwise it will fail

prevents double-spending when all nodes behave honestly. However, not all nodes can be trusted as some might be used to act maliciously and propose incorrect versions of the distributed ledger for their own gain. For instance, by introducing non-valid transactions to increase their own balance. Literature on distributed systems refers to this issue as the *Byzantine Generals Problem* (Lamport et al., 2019). Figure 2 illustrates this problem.

The illustration should be regarded as a metaphor for how distributed systems work: Imagine that there are several generals that have laid siege to Byzantium. The generals must collectively decide when to attack the city. Only when all generals launch their attack simultaneously, they can capture the city. However, if they do not the attack fails. Unfortunately, the generals cannot safely communicate with each other because all messages they will send might be intercepted or deceptively sent by the defenders of Byzantium. This raises the question how the generals can successfully organize their attack simultaneously?

When applying this analogy to blockchain Byzantium is the distributed ledger, and the generals are the nodes within the P2P blockchain network. Similar to the generals in the Byzantines Generals Problem, some nodes will try to manipulate the ledger and thus dismantle its integrity. Honest nodes need a method that enables them to identify transactions on the ledger that are fraudulent or incorrect to keep the distributed ledger free from errors.

To overcome this problem, several safeguards are presented in the original Bitcoin paper (Nakamoto, 2008). One of these safeguards is that transactions are processed in batches by several nodes[3] and are then stored in data structures called *blocks*. Note that each block can only contain a specific amount of data called the *blocksize*, meaning that a limited number of transactions can be included in the block. To create a block, the nodes proceed in the following manner: First, a node checks the validity of a requested transaction. Then, the node uses a timestamp server to timestamp a batch of transactions. Thereafter the node uses the Secure

[3]On some blockchain platforms like Ethereum, the number of nodes that process the transactions can amount up to 10,000.

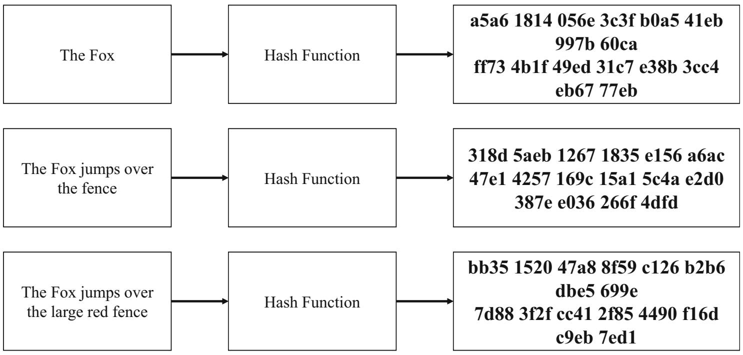

The Fox
Hash Function
a5a6 1814 056e 3c3f b0a5 41eb 997b 60ca ff73 4b1f 49ed 31c7 e38b 3cc4 eb67 77eb
The Fox jumps over the fence
Hash Function
318d 5aeb 1267 1835 e156 a6ac 47e1 4257 169c 15a1 5c4a e2d0 387e e036 266f 4dfd
The Fox jumps over the large red fence
Hash Function
bb35 1520 47a8 8f59 c126 b2b6 dbe5 699e 7d88 3f2f cc41 2f85 4490 f16d c9eb 7ed1

Fig. 3 Example of three texts translated into three unique hash digests. Note how although the length of the text differs the length of the hash is always 64 symbols

Hashing Algorithm 256 (SHA-256) to create a hash of each individual transaction. When given the same input, the SHA-256 algorithm will always return the same output as hash better known as a *digest*. Any small change to the original input however, will render a completely different digest. Figure 3 shows the differences in hashes with two different inputs.

Using the hash, it can effectively be proven that data existed at a certain point in time. More important, any tampering with the hash of a transaction would immediately be recognized as the corrupted hash would not be identical to the one of a correct transaction. Storing the individual hashes of each transaction would require vast amounts of storage space to store the data. Therefore, as a second step nodes bundle the batch of hashes using a *Merkle tree*. An example of a Merkle tree is shown in Fig. 4. In effect this means that the hash of each transaction re-hashed with that of other transactions until only one hash remains.

Another safeguard is proposed in the paper to further guarantee the historical integrity of the distributed ledger. In the hash of a novel block, the hash of the previous block is also included. Effectively this means that the blocks are chained together, and the more blocks are appended to this chain the more difficult it becomes to tamper with the ledger. This solution safeguards the ledger against tampering with the chronology of the transactions by malicious nodes. However, incorrect novel transactions could still be introduced. BCT remedies this problem by demanding that the nodes in the network verify whether (a) any of the newly announced transactions are legit and (b) what the correct version of the distributed ledger is. These activities are integral part of a consensus protocol with the aim of ensuring that the nodes in the P2P network reach a consensus on these aspects. A simple way of reaching consensus would be to allow all nodes to vote. Unfortunately, this would enable malicious nodes to launch a *Sybil attack* by creating an infinite number of duplicates of itself to gather more votes and control the P2P network.

The BCT underpinning the Bitcoin decreases the chance of a sybil attack by employing a Proof-of-Work (PoW) consensus protocol that nodes follow to verify

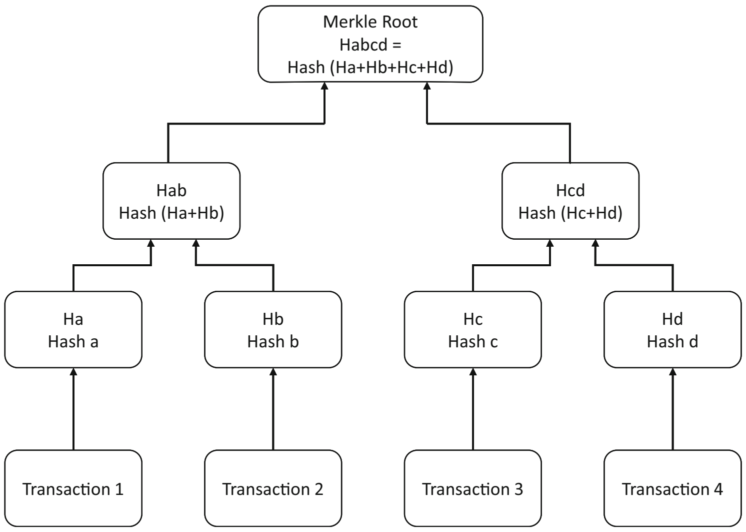


Fig. 4 Graphical depiction of a Merkle tree

transactions. This PoW entails that nodes use their computational power to "vote" on the validity of transactions instead of IP addresses, effectively meaning that the majority of computational power within the network decides. Although it might be easy for someone of ill-intend to amass several IP addresses, obtaining large amount of computational power is likely to be more difficult. Nodes deliver their PoW by solving a computational difficult mathematical puzzle. The first node to solve the puzzle is granted some Bitcoin as a reward. Finding the solution to the puzzle requires finding the right nonce (a random number) that matches the header of the current block, given information of the prior block. The process of finding the right solution to build a block is called *mining*, and nodes that make the effort to solve the puzzle are referred to as *miners*. There is only one miner that can be the first to mine a block. Whenever a node has found the right solution, it propagates the block it constructed to the other nodes. The other nodes then verify the correctness of the block, and if correct append it to their copy of the ledger.

Due to slow propagation of the block among nodes situations might arise where two different miners propagate a block concurrently as they are not aware of the existence of another new block. From that moment on it remains unclear for other miners which of the new blocks is the correct one. In such instances a *fork* in the chain of blocks is created. Figure 5 depicts what a fork looks like from a schematic perspective.

Whenever a fork occurs as a rule, nodes should always trust the longest chain as it represents the branch on which the most computational power has been spend. Nodes that did not propagate the novel block will have to wait until one of the chains becomes longer than the other. Forks are resolved by nodes choosing to adopt the longest chain over the other chain. It is only when the fork is resolved that the

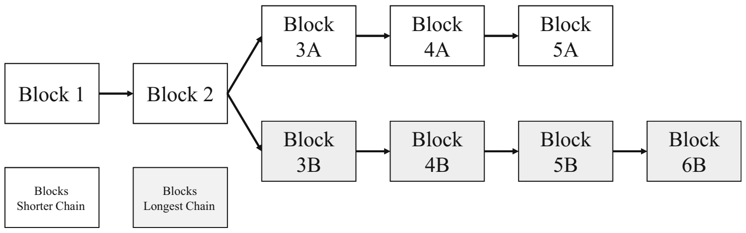


Fig. 5 Graphical example of the longest chain rule. Eventually all nodes will accept the bottom branch as it is the longest of the two

transactions in the new blocks that are part of the longest chain are confirmed. Besides resolving accidental forks, the longest chain rule also protects the integrity of the ledger from malicious users.

2.2 *Smart Contracts*

Initial versions of BCT only allowed their users to make transactions without a trusted intermediary. The desire and potential to employ the technology for uses other than cryptocurrency led to the creation of the Ethereum platform in 2015 by Vitalik Buterin (Buterin et al., 2016). Besides allowing users of the platform to request transactions using the native cryptocurrency called Ether, the Ethereum platform also supports the storage and execution of smart contracts. Smart contracts are computer programs that are stored on the blockchain and contain transaction logic in the form of code. The interesting prospect that this ability offers is that user can stipulate the conditions that have to hold before the transaction is executed (Zheng et al., 2020). Because a smart contract has its own balance and account, they can even hold funds in escrow until these conditions are met. Users can communicate with the smart contract and prompt it to execute some logic. Because these transactions that prompt the smart contract are also stored on the blockchain, a record is created who prompted the smart contract to perform the transaction. If the logic executed by the smart contract involves performing a transaction, this transaction is also recorded (Zheng et al., 2020). The execution of the smart contract and the transactions potentially resulting from this execution are performed by a large number of nodes in the blockchain network. It is therefore important that the execution of the smart contract code always yields the same output when executed by different nodes. If this were not to be the case, the nodes would never be able to reach a consensus on the validity of the transactions resulting from the execution. On public blockchains like Ethereum, a fee is paid for the execution of a smart contract to diminish the chance of abuse and to reward the executing nodes for their efforts (Xu et al., 2017).

Smart contract as a term has been coined by Nick Szabo already in 1994 (Szabo, 1997). However, the concept gained little traction in practice because there was no suitable platform to store the smart contracts or to process transactions resulting from execution of the contract itself. With the rise of blockchain an infrastructure has been provided capable of storing and executing smart contracts while also enabling the processing of transactions resulting from the execution of the smart contract. Because smart contracts are deployed and stored on a blockchain, they inherit some important characteristics from the technology:

- Automatic execution: Smart contracts are in essence coded programs stored on a blockchain. By stipulating conditions with code users control under which circumstances a transaction is executed. It is because of this feat that smart contracts enable the automatic execution of transactions.
- Immutable: Once a smart contract is stored on the blockchain, it cannot be changed. Equally important, a deployed smart contract cannot be removed unless specifically instructed to do so.
- Tamper proof: Because a smart contract is immutable once deployed, no one can tamper with the code in order to influence the outcomes of a transaction process. Because transactions resulting from the execution of the smart contract are verified and performed via the blockchain, these are also tamper proof.
- Self-enforcing: All smart contracts have their own balance. Data concerning this balance is stored on the blockchain. This enables smart contracts to hold funds in *escrow* on their own balance until the predefined conditions are met.

The importance of smart contracts for the further development of BCT cannot be understated. By allowing users to stipulate their own transaction logic, the technology can be used for several applications that go well beyond cryptocurrency transactions. Collectively these applications are referred to as Decentralized Applications or DApps for short. Whereas traditional applications are connected to a database to retrieve information, smart contracts and by extension DApps, are connected to a blockchain from which they can obtain information. As can be noted, a blockchain therefore provides the infrastructure for a smart contract. The addition of smart contracts to the blockchain technology stack has significantly influenced the architecture of blockchain platforms. We will now further dive into the architectures of several blockchain architectures.

2.3 An Overview of Blockchain Architectures

Since the advent of Bitcoin, other blockchain platforms have been established like Ethereum that offer services other than cryptocurrency transactions. As a result, nowadays there are several types of blockchain platforms that can be discerned based on two main characteristics: how access to the network is arranged and whom has what permissions. Table 1 depicts the network arrangements.

Public blockchain platforms like Bitcoin and Ethereum allow for anyone to join the network as a miner or a client. Because anyone is allowed to join the network and subsequently verify and request any transactions, these platforms are also considered

Table 1 Spectrum of blockchain network arrangements

		Accessibility	
		Private	Public
Authorization	Permissioned	Participants in the network need to request access to an administrator to join the network. Each participant is assigned a unique set of rights linked to their digital identity.	In a public permissioned network, anyone can join the network. However, the rights on the network are restricted per participant. For instance, anyone can join the network, but not everyone can read or verify transactions.
	Permissionless	Private and permissionless allow only a group of network participants that have been admitted to the network to perform all actions possible on the network. Participants do have an identity but not a unique set of rights.	Public permissionless networks have as a characteristic that anyone can join the network. When a participant joins the network, they are allowed to read, write, and verify transactions. All data about the transactions (e.g., the blocks) is shared among all willing participants.

to be *permissionless*. It is important to note that the public and permissionless nature of a public blockchain is usually encapsulated in the algorithms that the platform uses to process data among things. Such features are therefore not easily changed.

Unfortunately, the fact that anyone can join the network and perform all possible actions might be considered as inconvenient by some organizations as their control over the platform is diminished. Moreover, public blockchains require complete transparency of the transactions history which is sometimes at odds with the privacy concerns of an organization. Combined, these two factors have led to the introduction of permissioned and *private* and *consortium* blockchains. Proponents of such blockchains advocate that more privacy and access control is needed to guarantee that the blockchain can be used for business. Rather than having one network for all participants, and being owned by all participants private/consortium blockchains are owned by a consortium of organizations or even one organization. Contrary to public blockchains, most private and consortium blockchains have tailor-made distributions of the permissions each participant is granted. Therefore, these types of networks can be considered *permissioned*. Projects like Hyperledger Fabric (Androulaki et al., 2018) provide frameworks to build these consortium/private networks. There are also blockchains that combine features of both architectures.

Blockchain networks provide the technical infrastructure on which several services like smart contracts can be run. As said, ultimately the blockchain infrastructure potentially combined with a smart contract allow for the creation of DApps. Figure 6 depicts a full stack architecture of a DApp that most platforms use.

Working from top to bottom, the first layer is the front end that like for any normal application serves as the *presentation layer* for end users. As blockchain services are normally offered via the internet, the front end is usually a website. Similar like any

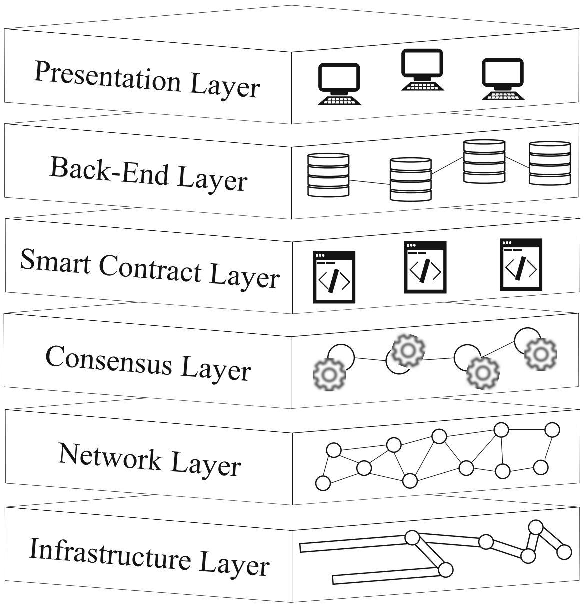


Fig. 6 Full stack architecture of a Decentralized Application (DApp)

normal application, a DApp has a *back-end layer* that processes the programming logic when for instance a user pushes a button. In this case, the back-end usually also sets in motion the actions that a smart contract needs to perform or that needs to be executed on the blockchain. Where a traditional app differs from a DApp is that instead of being connected to a database, a DApp is connected to a blockchain that serves as the point for data storage. Although the back-end is supposed to process the logic within the DApp, it cannot execute any logic used for the blockchain. Executing logic on the blockchain is the purpose of a smart contract that serves as a connector between the users' back-end and the blockchain and forms the *smart contract layer*. This feature is made possible because smart contracts are deployed on the blockchain and users can send transactions to trigger them. Like a normal program a smart contract can be programmed to follow a certain logic when performing transactions. A smart contract could, for instance, store conditions and logic that need to be satisfied before a transaction is executed. Not all blockchain platforms or frameworks cater for smart contracts. As explained in Sect. 2.1, to ensure the validity of transactions and secure the historical record of transactions the nodes in the network need to reach a consensus. The specific set of algorithms deployed to ensure the consensus between the nodes is called the *consensus-layer*. A consensus-layer is the beating heart of the blockchain. Nodes within the blockchain network form a *network-layer* on which the data concerning the blockchain is shared. This data includes the blocks, in other words the data about the transactions but also the code of smart contracts that have been deployed on the blockchain. Communication and distribution of data about the blockchain is shared by the nodes via the *infrastructure-layer*. Nodes are not natural persons but machines or computers that execute the algorithms required for the blockchain. The standard TCP/IP protocol used for everyday communication on the internet provides the channel for nodes to communicate.

3 Artificial Intelligence

AI is nowadays often the subject of conversation within society. The potential to use AI for a wide variety of processes has led organizations to explore how they could harness its potential. Some examples of processes for which AI is employed are fraud detection, marketing, Siri on your phone. Although AI is often referred to as one technology, the term actually represents a broader concept of intelligence demonstrated by machines. The term AI was coined in 1956 by John McCarthy (1995) that describes it as:

> It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

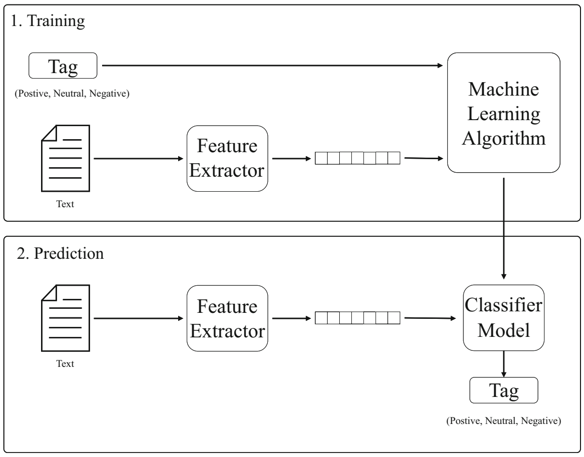
It is important to note that the field of AI focuses on intelligent machines with a strong emphasis on computer programs. Computer programs encompass a combination of *algorithms* that have been designed to *learn* how to perform a specific task, usually by employing statistics. This notion is important because when evaluating how the AI program performs the task at hand the combination of algorithms needs to be examined. What further can be noted from the definition provided by McCarthy is that the aim of AI is to *mimic human intelligence*. With their intelligence humans are capable of performing several tasks. Researchers and practitioners in the field of AI developed several algorithms over the years that have enabled sophisticated programs to mimic the performance of these tasks. Each of these tasks has over time constituted to specialized subfields of AI.

In the remainder of this section, we will first explore how machines learn to perform tasks in Sect. 3.1. To understand how AI is used in practice, in Sect. 3.4 an overview of all subfields of AI will be provided. Each of these subfields will thereafter be explained, and some important concerns for auditing are discussed.

3.1 How Machines Learn

The effort of letting machines learn in order to perform human-like tasks is collectively called machine learning (ML) (Samuel, 1959). Like humans, machines learn by example. When using ML these examples are provided in the form of a machine-readable data set. Each data set encompasses several observations, or measuring points linked to variables. In turn from each observation several features can be

Fig. 7 Separate steps to train a neural network



discerned which are the characteristics or properties of an observation (Bishop, 2006). Relations that the machine has learned are represented as models, that express these relations as parameters, variables, or other mathematical concepts like vectors.

ML algorithms can learn in a descriptive, predictive, or prescriptive manner from a provided data set. These types of learning differ from one another because the aims of the learning process are different. Descriptive learning focuses on extracting relations between features in the data set with the aim of understanding laying bare these relations. For instance, a data set encompassing several customers of a firm can be used to learn how customers are grouped, and on the basis of what characteristics.

Predictive learning is not only aimed at learning relations between features in the data set, but in addition being able to predict what outcome is most likely given a certain input. An example of a task that such an algorithm could learn is to predict the likelihood that a customer will make an insurance claim based on several demographic factors. Similar to descriptive learning, when learning to predict outcomes ML algorithms first examine and learn the relation between features. However, the important difference is that these relations are considered independent variables that serve to predict one or many dependent variables. Getting back to our insurance example, in this case the goal is to predict whether someone will make an insurance claim (dependent variable) based on other independent variables like demographics and so on. The creation of a model to predict outcomes generally takes place in two steps: (1) Training and (2) Prediction (Ashmore et al., 2021). Both steps are depicted in Fig. 7.

An algorithm written with the purpose of training inspects a set of machine-readable observations provided as the input data. These observations serve as examples for the algorithm to determine how the input with certain features is related to certain outcomes. For instance, how demographic factors like postal code, age, and income predict whether or not a customer is likely to make an insurance claim. In some cases, a tag (label) is provided as a target that the algorithm should be able to predict as the dependent variable. The relations between the features and outcomes are then captured in a model. In the next step, called prediction the "fit" of the model is examined. In other words, given a set of provided examples how well does the model predict the expected outcome. Some ML algorithms further improve the fit of the model by using another set of examples to partially retrain the model after an initial training. Again, statistical methods underpin the predictions made using the model.

Prescriptive learning is another approach to ML learning that combines aspects of descriptive and predictive learning with the addition that the algorithm is able to take an informed action based on the data provided. Self-driving cars for instance are not only capable of detecting objects like other cars around them but also to take appropriate action when needed (e.g., hitting the breaks). An important aspect of prescriptive learning is that the algorithm cannot only understand patterns based on prior examples, but can also make informed decisions which action to perform given the information provided.

Besides discerning the algorithms based on its aim, we can also make another distinction between ML algorithms that is related to the manner in which the algorithm is trained or learns from data. The approaches to learn machines are usually divided into three generic categories, based on the nature of stimuli and feedback that is provided to the learning system (Ayodele, 2010):

- *Supervised learning*: For a supervised learning approach the computer is presented by a human with a dataset containing multiple examples with inputs and correct outputs. The main aim of this approach is to learn the algorithm the relations between the inputs and the outputs.
- *Unsupervised learning*: No desired outcomes are provided to the learning algorithm. The algorithm itself has to determine what relations exist in the data set. Note that the discovering these relations or patterns in the data can be the aim itself, or a means towards an end (e.g., to subsequently predict a relation).
- *Reinforced learning*: When employing reinforced learning, a computer interacts in a dynamic environment. In this environment, it must be able to perform a specific task such as driving a vehicle or vacuum clean your house as a robot. While carrying out the task the algorithm is provided with feedback from the environment through which it learns to maximize efficiency. Using this approach, the algorithm learns by trial and error.

What can be noted when closely examining these different types of learning is that they can be discerned based on how and when the input for training is administered. When using unsupervised learning, the builder of the algorithm does not offer any of his own knowledge to the algorithm. In supervised learning, this knowledge is offered by providing the algorithm with examples of the data and classifying (labeling) each example. For instance, providing a set of messages with coherent classification of the sentiment of the message (e.g., angry, happy, or sad). This also introduces hazards however, because what if the provider of the examples made a misjudgment about what sentiment a message, or even several messages actually have. In other words, what if the provider of the examples has provided the wrong examples to the algorithm. Obviously, this would greatly reduce the accuracy of the ML algorithm because it learns from incorrect examples. To diminish the possibility of errors when providing examples for supervised learning, it is desirable to maintain a *four eyes principle*, meaning that at least two or more distinct persons independently label each example provided to the algorithm as input. The distinct sets of independently labeled examples are then compared for agreement. The measurement of the agreement between two raters is called *inter-rater reliability* and serves to provide an indication about the reliability of the labeling of the dataset (LeBreton & Senter, 2008). Several tests like Krippendorf's Alpha, and Cohen's Kappa can be used to measure the inter-rater agreement. However, the process of labeling examples is often arduous and time consuming. Therefore, instead of examining all of the examples provided by another person it is common to only assess a sample.

3.2 Deep Learning and Neural Networks

Oftentimes deep learning is discerned as another subset of machine learning. Like "normal" machine learning deep learning can be employed for descriptive, predictive, and prescriptive purposes and can also be taught to learn using a supervised, unsupervised, or reinforced learning approach. What sets deep learning apart from other machine learning approaches is how the relations between features are stored. Neural networks often consist of many hidden layers to extract and store features from data. In essence, neural networks are data structures modeled to resemble the human brain. Figure 8 depicts a schematic version of a neural network.

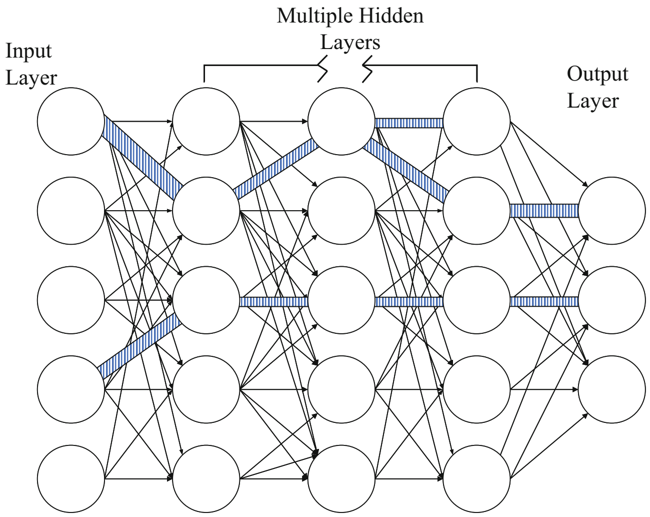


Fig. 8 Schematic depiction of a neural network

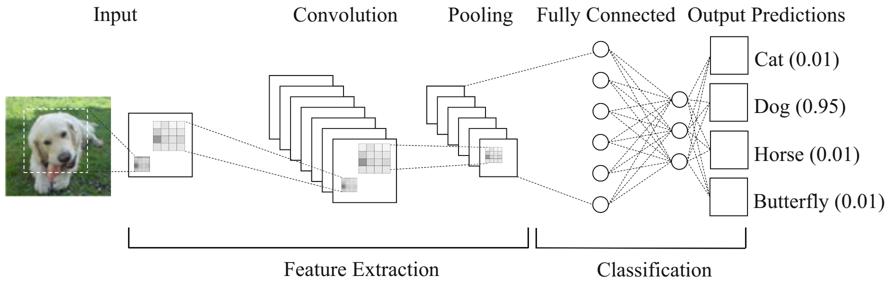


Fig. 9 An architecture for a convolutional neural network

Neural networks are vastly complex multi-layered networks. Similar to a human brain, a neural network encompasses several nodes (similar to a neuron) that are inter-connected which allows data to be passed between them. The neural network always encompasses an input layer and an output layer. In between the input and output layer there are multiple hidden layers. Some neural networks can encompass millions of hidden layers, whereas others only have 20. The hidden layers in a neural network pass on data from the input layer and provide a subsequent outcome to the output layer. Due to the complexity of neural networks it difficult, if not impossible, to understand what happens when data is passed between the nodes. Therefore, neural networks in all of their different shapes and sizes are considered a *black box*, meaning that we know the input and the output of the algorithm but not what happens during the processing of the data. How neural networks are structured strongly depends on the deep learning algorithm used to perform a task. In turn, research (Pouyanfar et al., 2018) has demonstrated that some types of deep learning algorithms are more suitable than others for a specific task. Hence, there is often a strong relation between the task at hand and the type of neural network employed to store the data. Roughly speaking neural networks can be divided into two groups: convolutional neural networks and recurrent neural networks.

Convolutional neural networks are predominantly used for image recognition. Hence, the input to train these neural networks is almost always an image. Figure 9 depicts a typical architecture for convolutional neural networks.

The typical architecture of a recurrent neural network encompasses several layers. However, they perform the two distinct tasks of feature extraction and the classification. In the input layer images are provided to the convolutional neural network model in the form of a matrix. Next, the images are passed on to the convolutional layer that performs the mathematical operations. Each image is then convolved with a separate square matrix that functions as a kernel or filter. The kernel is then slid over each pixel of the image to attain a feature map that contains the information about features of the image such as edges and lines. However, raw feature maps consume vast amounts of memory and are computationally expensive. Therefore, after convolving the image a dedicated pooling layer diminishes the size of the feature map. Several types of formulas like max pooling, average pooling, and sum pooling can be used for this purpose. The last layer or fully connected layer is used to

Table 2 Differences between a CNN and RNN

Convolutional neural network	Recurrent neural network
CNNs are neural networks for deep learning that is predominantly used for image processing.	RNNs are neural networks that are commonly used temporal and sequential data. An important feature of RNNs is that the nodes in the network are sequentially connected allowing for the creation of memory.
CNNs are feed-forward network that require little preprocessing, made possible by multi-layers of nodes.	An RNN can use its internal memory to handle different sequences of input.
Compared to an RNN, a CNN is far more powerful.	RNNs can include and combine far less features compared to a CNN.
A CNN always takes fixed size inputs, and returns fixed size outputs.	An advantage of an RNN is that they can process different sizes of input versus output.

make predictions over the images as they are activated. Although recurrent neural networks are helpful in many aspects, they are not particularly useful to process temporal or sequential data (e.g., a movie).

Recurrent neural networks are better equipped to work with temporal or sequential data. This is largely due to the fact that recurrent neural networks use the input of prior nodes in the network to weigh in their information in order to establish the relation between input and output. Effectively this constitutes an internal memory that is able to distinguish important details such as those related to the input they received. Using its memory, the neural network is able to predict what will come next. This important characteristic of a RNN makes them highly usable for tasks related to speech, video, and text. The key takeaway about RNNs is that when sequence is of the essence, a RNN will learn a far more profound understanding of the sequence as compared to other algorithms.

What sets a RNN apart from a CNN is that the output that has been passed through a prior step is provided as input to the current step. A RNN has therefore two inputs: data concerning the current step and data concerning the recent step(s). This memory build-up is pivotal because the chain of information that is forwarded to each step is what makes that a RNN performs so well on sequential tasks. Contrary to CNNs, the hidden layers of a RNN actively memorizes information about the calculations on the sequential data it has been trained on. Like a CNN the size of a model can vastly increase depending on the task it is trained for. To reduce the complexity and thus size of the model the same parameters are used for each task. The differences between the two types of networks are summarized in Table 2.

3.3 Measuring the Accuracy of Machine Learning Algorithms

When using algorithms to predict or even prescribe certain outcomes, assessing the accuracy of an algorithm is important for auditors. What we mean here by accuracy is how well the model is at doing its task in predicting the right outcome. The accuracy of predictive or prescriptive ML algorithm can be verified by using another distinct set of examples as input and then scoring how many times the algorithm performed the task in line with the right outcome. A dedicated metric to measure the accuracy can then be employed to calculate the accuracy. Because there are several statistical techniques that enable machines to predict, over the years several metrics have been developed to test the efficacy of an ML algorithm. The simplest of these metrics is to measure the precision of an algorithm. *Precision* in this context means how many of all of the observations predicted by the algorithm as positive were actually positive. We can calculate the precision by using the following formula:

$$\text{precision} = \frac{\text{true positives and selected elements}}{\text{Selected elements}}$$

To explain this formula, consider that we have an algorithm that is built to predict whether there is a tree on a picture or a house. In the set of pictures that are provided to the algorithm there are 12 pictures of a tree and 12 with a house, making a total of 24 pictures. The algorithm predicts that in this set of 24 pictures there are 9 pictures that contain a tree. However, in reality of these nine selected pictures there are only four trees on the picture the other five are houses. We call these four correctly predicted pictures with trees *true positives*, while we refer to the total of nine pictures as the *selected elements* as they are predicted by the algorithm.

Another important metric is *recall* also referred to as sensitivity that measures the ratio of correctly identified elements (true positives) among the total of relevant elements in the entire set. Coming back to our example, the relevant elements here are all the pictures with a tree depicted on it (total of 12). We can calculate the recall for this example using the formula:

$$\text{recall} = \frac{\text{true positives and relevant elements}}{\text{relevant elements}}$$

Contrary to the precision metric we use the identified true positives and the total known of relevant elements to calculate. Taking the same example again we would now use the 4 pictures of the tree and divide it by 12. Although at a first glance precision and recall seem appropriate metrics to measure the accuracy of an algorithm, they have some disadvantages. For instance, what if both the precision and recall of an algorithm matter? It is not unreasonable to say that both do and thus to address this problem the $F1$ score was introduced. Using a $F1$ score as a metric is especially popular because it measures the harmonic mean between precision and recall. The formula to calculate the $F1$ score is as follows:

$$F1 = \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Because we already know how to calculate the precision and recall we can simply plug in these calculations into the formula. Where we multiply precision by recall and dividing it by the product of precision and recall. Please note that despite the fact that the $F1$ score is a commonly used metric there is an ongoing debate on the appropriateness of the metric. In the example here above, we only used the $F1$ score to calculate an algorithms performance on two classes. However, an adjusted version of the $F1$ score can also be used for multi-classification testing.

3.4 Using AI in Practice

As mentioned already the main aim of employing AI is to let computers perform tasks otherwise carried out by humans. Over time, a logical division of these tasks has led to the creation of several subfields within the AI domain. In Fig. 10, these subfields are portrayed.

Some of these subfields overlap and this overlap can be attributed to the fact that because these subfields are organized by task, some or almost all of them are one way or another related. Let us now further explore how each of these tasks is performed by AI algorithms.

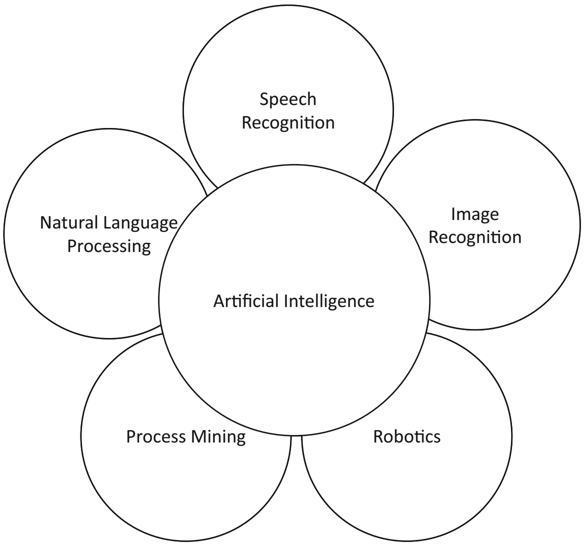


Fig. 10 Representation of subfields in AI. Note that this depiction is not exhaustive and some fields may be missing

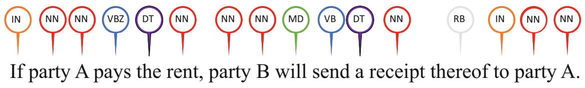
3.4.1 Natural Language Processing

Natural Language Processing (NLP) is a subfield of AI that focuses on developing approaches to enable machines to understand and generate written natural (human) language. The goal of NLP is to create an IS that can sensibly process text to perform a variety of tasks like spell checking, determine the sentiment of a text, or extract relevant information from a text. To understand how these algorithms are able to perform these tasks, we must first understand that computers are ill equipped to perform tasks on text because they are meant to calculate, not interpret. A human reader when presented with a text has learned to discern paragraphs, sentences, words, and letters. Computers however would consider a text merely as a sequence of characters (mostly letters) and without human guidance do not have the capacity to identify sentences or even words. However, paragraphs, words, and other characters often are employed in NLP as features. For a machine to learn relations in texts, whether that be in a descriptive, predictive, or prescriptive manner, these features first have to be created.

Most NLP algorithms therefore require that a certain piece of text is first split into units that serve as an observation. For example, if we want to create an algorithm that is able to predict what sentiment (e.g., angry, happy, or satisfied) a customer review has we take the whole review as the observation. Instead of multiple sentences, single sentences or even single words could also be the unit of observation. This would make it possible for instance to classify a word as being a verb, noun, or other. However, merely dividing a text into observation units usually does not provide enough features for ML to identify meaningful relations. To remedy this problem most ML algorithms for NLP employ a technique called *tokenization* (Webster & Kit, 1992). Tokenization means that an algorithm is employed to divide the set of characters that the text encompasses into a set of strings (like words) that each contain sequence of tokens. At the most generic level the algorithm can predict for a given sentence to what class it belongs. A common application for this is to determine whether customer can be classified as angry, sad, or happy. However, at a more granular level NLP algorithms are able to classify words.

NLP is predominantly used for natural language understanding by analyzing pieces of text for either syntax or semantic meaning. Syntactic analysis involves creating algorithms that are able to dissect the syntax of a sentence, paragraph, or entire text. A well-known task for instance is *Part-of-Speech tagging* (POS) where an algorithm is tasked with syntactically classifying and predicting whether a word is a noun, verb, or coordinating conjunction. An example of what the output of a POS task looks like is depicted in Fig. 11.

Fig. 11 Labels related to words when using POS tagging


IN
NN
NN
VBZ
DT
NN
NN
NN
MD
VB
DT
NN
RB
IN
NN
NN
If party A pays the rent, party B will send a receipt thereof to party A.

Legenda: **IN** = Preposition. **NN** = Noun. **VBZ** = Verb, 3rd person singular present. **DT** = Determiner. **MD** = Modal. **VB** = Verb, base form. **RB** = Adverb.

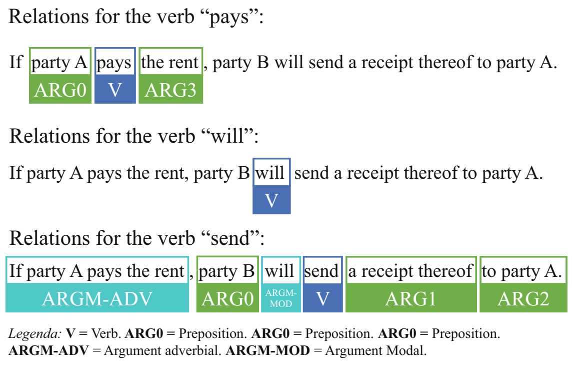

Relations for the verb "pays":
If party A pays the rent, party B will send a receipt thereof to party A.
ARG0
V
ARG3
Relations for the verb "will":
If party A pays the rent, party B will send a receipt thereof to party A.
V
Relations for the verb "send":
If party A pays the rent, party B will send a receipt thereof to party A.
ARGM-ADV
ARG0
ARGM-MOD
V
ARG1
ARG2
Legenda: V = Verb. ARG0 = Preposition. ARG0 = Preposition. ARG0 = Preposition. ARGM-ADV = Argument adverbial. ARGM-MOD = Argument Modal.

Fig. 12 Labels related to different verbs and words when using SRL

Semantic analysis focuses on understanding meaning within a text. This task goes well beyond merely dissecting a sentence by predicting whether words are nouns, verbs, and so on. As the name suggests, the aim of *semantic role labeling* (SRL) is investigating which parts of for instance a sentence play what role. To discern the different roles a part of the sentence has these are connected to verbs. Like POS that is very much akin to SRL a dedicated annotation schema is needed usually also with integrated BIO (Begin Inside Outside) tagging that indicates where a role starts and ends. The most annotation schema used for SRL is that by Palmer et al. (2005). To explain how SRL works take the following sentence: "If party A pay's the rent, party B will send a receipt thereof to party A. The algorithm would first try to predict all of the verbs in the sentence, and then for each of these verbs predict what the relation is between the verb and other parts of the sentence. In the case of our example, this would yield the result depicted in Fig. 12.

The explanation of the labels is omitted here for brevity's sake but further information can be publicly consulted.[4] Taking this notion a step further, practitioners and scholars have started to design algorithms for information extraction. One important part of information extraction is Named Entity Recognition (NER) where NLP algorithms are used to find people, dates, and places in a text. Information extraction also relies on SRL as a basis but an additional algorithm is used a top of a SRL algorithm to give the labels more contextual meaning.

Besides using NLP algorithms to analyze existing text, they are also employed to generate new text. This task is called natural-language generation (NLG) and it serves to produce natural language as output (Reiter & Dale, 1997). The general idea behind NLG is that instead of letting a human author a text, a machine will perform this task. In practice, NLG is used for a tremendous number of applications like (Gatt & Krahmer, 2018):

1. Checking spelling and grammar to suggest text corrections.
2. Generating paraphrases or responses.

[4] Please visit: https://www.cs.rochester.edu/~gildea/palmer-propbank-cl.pdf for a guide of the labels.

3. Translating texts from one language to another.
4. Simplifying complex texts to make them easier to read for a broader audience.
5. Text summarization to automatically create abstracts from long texts.

Defining the difference between natural language understanding and NLG is oftentimes hard (Gatt & Krahmer, 2018). In practice, these aspects are combined to attain a certain result. To create a chatbot for instance, SRL is employed to make the machine understand what a customer is asking. Then NLG is used to formulate an answer to the customer's question.

3.4.2 Speech Recognition

Speech recognition once was considered a subfield of NLP. However, recently it has developed into a full-fledged interdisciplinary subfield of computational linguistics. The aim of speech recognition is to develop methodologies and coherent algorithms that enable computers to recognize and translate spoken language into text, or machine-readable format. A prime example of speech recognition usage is Apple's Siri, or the Alexa home appliance from Amazon. Both use a sophisticated speech recognition algorithm to capture and process spoken language with the aim to understand what a user is commanding them. These interpretations then prompt the program to execute whatever the user is asking. *Voice recognition* can be considered another aspect of speech recognition. Algorithms for voice recognition are not designed to understand a users' commands but recognizing different users. Again, like all ML techniques speech recognition algorithms learn from features, in this case the audio provided to train the algorithm. Compared to NLP speech recognition uses several different features:

1. Language weighting: When mentions of words are of interest, the algorithm can be trained to listen to a particular set of words. Training the algorithm to specifically identify these words increases the chance of filtering out conversations or audio of interest based on subject.
2. Acoustic training: Inevitably with some audio there is ambient sound or other noise pollution. Acoustic training serves to aid the algorithm to discern for instance background noise and speak.
3. Speaker labeling: For voice recognition, speaker labeling is important to understand who is speaking, and by extension who is saying what in a conversation. Algorithms trained on this aspect are able to discern several speakers at once and translate their contribution.
4. Profanity filtering: The use of profanity filtering is to detect specific words in a conversation to filter them out or extract them. This feature differs from language weighting as it is designed as a filter not to identify conversations of interest among for instance a set of audio fragments.

Like other ML architectures, speech recognition algorithms are made up out of several components. First, there is the speech input that consists out of multiple audio fragments. Based on these audio fragments, a model is created containing several feature vectors that capture a myriad of relations between features like tone and length of a tone. When using the model in practice, a decoder is required to interpret the outcomes that have been attained through the use of the features. The decoder itself employs pronunciation dictionaries, language models, and one or more acoustic models to attain the right output.

Although great improvements have been made to speech recognition algorithms, the current best score was made by Google Cloud Speech in 2017. Their algorithm yielded a score of 95% with an error rate of 5%. In itself this error rate possesses no problem to the use of the algorithm. However, speech recognition algorithms are often used in combination with an NLP algorithm that also has an error rate compounding the errors in the final output. Consider the example of a speech assistant of Google Home, Amazon's Alexa, or Microsoft's Cortana; These systems employ speech recognition algorithms to understand when a command is given to them by whom and translate this to text with a potential error rate of 5%. In a sequential step, an SRL algorithm (NLP) uses the text created as output by the speech recognition algorithm (with the errors) as input to determine what the user has commanded. The SRL algorithm thus receives input with errors that in turn is far more prone to generate wrong output, not even taking into account the error rate of the SRL algorithm itself.

3.4.3 Image Recognition

Image recognition is a strand of AI methods that focuses on classifying images. The applications for these algorithms are around us everywhere. A prominent example is the face recognition on most smart phones. Image recognition is also used for self-driving cars that need to recognize obstacles on the road, or find persons of interest on camera footage. An image recognition algorithm can perform several tasks:

1. Classification: This task involves classifying that what "class" the image belongs. For instance, the depiction of a dog or a cat.
2. Tagging: Is a task similar to classification, but is more fine-grained. The tagging task involves identifying (potentially) multiple concepts and/or objects in an image. For one image several tags or labels can be appropriate.
3. Detection: When the algorithm is assigned to identify and locate an object in an image (or video), it is a detection task. An example for the use of such an algorithm is software for self-driving cars.
4. Segmentation: This task is similar to detection however, but is yet again somewhat more fine-grained. The algorithm is able to locate objects on a pixel level which is sometimes required for very precise identification.

As explained in Sect. 3.2, CNNs are predominantly used for image recognition. When training ML algorithms for image recognition, the trained weights and biases are assigned to several parts of the image that serve as the features so that they become indistinguishable from one another. Based on the knowledge on these

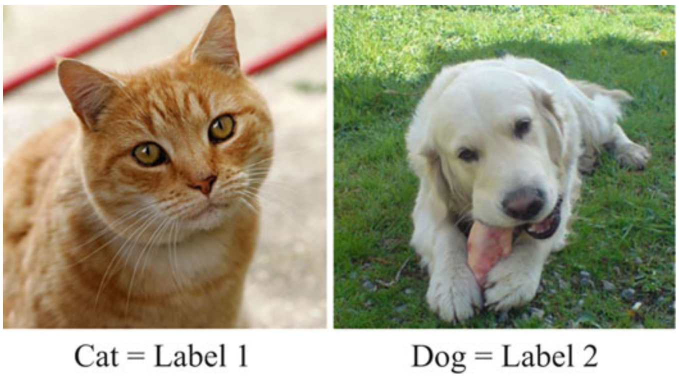


Fig. 13 Classification task input per image

distinctions, the recurrent neural network can be activated for several tasks like image recognition, object and face detection, and image recognition using a set of activation functions. The training examples (i.e., images) can be both labeled for supervised learning and unlabeled for unsupervised learning. The algorithm regards each input image as an array of pixels translated as a matrix. This matrix usually pertains data in the form of Height × Width × Dimension. To illustrate how this works, consider an image of 20 pixels × 15 pixels × 1 where the 1 denotes the RGB color. The range of the numbers that are stored in the matrix is referred to as the color depth. Hence, the color range strongly dictates the maximum number of colors that can be used. For RGB colors that are a mixture of red, green, and blue often used in images, this range is from 0 to 255. After converting the matrices to a plethora (sometimes millions) of features, labels can be added to the images to train the model.

In Fig. 13, a training example is shown for an image recognition algorithm that is trained perform a classification task. Figure 14 depicts how training images are labeled for image recognition algorithms that carry out detection and segmentation tasks.

Whereas in Fig. 13 the entire image is labeled, in Fig. 14 the objects in the image are "boxed" with a red line. Unless programmed to do so, an image recognition algorithm does not provide a "boxed" picture as output but only a tag (if any).

3.4.4 Process Mining

The purpose of process mining is to discover a process in the context of an organization and potentially make predictions or prescriptions about how the process takes place. Although in the past most processes were carried out by hand, nowadays most activities within an organization are performed using a computer. The fact that most processes are now carried out using a computer makes process mining easier and more predictable. When carrying out actions via a computer, a log of activities is

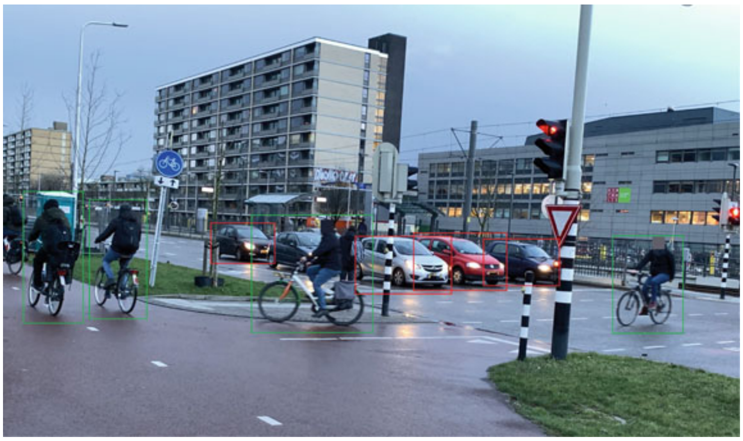


Fig. 14 Image recognition task with different classes of labels

created containing all data related to the sequence of activities a user has carried out. Specialized process mining algorithms can be used to mine a process from a set of logs. These algorithms discern several activities from each individual activity log to identify sequences that are shared across all provided audit logs. There are several statistical techniques available for this purpose but the most popular is clustering that is used for description of a process. AI and machine learning can further help in process description by detecting anomalies and finding processes that are similar based on an example.

When a process has already been discovered and laid out, *diagnosis* might also be useful. Consider for example the case where a process is known to be performed sub-optimal. A sub-optimal performance of a process might have many causes. ML can be harnessed to find the causes of a problem by reasoning back and generating a root-cause analysis of the problem. If any problems have been identified during the execution of a process, ML can also be employed to classify these problems. In turn, the classification of the problems makes it easier to remedy them. The evaluation of the changes that may have occurred to the process over time might also be mapped using ML to spot trends.

Knowing how processes have been carried out in the past is useful to prevent errors, delay, and other problems in the future. Machine learning based process mining is also employed in practice to monitor ongoing processes and make *predictions* about the next event that will occur, how the process will influence certain outcomes, or even the final outcome of a process. Similar to other AI applications, once a machine has learned how to predict events or outcomes it can act *prescriptive*. For instance, when a process will occur during a process it can send an email to the appropriate person to notify them of the problem. Or, when the AI is advanced enough, activate robots or programs to solve the problem.

3.4.5 Robotics

Robots are perhaps the most classical picture that we have in our minds when we imagine AI. However, robots do not necessarily possess AI to perform their tasks. The word robot was first introduced by Karel Čapek in 1920 with the connotation that we know it for today. In Czech the word “robota” means “labor” or “compulsory labor.” A robot is a machine that performs physical labor in the form of one or many steps, usually in a specific sequence. In the car industry for instance welding robots are now commonplace to perform welding tasks. Robots have the following characteristics:

1. All robots are composed of a material mechanical construction that allows it to interact with its physical space and to manipulate it. This construction can include several sensors to perceive the environment, and mechanical instruments to perform actions.
2. Robots need a power supply to function and feed their mechanics with power. Not all robots use electricity for this purpose, e.g., steam is another supply of power that can be used.
3. At least some sort of algorithm is needed to instruct the robot what to do and how. The absence of the computer program would mean that the robot is a piece of simple machinery. Because the program instructs the robot what to do it is able to operate in the physical space.

At first glance, robots and computer programs seem alike and even interchangeable. An important difference between robots and computer programs is that a program does not carry out a task in physical space but rather only virtually (Luckcuck et al., 2019). The physical activities are made possible because robots are a combination of software and hardware components. Some of the software used for robots is simply an algorithm that always performs the exact same steps, or is programmed by a user to follow a sequence of different steps. More advanced robotics that employ AI to determine the sequence of steps they need to take are called *autonomous robotic systems* (Luckcuck et al., 2019). Like most applications of AI, autonomous robotic systems touch upon ethical and legal considerations, however compared to other AI applications safety is an even more import aspect. As autonomous AI based systems can manipulate their environment, they can physically harm their environment, including humans.

To prevent unsafe situations, the physical environment of the robot can be modeled. Two approaches are predominantly used for this purpose: the workplace of the robot is modeled or the environment itself is continuously monitored. The first approach has been proven to be extremely difficult in dynamic environment where all potential future circumstances that may lead to unsafe situations need to be captured. Continuously monitoring the robot leads to similar problems in that unsafe situations need to be known in advance in order for them to be prevented. Providing trust and required certification evidence is challenging for auditors (Luckcuck et al., 2019). Formal methods are a commonplace to ensure the correctness and safety of (software) systems, and thus to provide trust and certification evidence. However, hitherto there is not one uniform widely accepted formal method that has been adopted for the development of autonomous robotics. Thus, developers are provided with few guidelines to select the appropriate formal method to build and verify an autonomous robot (Kossak & Mashkoor, 2016). More important, the technology for autonomous robotics is still in its infancy. Consequently, regulations on the topic are still being developed, making it difficult for certification bodies to establish criteria for an audit (Webster et al., 2014). Besides these safety concerns, another notable problem is how to coordinate swarms (several) of autonomous robots that have to operate in concert to attain a goal. Because these swarms magnify the pre-existing problems with autonomous robotics while adding a coordination problem. The introduction of machine learning enhances the complexity of autonomous robotic systems even further by obfuscating how the robot has made its decisions making it hard to monitor.

4 Cloud Computing

Cloud computing is a term used for computing services delivered via the internet. These services encompass a broad array of computing resources that are nowadays offered by hundreds of providers like Amazon, Google, Microsoft, IBM, and VMware. Although the term is often used, it is often ill defined. The US National Institute of Standards and Technologies (NIST) provides a broadly accepted and concise description of the generic properties of cloud computing (Mell & Grance, 2011):

- On-demand Self Services: Any client is able to procure computing resources without any human interaction.
- Broadly Accessible: Standard mechanisms and protocols enable the access to the cloud computing resources.
- Pooled Resources: A cloud computing service provider has a pool of computing resources that are allocated and provided to clients on demand.
- Rapid Elasticity: Computing resources can easily be provided, scaled up and down based on the clients' requirements and demands.
- Measured Service: A cloud computing system charges a client based on the resources used. To enable this feat the system must be able to automatically monitor, control, and report to the client how much of the resources have been used.

Hereafter the components of a cloud computing architecture are further explained in Sect. 4.1. The ecosystem of cloud computing is further described in Sect. 4.2.

4.1 Cloud Computing Architecture

To support cloud computing most cloud computing providers employ a three-layered architecture. The architecture encompasses a service layer, resource abstraction and control layer, and a physical resource layer. This architecture is depicted in Fig. 15.

According to the US NIST definition of cloud computing, the *service layer* of the services a cloud provider provides typically encompasses three components (Liu et al., 2011):

1. Infrastructure as a Service (IaaS)
2. Software as a Service (SaaS)
3. Platform as a Service (PaaS)

The *IaaS* component provides the computing resources to the client. These resources include virtual machines (VMs), data storage, connected networks, and other utilities through a service model. The promise and premise of cloud computing is founded on the hardware the cloud provider provides. *SaaS* is the next component in a cloud computing service layer, that refers to the delivery of an application. These applications are delivered via the network (infrastructure) to users of the applications. Users of SaaS divided into several groups: organizations that give access to the software applications, the administrators charged with the configuration of the software application, and end users. For each cloud computing provider, there are several manners to calculate the costs of deploying an application. Some cloud providers charge based on the number of end users that use the application, others on the time used, volume of the data stored or its duration. The *PaaS* component binds the other two layers together and provides a platform for the client of a cloud provider to use tools and other resources to develop, test, and deploy their applications. During the life span of the application, the PaaS layer also enables the management of the hosted application. Among the users of the PaaS are application developers, testers, owners, and administrators.

Between the service layer and the physical resource layer is the *abstraction and control layer*. The abstraction and control layer are employed by cloud providers to

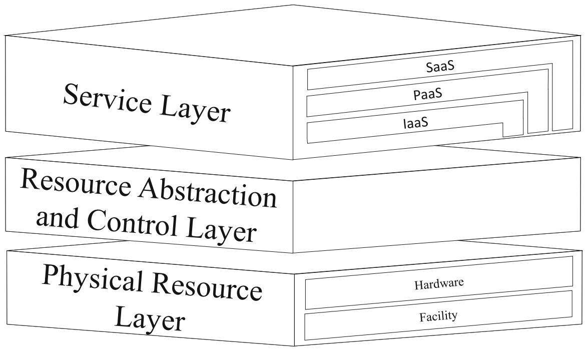


Fig. 15 Reference architecture cloud computing

control the accessibility to physical cloud computing resources. To enable the management and controlling of the physical resources, software abstraction in the form of hypervisors, virtual machines, and virtual data storage is utilized. The layer is especially important because it is used to allocate computing resources to clients, control access, and compute costs. The bottom most layer is the *physical resource layer*. All physical computing resources like CPUs, memory, networks, and data storage facilities are part of this layer. Most physical computing resources also need a plant with their own resources where they are installed. These plants and coherent facilities are also part of the physical layer.

4.2 Cloud Computing Ecosystem

A cloud computing ecosystem encompasses several actors. There are cloud clients, providers of the cloud, cloud carriers, brokers of the cloud and auditors. Within the ecosystem each of these actors are entities that are involved in the processes that take place using the cloud. *Clients of the cloud* provider make use of the services a cloud provider provides. Cloud providers usually offer a catalogue of the services they provide from which the cloud client can make selection. After selecting the desired services, the clients of the cloud provider can immediately make use of it. The services provided by a cloud provider are not for free and a service agreement for payments must be made. Not only the payment terms for the services are important. Clients of a cloud provider might have specific technical requirements for the services they consume. *Service Level Agreements* (SLAs) are commonly used to stipulate the technical requirements and performance the client and provider have agreed upon. These technical requirements may include specific details on the level of security, quality of the services, and potential remedies when the service fails to deliver.

Cloud providers are entities tasked with guaranteeing the availability of cloud services to interested clients. In effect this task includes providing the required infrastructure, managing and running the clouds' software, and providing access to clients of the cloud via a network. Maintenance is another task of the cloud provider that involves servicing any software and updating databases used by the clients. Because the clients develop the software applications themselves, cloud providers often offer several development and management tools for their platform. Some examples of these tools are integrated development environments (IDEs) and software development kits (SDKs). These tools aid the clients in developing and deploying their application on the platform of a cloud provider. Although clients can deploy and control their application via the provider, they have no control over the operating system and other aspects of the platform.

Recently cloud computing has become very complex and this makes it hard for cloud clients to manage their consumed services. This need is addressed by *cloud brokers* that indirectly offer the services of a cloud provider. Cloud brokers provide *service intermediation* by enhancing a service that is originally provided by a cloud provider to enable some additional capability. Sometimes cloud brokers *aggregate services* from several cloud providers into one main service. Service arbitrage is akin to aggregating services with the difference being that the arrangement of the services is flexible.

5 Conclusions

The complexity of recent novel technologies like blockchain, AI, and cloud computing constitute a genuine challenge to IT-auditors tasked with auditing these IS to provide assurance. The first step towards a clear understanding of how these complex technologies can be audited is to understand the technology itself. This chapter provides the basis of such understanding. Blockchain technology is a complex technology because many sophisticated technologies are combined to create one IS that is able to process transactions without a trusted intermediary like a bank. Smart contracts add more complexity and potential to the technology by allowing for conditional transaction logic. Combined, this constitutes to a unique technology stack.

The term artificial technology is often used for a wide variety of algorithms with different tasks. In this chapter we discussed that the type of algorithm employed and how it learns to perform its task determines how to investigate a particular AI algorithm. To provide a broader perspective, we explain some of the fields for which AI is employed. This overview clearly shows that the term AI should be nuanced in terms of the algorithms discussed, and the task at hand.

Cloud computing is another complex technology that is explained in this chapter. A key takeaway from this discussion is that the term cloud computing is not always concisely used. The definition suggested by the NIST provides clarity by stating the properties of cloud computing. Cloud computing has a three-layered technology stack, that generally speaking provides three types of services to its clients. Nowadays a comprehensive ecosystem has developed around cloud computing. Within this ecosystem there are several actors that fulfill their own role.

References

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., ... Yellick, J. (2018, April). Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (pp. 1–15).

Ashmore, R., Calinescu, R., & Paterson, C. (2021). Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR), 54*(5), 1–39.

Ayodele, T. O. (2010). Types of machine learning algorithms. *New Advances in Machine Learning, 3*, 19–48.

Back, A. (2002, augustus 1). *Hashcash: A denial of service counter-measure*. Hashcash. Retrieved from http://www.hashcash.org/papers/hashcash.pdf

Bishop, C. M. (2006). Pattern recognition and machine learning, 128(9) Springer

Buterin, V., Wood, G., & Wilcke, J. (2016). *Ethereum homestead documentation*. Ethereum Community. Retrieved from https://ethdocs.org/en/latest/

Chaum, D. L. (1979). *Computer systems established, maintained and trusted by mutually suspicious groups*. Electronics Research Laboratory, University of California.

Gatt, A., & Krahmer, E. (2018). Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research, 61*, 65–170.

Kossak, F., & Mashkoor, A. (2016, May). How to select the suitable formal method for an industrial application: A survey. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z* (pp. 213–228). Springer.

Lamport, L., Shostak, R., & Pease, M. (2019). The Byzantine generals problem. In *Concurrency: The works of Leslie Lamport* (pp. 203–226). Association for Computing Machinery.

LeBreton, J. M., & Senter, J. L. (2008). Answers to 20 questions about interrater reliability and interrater agreement. *Organizational Research Methods, 11*(4), 815–852.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2011). NIST cloud computing reference architecture. *NIST Special Publication, 500*(2011), 1–28.

Luckcuck, M., Farrell, M., Dennis, L. A., Dixon, C., & Fisher, M. (2019). Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR), 52*(5), 1–41.

McCarthy, J. (1995). What is artificial intelligence? *Annali di Matematica Pura ed Applicata., 169*, 321–354.

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing*. National Institute of Standards and Technology.

Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system* (Decentralized Business Review, 21260). Satoshi Nakamoto Institute.

Palmer, M., Gildea, D., & Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics, 31*(1), 71–106.

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M.-L., Chen, S.-C., & Iyengar, S. S. (2018). A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR), 51*(5), 1–36.

Reiter, E., & Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering, 3*(1), 57–87.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 210–229.

Sherman, A. T., Javani, F., Zhang, H., & Golaszewski, E. (2019). On the origins and variations of blockchain technologies. *IEEE Security & Privacy, 17*(1), 72–77. https://doi.org/10.1109/MSEC.2019.2893730

Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*.

Szabo, N. (2005). *Bit Gold*. Nakamoto Institute. Retrieved from https://nakamotoinstitute.org/bit-gold/

Webster, J. J., & Kit, C. (1992). Tokenization as the initial phase in NLP. In *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*.

Webster, M., Cameron, N., Fisher, M., & Jump, M. (2014). Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. *Journal of Aerospace Information Systems, 11*(5), 258–279.

Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., & Rimba, P. (2017, April). A taxonomy of blockchain-based systems for architecture design. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 243–252). IEEE.

Zheng, Z., Xie, S., Dai, H. N., Chen, W., Chen, X., Weng, J., & Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems, 105*, 475–491.