Explainable Artificial Intelligence: Exploring XAI Techniques in Military Deep Learning Applications

LINUS J. LUOTSINEN, DANIEL OSKARSSON,
PETER SVENMARCK, ULRIKA WICKENBERG BOLIN

FOI-R--4849--SE
ISSN 1650-1942 December 2019

FOI-R--4849--SE

Linus J. Luotsinen, Daniel Oskarsson,
Peter Svenmarck, Ulrika Wickenberg Bolin

Explainable Artificial Intelligence: Exploring XAI Techniques in Military Deep Learning Applications

Bild/Cover: Eevamaria Raudaskoski, @eevagraphics

Titel	Förklarande Artificiell Intelligens: Översikt av XAI-Teknik för Militära Djupinlärningstillämpningar
Title	Explainable Artificial Intelligence: Exploring XAI Techniques in Military Deep Learning Applications
Report no	FOI-R--4849--SE
Month	December
Year	2019
Pages	54
ISSN	1650-1942
Customer	Swedish Armed Forces
FOI Research area	C3 and Human Factors
Armed Forces R&T area	Command and Control
Project no	E60916
Approved by	Cecilia Dahlgren
Division	Defence and Security, Systems and Technology
Export control	The content has been reviewed and does not contain information which is subject to Swedish export control.

Abstract

As a result of the advancements in artificial intelligence (AI), machine learning and specifically deep learning, the explainable artificial intelligence (XAI) research field has received a lot of attention recently. XAI is a research field where the focus is on ensuring that the reasoning and decision making of AI systems can be explained to human users. In a military context, such explanations are typically required to ensure that:

- human users have appropriate *mental models* of the AI systems they operate,
- specialists can *gain insight and extract knowledge* from AI systems and their hidden tactical and strategical behavior,
- AI systems *obey international and national law*,
- developers are able to *identify flaws or bugs* in AI systems even prior to deployment.

The objective of this report is to explore XAI techniques developed specifically to provide explanation in deep learning based AI systems. Such systems are inherently difficult to explain because the processes that they model are often too complex to model using interpretable alternatives.

Even though the deep learning XAI field is still in its infancy, many explanation techniques have already been proposed in the scientific literature. Today's XAI techniques are useful primarily for development purposes (i.e. to identify bugs). More research is needed to conclude if these techniques are also useful for supporting users in the process of building appropriate mental models of the AI-systems they operate, tactics development and to ensure that future military AI systems are following national and international law.

Keywords

Artificial intelligence, explainable AI, transparency, machine learning, deep learning, deep neural networks

Sammanfattning

Förklarbar artificiell intelligens (eng. explainable artificial intelligence) eller XAI är ett forskningsområde som har sett en stor tillväxt under de senaste åren. Detta är ett resultat av den snabba utveckling som skett inom artificiell intelligens (AI), maskininlärning och framförallt djupinlärning. Inom XAI bedrivs forskning som syftar till att förklara AI-systemens resonemang och beslutsfattande för mänskliga användare av systemen. I en militär kontext är förklarbarhet hos AI-system nödvändig för att säkerställa att:

- den militära slutanvändaren har en lämplig *mental modell* av hur systemet fungerar,
- specialister *kan skaffa insikt och extrahera kunskap* från AI-systemens, ofta dolda, taktiska och strategiska beteende,
- AI-systemen *följer internationell och nationell lag*,
- utvecklare kan *identifiera fel* innan AI-systemet sätts i produktion.

Syftet med denna rapport är att presentera XAI-tekniker som har utvecklats för ökad förklarbarhet i AI-system implementerade med djupinlärning. Modeller som bygger på djupinlärning är "svarta lådor" som har hög kapacitet och kan modellera komplexa processer som är svåra att modellera med alternativa, mer förklarbara, angreppssätt.

Även om XAI för djupinlärning är ett relativt nytt område så har flertalet förklarbarhetstekniker föreslagits i den vetenskapliga litteraturen. Idag är dessa tekniker främst utvecklade till stöd för modellutvecklare (d.v.s. för att identifiera fel och avvikelser). Mer forskning krävs för att utvärdera om och hur dessa tekniker också kan stödja skapandet av lämpliga mentala modeller hos slutanvändaren, vid taktikutveckling, samt för att säkerställa att AI-systemen följer lagar och förordningar.

Nyckelord

Artificiell intelligens, förklarbar AI, transparens, maskininlärning, djupinlärning, djupa neuronnät

Contents

1 Introduction

Artificial intelligence (AI) is a research field that is of strategic importance to Sweden [1] and the Swedish Armed Forces (SwAF). The main contributing factor to the success of today's AI are breakthroughs within machine learning (ML) and, more specifically, deep learning (DL). DL is a potentially disruptive technology that allows us to use deep neural networks (DNNs) to model processes that were previously too complex to model using traditional techniques. For instance, DL can be used to accurately transcribe (speech-to-text) [2, 3], translate (text-to-text) [4], synthesize speech (text-to-speech) [5], play real-time strategy games (video-to-action) [6, 7], read lips (video-to-text) [8], identify faces (image-to-identity) [9] and control self-driving vehicles (video-to-action) [10, 11].

However, DL is still in its infancy and there is no mathematical framework that can be used to guarantee model correctness [12]. Hence, there are many challenges that need to be considered and addressed when developing, deploying, using and maintaining DNN models in military applications.

Perhaps the most important challenge from a military user's perspective (operator, data analyst, etc.) is *explainability*. As a rule of thumb, the need for explainability is greater when human lives are deeply affected. This is true in the military domain but also in medicine, law enforcement and other civilian services. Explainability is important because it influences the users' trust and reliance in the system. The trust relationship must be balanced; too much trust may result in a misuse of the system whereas too little trust may result in a complete disuse of the system [13]. Ultimately, explanations aim to help users build an appropriate mental model of the system to ensure that it can be used efficiently [14].

Deep learning has the potential to improve autonomy in complex military systems such as fighter jets, submarines, drones and satellite surveillance systems. However, it would also make these systems even more complex and difficult to explain. The main reason is that DL is an end-to-end machine learning technique, meaning that the machine learns to extract the features from the input data that are the most important ones to achieve high performance. This is known as representation learning and it differs from traditional techniques where human intuition is used to manually extract such features. Representation learning often results in high performance, but it also requires the model to be highly expressive and nonlinear. DNNs trained using DL may therefore consist of millions or even billions of parameters. This makes them difficult to interpret and explain to humans, even though learning algorithms, model architecture, training data, etc. are known and well understood.

The explainable AI (XAI) program that was initiated in 2016 by the United States Defense Advanced Research Projects Agency (DARPA) is perhaps the most comprehensive military initiative taken towards addressing this challenge. The aim of this program is to [15]:

- "Produce more explainable models, while maintaining a high level of learning performance (prediction accuracy)."
- "Enable human users to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners."

Many technical advancements have been made since the start of the XAI program. Some XAI techniques have even been implemented and packaged in software libraries that can be used to gain insight, debug and verify DNNs [16, 17, 18]. This is a step in the right direction, but from a military perspective it is critical that XAI techniques and tools are also tailored for military users where aggregated, high-level explanations are needed to ensure trust, use and performance.

1.1 Purpose and scope

The objective of this report is to present representative XAI techniques that have been developed in the context of DL. The report is not exhaustive and it does not cover all XAI techniques proposed in the literature.

1.2 Target readership

The target readership of this report is personnel that operates, acquires or develops military systems where AI, ML and DL technologies are used by or embedded in the system.

1.3 Outline

Chapter 2 introduces the concepts of intelligent agents, machine learning and deep learning. Chapter 3 introduces a variety of XAI techniques proposed in the literature. Chapter 4 introduces methods and techniques that can be used to evaluate the explanations provided by XAI techniques. Chapter 5 presents a case study where XAI was used to explain the behavior of a deep learning model. Finally, Chapter 6 concludes the report and provide recommendations for future work.

2 Intelligent agents, machine learning and deep learning

This chapter introduces concepts, methods, terms and techniques that are useful when reading the remainder of this report. Readers that already have a basic understanding of intelligent agents, machine learning and deep learning can skip this chapter.

2.1 Intelligent agents

AI is a broad term that can be defined in many ways. In this report, AI is used in terms of *the study and design of intelligent agents* (IAs). An IA is an autonomous entity capable of sensing, reasoning and acting in an environment. Typically, IAs interact with other agents (i.e. multi-agent system) as well as humans (e.g. human-machine teaming) in the environment.

When implemented in the physical world, IAs may represent anything from simple thermostats to complex self-driving vehicles, autonomous robots, drones, etc. In virtual environments, IAs are typically represented by bots or virtual assistants capable of translating, transcribing, and so on. In military simulation, IAs are often referred to as non-player characters (NPCs) or computer generated forces (CGFs).

Figure 2.1 illustrates the main components of an IA. These components are typically implemented using a combination of traditional programming and AI techniques such as expert systems, state machines, behavior trees and machine learning. This report focuses on XAI for IAs that are fully or partially implemented using DNNs.

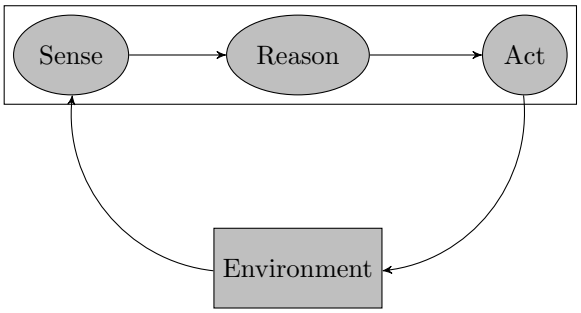


Figure 2.1 – An intelligent agent (IA) is an autonomous entity that is able to sense, reason and act in an environment. The environment can be physical (i.e. real world) or virtual (e.g. the internet, virtual simulation, serious games). IAs typically interact with other agents and humans to form multi-agent systems and human-machine teams respectively.

2.2 Machine learning

ML is a subfield of AI where the focus is on developing intelligent systems or IAs that can learn from observations and experience. In this section, the main learning strategies used in ML are introduced.

2.2.1 Supervised learning

In supervised learning, the IA learns from training examples that have been tagged or labeled. The learning objective is to minimize the deviation from these examples while also maintaining the ability to generalize to unseen inputs. In effect, the IA will imitate the behavior presented in the training data. The supervised learning process is illustrated in Figure 2.2.

In supervised learning, the labeling process is often manually performed by humans, which is why the approach can be expensive and impractical in many applications. The main advantage of supervised learning is that the learning process, once the dataset has been created, is stable and relatively easy to monitor.

The main applications of supervised learning are classification and regression where, discrete class labels and continuous values represent the output of the model respectively. Classifiers can be used to detect objects of interest in the agent's field-of-view or to recognize if a particular situation is hazardous or not. Regression is typically used for the low-level continuous control of the agent's actuators (robotic limbs, steering wheel position, etc.)

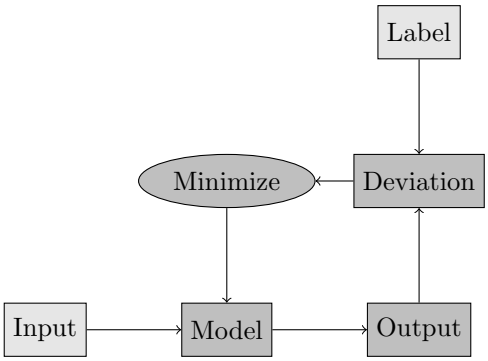


Figure 2.2 – Supervised learning. The IA learns from examples that have been tagged or labeled. The objective of the learning process is to create a model that minimizes deviation from the presented training examples. Light gray boxes represent training examples (i.e. inputs and their labels).

2.2.2 Reinforcement learning

In reinforcement learning, the IA learns by performing actions in an environment that is often simulated. The learning objective is to maximize the reward of the IA as it performs actions in the simulator. Rewards are typically represented by the outcome of games, so that the actions used to win or loose are positively and negatively reinforced by the learning algorithm respectively. The learning process is illustrated in Figure 2.3.

A major advantage of reinforcement learning is that there is no longer a need to manually label training data. Instead, a reward function is used to, in a sense, automatically label the data. However, designing a reward function for real world problems is a non-trivial task. It requires that appropriate rewards

can be assigned to the IA's actions over time [19]. A poorly designed reward function may result in undesirable and unexpected behavior.

Reinforcement learning is used in applications where IAs need to learn optimal action selection strategies. When applied in real-time strategy games, the IA can learn to select actions better than most human experts [6, 7]. Hence, it is reasonable to believe that reinforcement learning will eventually also be capable of generating alternative or even new tactics and strategies for military purposes.

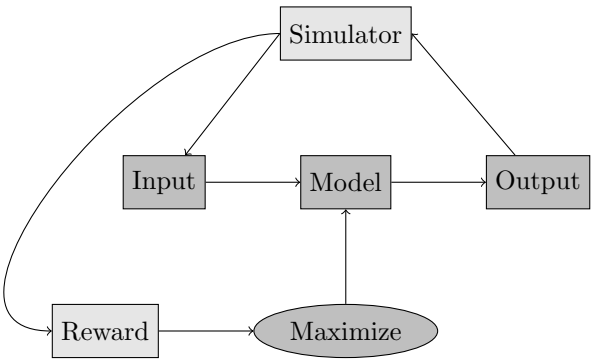


Figure 2.3 – Using reinforcement learning the IA learns by taking actions in a simulated environment. The objective of the learning process is to maximize the reward signal provided by the environment. Light gray boxes represent the input, in this case a simulator and a reward function, needed by this learning strategy.

2.2.3 Unsupervised learning

In unsupervised learning, the IA learns to identify patterns and structure in unlabeled data as illustrated in Figure 2.4. Note that learning, although referred to as unsupervised, is always guided by a pre-defined metric. For instance, the k-means clustering algorithm uses Euclidean distances to cluster data. Similarly, autoencoders (AEs) require the existence of a loss or error metric function.

The most common applications of unsupervised learning include clustering, visualization, dimensionality reduction and anomaly detection. A more recent application of unsupervised learning in DL is meta-learning, where IAs are trained with the objective to become faster learners (i.e. learning how to learn).

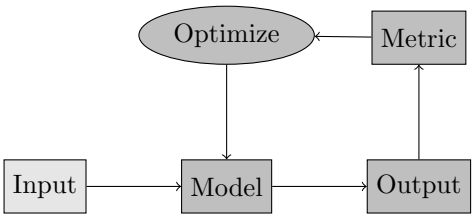


Figure 2.4 – In unsupervised learning, the IA learns to identify patterns and clusters in unlabeled data. Unsupervised learning is guided by pre-defined metrics (e.g. Euclidean distances in k-means clustering) to learn from the data.

2.3 Deep learning

Deep learning is a machine learning approach that can be used for all of the abovementioned learning strategies (i.e. supervised-, reinforcement- and unsupervised learning).

2.3.1 Deep neural networks

In DL, the model that is used to capture and learn from experience is represented by a DNN. A DNN is essentially a mathematical expression that consists of a large number of nested and differentiable subfunctions. The reason why the DNN must be differentiable is explained in Section 2.3.3.

DNNs are typically visualized using graphs where layers of nodes are connected to each other using edges as illustrated in Figure 2.5. In this representation, each edge represents a trainable parameter or weight, and each node represents a neuron (i.e. a differentiable subfunction) that uses the weights to transform inputs to outputs. Figure 2.6 illustrates the operations carried out by a single neuron. The neuron first calculates the sum of the products of its inputs and weights. This value is then processed by the neuron's nonlinear activation function to produce an output. The output is then used as input in the next layer of neurons.

In real world applications, the number of weights (edges in Figure 2.5) typically grows to millions and even billions. Note also that there are different types of DNNs besides the fully connected neural network (FCNN) illustrated in Figure 2.5. Convolutional neural networks (CNNs) are used when there are spatial relations in the data, which is typically the case in images. Similarly, recurrent neural networks (RNNs) are often used when there are known temporal relations in the data (e.g. text and audio). In real world applications, the model is typically designed using a mix of carefully selected CNNs, RNNs and FCNNs. The remainder of this section focuses on FCNNs. However, the same principles for inference and training also applies to CNNs and RNNs.

In this report the mathematical notation used to represent a DNN is f_θ, where θ represents the trainable weights or parameters of the DNN.

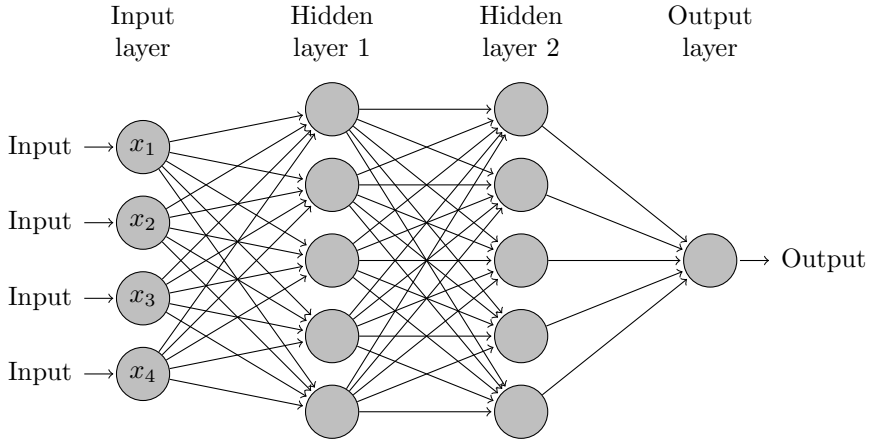


Figure 2.5 – Visualization of a fully connected DNN with four inputs, two hidden layers and one output. In this representation, each edge represents a trainable parameter or weight, and each node represents a neuron (i.e. a differentiable subfunction) that uses the weights to transform inputs to outputs. Each neuron calculates the sum of the products of its inputs and weights. This value is then processed by the neuron's nonlinear activation function to produce an output.

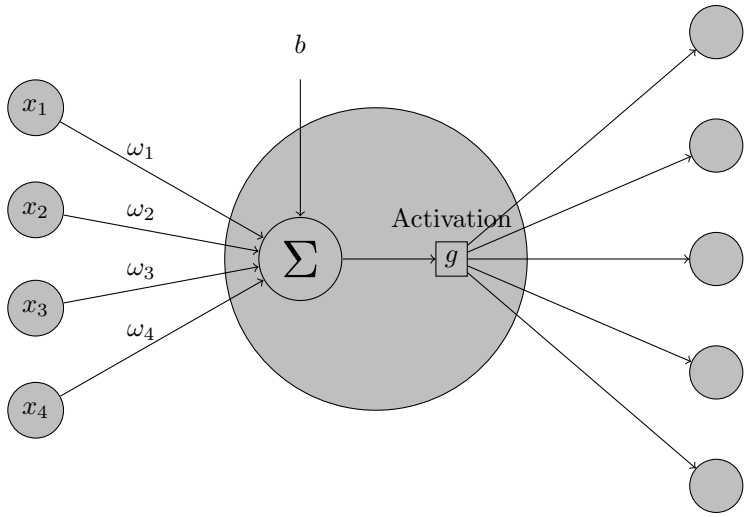


Figure 2.6 – Visualization of a neuron in a DNN. First, the sum of products using the inputs, x, and the weights, ω, are calculated. This value is then fed into the neuron's nonlinear activation function, g, to produce an output that can be fed into neurons in the next layer. The mathematical expression representing a neuron is $g(\sum x_i \times \omega_i + b)$. Note that bias, b, is also a trainable parameter that, unlike the weights, is not connected to an input.

2.3.2 Inference

Inference is the process in which inputs are processed by an already trained DNN to produce an output. In DNNs the processing is carried out in a forward pass through the layers of the network. Computational graphs are perhaps the most intuitive way to describe inference. In a computational graph the DNN is modularized into primitive subfunctions that represent the operations embedded in the network. As an example, the computational graph in Figure 2.7 represents a neuron with one input. Using this representation it is easy to see how the input is transformed as it moves forward (left to right) in the graph.

The computational graph can be extended to model DNNs with arbitrary number of inputs, neurons and outputs. In practice, it is common to design DNNs using computational graphs that represent aggregated layers. Different layers can then be connected to each other to form the final DNN.

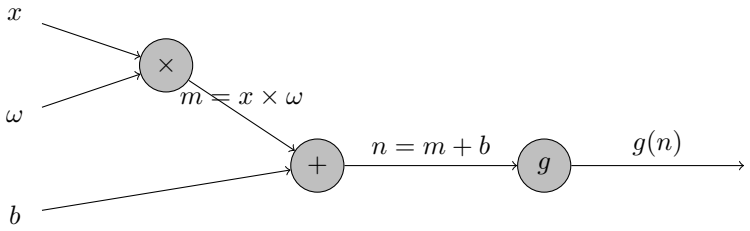


Figure 2.7 – Computational graph representing the operations, $f_\theta(x) = g(x \times \omega + b)$, of a neuron with one input, x, and pre-trained parameters $\theta = \{\omega, b\}$. The computational graph can be extended to include arbitrary inputs and outputs. In real world applications, the DNN consists of computational graphs representing aggregated layers of neurons.

2.3.3 Training

Training is the process where the DNN, f_θ, and its trainable parameters or weights, θ, are updated. Training is an iterative process where the objective is to adjust θ so that a loss function, $\mathcal{L}(f_\theta)$, is minimized. In practice, it is the gradient descent (GD) optimization method in Equation 2.1, or variants thereof, which is used to perform the update:

$$\theta = \theta - \alpha \nabla_\theta \mathcal{L}(f_\theta) \tag{2.1}$$

In the GD method, α represents a hyperparameter (i.e. a user defined parameter used to control the learning process) called the learning rate. The learning rate, α, controls the pace of the learning process. It is important that α is properly initialized to ensure that the trainable parameters are able to converge to an optimal solution. Generally, if α is too big the training process becomes unstable and the trainable parameters will not converge. In addition, if α is too small training will be stable, although it will take too much time to converge. For this reason, it has become common practice to use schedulers that dynamically can change the learning rate as the learning progresses.

The $\nabla_\theta \mathcal{L}(f_\theta)$ term in Equation 2.1 represents the gradients of the trainable parameters. The gradients determine in which direction to update the trainable parameters, θ, so that the loss function, $\mathcal{L}(f_\theta)$, increases. Note that updates are performed in the opposite direction of the gradients so that the loss is minimized.

To find these gradients the backpropagation algorithm is used. Given a training example $(x, \hat{y})$, the backpropagation algorithm first performs a forward pass to calculate the loss. Given the loss, a backward pass is then performed to calculate the gradients using the *chain rule* formula. Again, the most intuitive approach to explain backpropagation is to use computational graphs where DNNs are represented by a collection of subfunctions. To perform the backward pass all that is needed is to find the derivatives of these subfunctions. Let us illustrate backpropagation using a simple example where the DNN is represented by a linear function, $f_\theta(x) = \omega x + b$ with only two trainable parameters $\theta = \{\omega, b\}$. In this case, the loss function can be defined as the squared error of $f_\theta(x)$ and the desired output, $\hat{y}$:

$$\mathcal{L}(f_\theta(x), \hat{y}) = (f_\theta(x) - \hat{y})^2 = ((\omega x + b) - \hat{y})^2 \tag{2.2}$$

Hence, the loss measures if the DNN's prediction is close to the known output value, $\hat{y}$. When the loss is small, the prediction is good. Similarly, when loss is large, the prediction is poor.

The computational graph representing the loss function in Equation 2.2 is presented in Figure 2.8. In addition to the forward pass, this computational graph also includes a backward pass that propagates the loss (or error) back to the trainable parameters, $\theta = \{\omega, b\}$, using the chain rule. Note that it is only the derivatives of the loss with respect to the trainable parameters that are needed for training (i.e. $\nabla_\theta \mathcal{L}(f_\theta) = \left\{\frac{d\mathcal{L}}{d\omega}, \frac{d\mathcal{L}}{db}\right\}$). The backpropagation starts by setting $\frac{d\mathcal{L}}{dp} = 1$. From there it is easy to see how the chain rule propagates the error backwards (right to left) to find $\frac{d\mathcal{L}}{d\omega}$ and $\frac{d\mathcal{L}}{db}$. See Appendix A for a demonstration of the training process described in this section.

Even if the training process is simple and can be explained using computational graphs, it is difficult to understand and explain the behavior of the model. The next chapter presents XAI techniques that have been developed for these purposes.

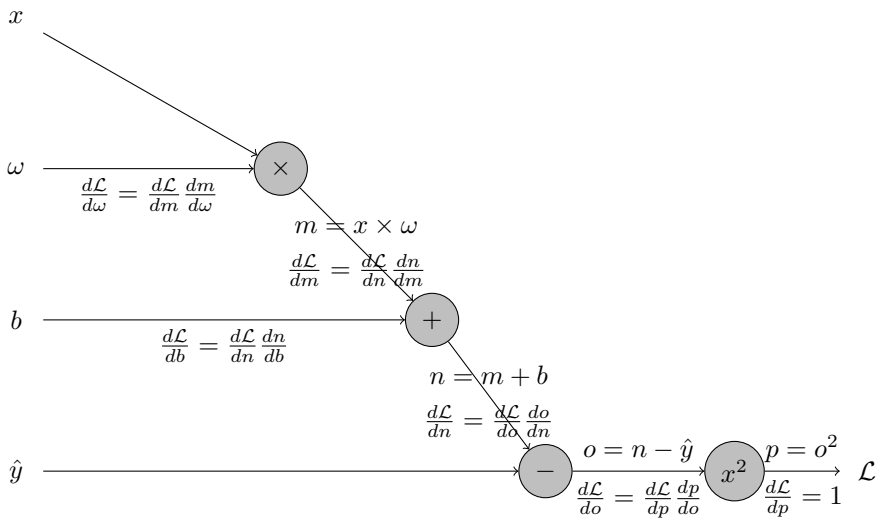


Figure 2.8 – Computational graph representing the squared error loss function, $\mathcal{L} = (f_\theta(x) - \hat{y})^2$. In this example, $f_\theta(x) = \omega x + b$ and $\theta = \{\omega, b\}$ represents the model and its trainable parameters respectively. x and $\hat{y}$ represents the input and its desired output (i.e. training data).

3 Techniques for explainable artificial intelligence

The focus of explainable artificial intelligence (XAI) research is to ensure that the reasoning and decision making of AI systems can be explained to human users. Although it has received a lot of attention recently due to the advancements in DL, the XAI research field is not new. It has been around since at least the 1980s [20]. For a comprehensive review of XAI research and its history, the reader is referred to [21].

Explainable artificial intelligence is a critical component in any military AI system used for high-stake decision making where human lives are affected. Examples of AI applications at the tactical level, where the focus is on short-term decisions, include autonomous control in unmanned vehicles as well as target identification, tracking and engagement in weapons and surveillance systems. Moreover, XAI is equally, or perhaps even more, important at the operational and strategic levels of warfare, where long-term decisions and planning activities could affect entire populations. At this level, AI systems are typically used for information analysis, but can also be used to propose plans or courses of actions (COAs) through simulation. The main purposes of XAI in military applications are:

- Mental modeling [14, 22]: XAI can be used to support users in the process of building appropriate mental models of the AI systems they operate. In any military system, AI enabled or not, users must have a clear understanding of the system's operating boundaries to ensure appropriate and efficient use.
- Insight [23, 24]: It has been shown that DNNs can be used to capture knowledge and identify patterns in observations of complex processes that are unknown to humans. Using XAI techniques it is possible for humans to unlock this knowledge and learn from it. Tactics and strategy development using reinforcement learning is a typical application where XAI could potentially generate deeper insights in the military domain.
- Laws and regulations [25, 26, 27]: XAI can potentially be used to ensure that AI systems follow national and international laws. Perhaps the most controversial application of AI is lethal autonomous weapon systems (LAWS) [26]. Some want to ban such systems altogether, while others argue that LAWS should be allowed as they could potentially improve accuracy and minimize collateral damage [27]. Nonetheless, the authors believe that XAI could play an important role in the process of developing policies regulating when, where and if AI systems such as LAWS can be used.
- Debugging [23, 28]: There are numerous cases in the literature where XAI has been used to identify bugs in DNNs. Bugs typically occur when artifacts such as copyright watermarks in images or unknown cheats in simulators and games that do not exist in real world data appear in the training data. The training process presented in Section 2.3.3 can learn to exploit, or take short-cuts, using such artifacts. The result is a DNN that works well when presented with test data, but fails when real world data is presented. This kind of problem can be detected and addressed before deployment if XAI techniques are used as an integrated part of the development process.

This chapter introduces several XAI techniques that have been developed specifically in the context of DL. XAI for DL is a major challenge because the DNNs may consist of millions or even billions of parameters, making them opaque and difficult to interpret by humans. Note that, to the best of our knowledge, the proposed techniques have not yet been scientifically evaluated in a military context. Hence, it is unknown to what degree these techniques can provide useful explanations in this context. Chapter 4 provides an introduction to how such evaluation can be performed.

3.1 Global explanation techniques

Global explanation techniques provide insight into the DNN and its behavior as a whole. In this section, we primarily focus on techniques that can be used to analyze and visualize high-dimensional training datasets but also how to acquire and interpret performance measurements for model evaluation purposes.

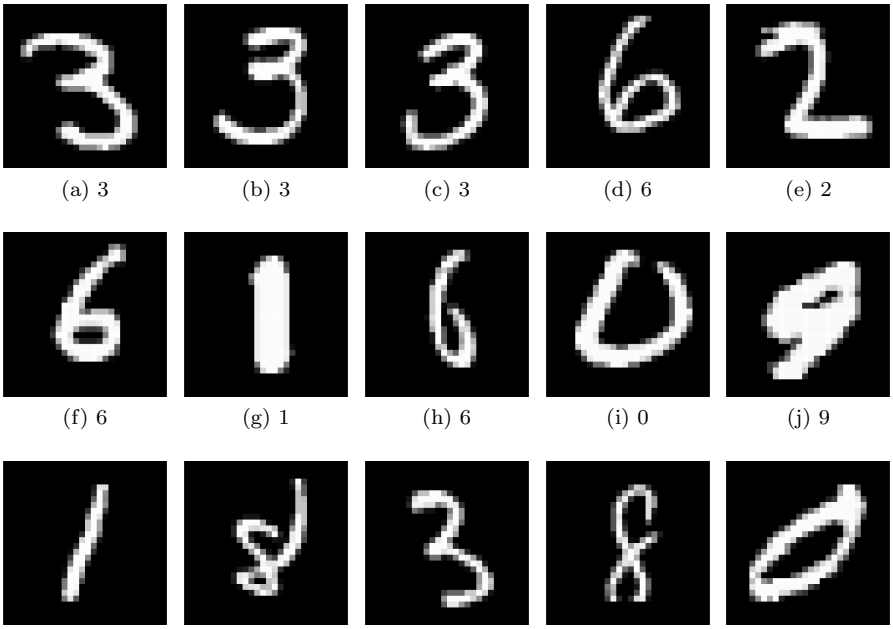
3.1.1 Visualization techniques for large high-dimensional datasets

In DL, training datasets typically consist of a large number of high-dimensional samples. To visually inspect such datasets they must be reduced to a dimensionality that is observable to humans (i.e. one-, two- or three dimensional space). Summarizing large datasets in a visualization can provide useful insights about the complexity of the task to be learned by the DNN. It may also be used to identify artifacts in the dataset that could negatively impact the performance of the DNN [23]. Below are three unsupervised techniques that can be used to reduce dimensionality for visualization purposes:

- Principal component analysis (PCA) [29]: This technique identifies the principal components of the dataset. The data is projected onto the components or vectors that are considered to be the most important ones. The main drawback of PCA is that it is a linear technique, hence, it may fail to identify patterns in nonlinear data. The main advantages of PCA is that the technique is well understood (i.e. it can be explained) and that it is computationally efficient compared to other techniques.
- Variational autoencoder (VAE) [30]: This is a DL technique that uses DNNs to reduce dimensionality. The VAE consists of two DNNs: the encoder and the decoder. The purpose of the encoder is to compress the high-dimensional input data into a latent space vector (in this case of one-, two- or three dimensions). The purpose of the decoder is to, as accurately as possible, reconstruct the high-dimensional data using the low-dimensional latent space representation. Training the DNNs is performed using a loss function that minimizes the error of the original input and its reconstruction using GD and backpropagation as introduced in Section 2.3.3. Once trained, only the encoder is needed to reduce dimensionality. The main advantage of this technique is that it is able to learn non-linearities in the data. The disadvantage is that the VAE is built using opaque DNNs that are not easily explained to humans.
- t-distributed stochastic neighbor embedding (t-SNE) [31]: This technique was specifically developed for visualization purposes. Similarly to VAE, t-SNE uses the GD procedure to learn how to optimally reduce the dimensionality of the data. In this case, the objective function targets the preservation of neighborhood distances. The advantage of t-SNE is that it generally produces better visualizations. A disadvantage is that it is computationally complex.

To demonstrate the above techniques, the MNIST dataset [32] will be used. This dataset contains gray-scaled images including labels representing 70000 hand-written digits. Each image consists of 28×28 pixels, hence, the dimensionality of the data is 784. Figure 3.1 illustrates 15 samples randomly drawn from the dataset.

The visualizations (scatter plots) in Figure 3.2 were created using a subset of 10000 images randomly drawn from the MNIST dataset. In this case the dimensionality was reduced from 784 to 2 using PCA (Figure 3.2a), VAE (Figure 3.2b), and t-SNE (Figure 3.2c and Figure 3.2d). The plots were rendered using all of the 10000 data points, and the label of each data point was color coded so that clustering tendencies can be visually inspected by humans. In Figure 3.2d the dataset was first preprocessed using PCA to reduce the dimensionality from 784 to 50 prior to using t-SNE. This is standard practice when using t-SNE to ensure computational efficiency. The visualizations in Figure 3.2 provide an insight into the complexity of the dataset. If clusters can be visually identified it is also very likely that a DNN will be able to efficiently learn from the data. Similarly, if clusters cannot be identified it will also be more difficult for DNNs to learn from the data. In this case, the PCA technique was not able to separate the clusters. Hence, a linear classifier cannot be expected to perform well.

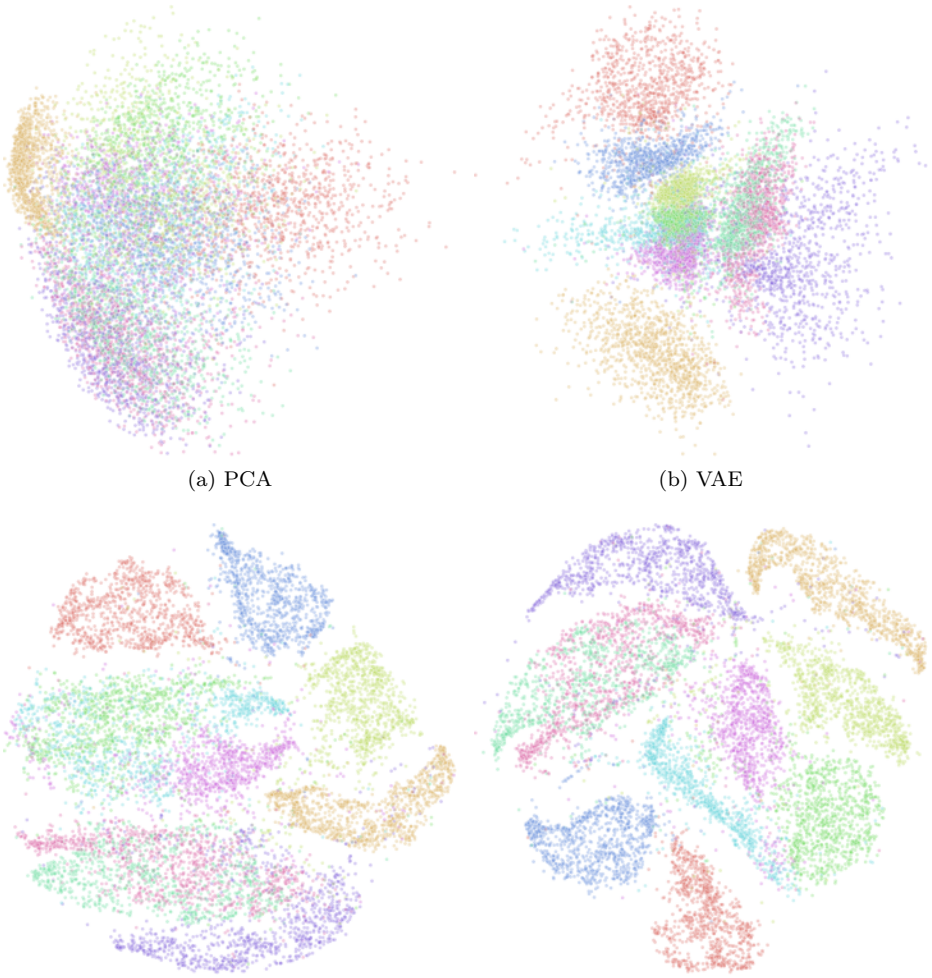


(a) 3 (b) 3 (c) 3 (d) 6 (e) 2

(f) 6 (g) 1 (h) 6 (i) 0 (j) 9

(k) 1 (l) 8 (m) 3 (n) 8 (o) 0

Figure 3.1 – Samples randomly drawn from the MNIST dataset. The labels of the samples are provided in the figure captions.



(a) PCA (b) VAE

(c) t-SNE (d) t-SNE with PCA preprocessing

Figure 3.2 – Visualization of high-dimensional data in two-dimensional scatter plots using principal component analysis (PCA), variational autoencoder (VAE) and t-distributed stochastic neighbor embedding (t-SNE). In this case, the dimensionality was reduced from 784, representing images of 28×28 pixels, to 2. The plots were rendered using 10000 data points, and each data point was color coded by its label (0 to 9) so that clustering can be visually inspected by humans. In Figure 3.2d the dataset was preprocessed using PCA to reduce the dimensionality from 784 to 50 prior to using t-SNE. This is standard practice when using t-SNE to ensure computational efficiency. The visualizations provide an insight into the complexity of the dataset. If clusters can be visually identified it is also very likely that a DNN will be able to efficiently learn from the data. Similarly, if clusters cannot be identified it will also be more difficult for DNNs to learn from the data.

3.1.2 Model evaluation

When training a machine learning model, a model developer continuously measures how the model is performing on input data it has not seen before, in order to recognize if the model is progressing towards useful behavior. When the developer is satisfied with the model's performance, the training process is stopped and a final evaluation is performed using unseen test data. This final test measures the expected performance of the model when applied in the real world, where it will typically encounter inputs it did not see during training. The extent to which the test dataset can be used to measure real performance depends on how well the test set corresponds to real world data. While the ongoing measurement of performance during model training and tuning is mainly interesting for model developers, the final performance measure is also valuable for users from an XAI perspective.

3.1.2.1 Evaluation of classifiers

In the example of classifying military vehicles from images, of which there are thousands for each class of vehicle, a significant proportion of the images would be used for training, a separate set of images would be kept apart for fine tuning and testing the model during training, and yet another set of images would be reserved for the final performance measure. Since the classifier did not see the images in the test set during the training process, measuring its performance on them provides an idea of how well the model performs on new data.

In a classification task, the most straightforward measure of performance is to count the proportion of correct classifications. This measure is called *accuracy*:

$$\text{accuracy} = \frac{\text{correct classifications}}{\text{all classifications}} \tag{3.1}$$

That is, if the vehicle classification model is tested on 100 images and 85 are correctly classified, the accuracy of the model on the test data is 85%. Accuracy works well if instances from the different classes tend to occur with equal frequency, that is, the data is *balanced*.

In the case of a sea-mine classification example the task is to analyze sonar images of mine-like objects and classify the object as a mine or something else (typically a rock). In this case there may be a relative shortage of mine images to train on, since data about rocks is easy to gather while data about mines, particularly those deployed by hostile forces, is not.

The mine detection case is an example of an *unbalanced* problem, which, if the test dataset is to reflect real world occurrences, will contain many more images of rocks than images of mines. As an example, assume that one example in a thousand in the test dataset is a mine (and the rest are rocks). A classifier that always returns negative classifications (not mine) would achieve 99.9% accuracy on the test set since 999 out of one classifications will actually be correct. Yet, it is useless for finding mines because out of the actual mines that are presented to it, it detects none. It has a *recall* rate of 0%.

The recall rate can be increased by making the classifier more prone to return positive classifications (mine) for suspicious objects. In the extreme, a classifier that always returns positive classifications would trivially achieve 100% recall, because it catches all the mines, along with all the rocks. Yet, again, it would be useless, because for every thousand positive predictions, only one would be correct. Its *precision* would be 0.1%.

Clearly, a good mine detector, or any classifier for that matter, needs to have reasonably high values for both *precision* and *recall*. That is, it must be

possible to trust positive classifications enough to invest further resources (such as deploying divers). It must also be possible to trust negative outputs enough to expect it to find a decent proportion of mines that are actually there. In reality there is, however, always a trade-off between the two, and the correct balance depends on the particular operational requirements. If, for instance, it is important not to miss mines, the classifier would be tuned to high recall. However, there is a price to be paid in terms of lower precision, leading to more time being devoted to investigating rocks.

Accuracy, precision and recall can be calculated by running the classifier on the test dataset and counting how many mines were correctly classified (*true positives* or TP), how many rocks were correctly classified (*true negatives* or TN), how many rocks were mistaken for mines (*false positives* or FP), and how many mines were mistaken for rocks (*false negatives* or FN). This yields a *confusion matrix*, as shown in Table 3.1.

Table 3.1 – The structure of a confusion matrix, tabulating the number of correct positive classifications (TP), the number of correct negative classifications (TN), the number of incorrect positive classifications (FP), and the number of incorrect negative classifications (FN).

	actual mine	actual rock
predicted mine	TP	FP
predicted rock	FN	TN

The confusion matrix is a compact but rich way of representing the performance of the model, from which many different metrics can be deduced. A high *precision* model has a high TP value compared to other values on the same row (FP), or more formally:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

A high *recall* model has a high TP value compared to other values in the same column (FN), or more formally:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

A high *accuracy* model has high values in all diagonal positions compared to non-diagonal positions, or more formally:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.4}$$

Other combinations of the values in the matrix yield other metrics, and each metric sheds light on some aspect of the model's performance. In general, a case with unbalanced data, which tends to be the norm in reality, will require more than a single metric to gauge the model's performance. The right set of metrics for the problem at hand, however, provides a concise picture of how the model can be expected to perform in the wild. Since all metrics are calculated from the confusion matrix, a trained analyst will quickly be able to extract this information from it.

3.1.2.2 Evaluation of multi-class classifiers

If a vehicle classifier is to distinguish between tanks, motorcycles and transport vehicles, there is a *multinomial* or *multi-class* classification problem. In such a case the confusion matrix will have as many rows and columns as there are

classes. An example of a multi-class confusion matrix is illustrated in Figure 3.3, where the task is to classify images of hand-written digits from 0 to 9, that is, ten classes.

The calculation of metrics from the confusion matrix generalizes in that accuracy is given by comparing the diagonal to the rest, whereas precision and recall are given for each particular class by comparing its diagonal value to the sum of its row (precision) or the sum of its column (recall). Thus, by color-coding the matrix, as in the digit classification example, much information can be gleaned by mere inspection. For example, as can be seen in this case, overall accuracy is extremely high (comparing the diagonal to the rest), but performance varies somewhat in the different digit classes. Number five is sometimes a misclassification of three or six or others, and five is conversely sometimes mistaken for three. The number one, however, is hardly ever confused with anything else.

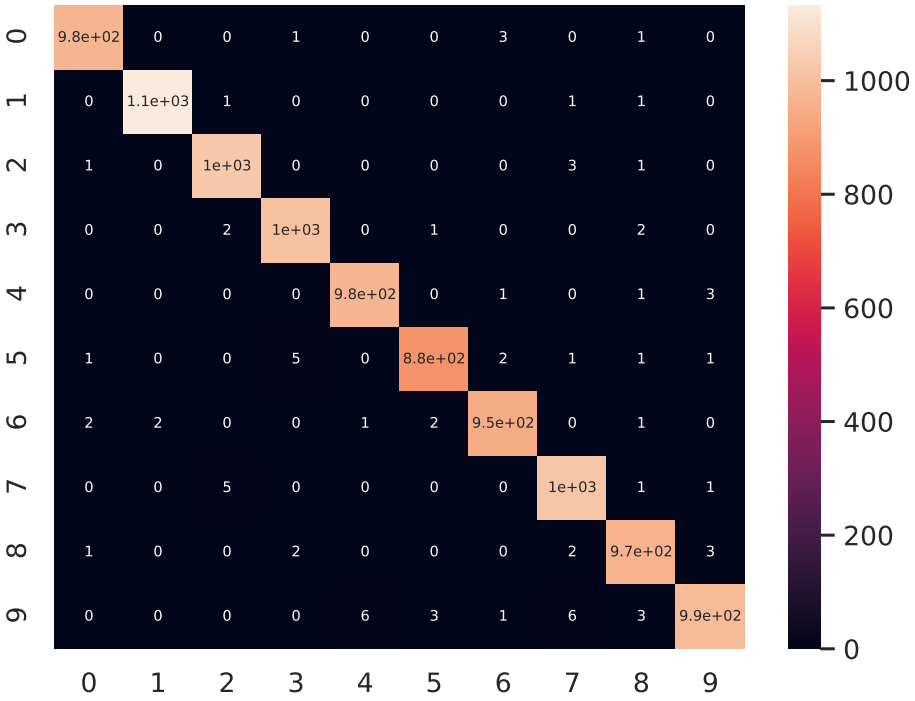


Figure 3.3 – Confusion matrix illustrating the performance of a DNN trained to recognize hand-written digits using the MNIST dataset. The confusion matrix can be used to gain insight into which digits the model is most likely to confuse with other digits.

3.1.2.3 Evaluation of regression models

In a *regression* task, it is impossible count correct classifications. Instead, it is necessary to compare continuous values produced by the model with correct values in the test set.

As an example, assume that an obstacle avoidance model for an autonomous ground vehicle (AGV) is being trained. The AGV must produce a steering signal based on input from mounted sensors. The steering signals are represented as a number between -1 and 1, where -1 means *sharp left turn*, 1 *sharp right turn*, 0 means *no turn*, and everything in between are gradations of turning in the corresponding direction. The AGV has been trained on data recorded by

human operators. It is tested by taking the steering signals it produces for a given sensor stimuli, and comparing it with the recorded data. For instance, the recordings might say that detecting an obstacle to the left at a far distance should produce a limited right turn signal (e.g. 0.2), whereas detecting an obstacle to the left at close proximity should produce a sharp right turn (close to 1). A model, A, that in the latter case produces a sharp left signal (-1) should be judged to perform worse than another model, B, that produces a slight rightward signal (e.g. 0.2). Comparing the model's prediction to the desired value, it is apparent that model A is at a distance of 2 from the value, whereas model B is at a distance of 0.8. Hence, model B is closer to the correct behavior. If such errors for all the instances in the test dataset are measured and aggregated, for instance calculating the average error, an overall measure of the model's performance is obtained.

Evaluation techniques for regression differ mainly in how this aggregation is done. *Mean absolute error* (MAE) takes the average of the absolute value of the error. The metric measures how much the model predictions deviate from the desired values. *Root mean square error* takes the root of the average square of the error. It corresponds to the standard deviation of the error and differs from MAE in that it penalizes large deviations more. *R Squared* (R^2) compares the mean squared error to the variance of the signal itself. Therefore, it tolerates larger errors for signals that vary a great deal in the first place.

3.2 Local explanation techniques

In contrast to the global explanation techniques, local explanations are used to explain the prediction for a specific input of interest. These inputs can be real world examples or examples from the training- or test datasets. The input to a DNN is essentially constituted by a list of numeric values, representing some real world process, such as pixels in an image, letters in a text, scientific data, and so on. A gray-scale image with 300 pixels thus represents its data in 300 dimensions, each dimension telling a part of the story (and all dimensions together providing the whole).

This section focuses on local explanation techniques where saliency maps are used for explanation. A saliency map explains the output of a model when provided with a particular input example, by attributing scores of relevance, or *saliency*, to each input dimension. That is, it shows how important each dimension is in producing the particular output corresponding to the example. For an image, these saliency scores translate into a heatmap that can be superimposed on the image to indicate which pixels the model paid attention to when producing its decision. Figure 3.4 provides an example of a saliency map generated in the context of a simulated self-driving car.

The first two techniques introduced here are white-box techniques. These techniques rely on access to the internal representation of the DNN (subfunctions, gradients, etc.) to produce explanations. The other techniques are black-box techniques that can produce explanations by querying the model, often many times, with selected inputs. Hence, black-box techniques tend to require more computational resources to produce their explanations.

Figure 3.4 – Saliency map highlighting the most important input pixels used by a DNN to control a simulated self-driving vehicle. In this case, the pixels representing the right side of the road appear to be the most important ones.

3.2.1 Gradient saliency

Gradient saliency (also called sensitivity analysis) is one of the earliest local explanation techniques, and it has been used to explain the behavior of neural networks for a long time [33, 34]. The idea in gradient saliency is to generate explanations by calculating how much changes in input values will change the model output value. Inputs where changes in their values will affect the model output value the most are considered more important for the model output value than other inputs. In mathematical terms, this is called the derivate of the model output for a given input, such as an image. Since DNN training uses derivatives, many deep learning software libraries can directly compute gradient saliency.

The computational graph in Figure 2.8 that was used to explain the DNN training process can also be used to understand how gradient saliency works. Instead of calculating the derivatives with respect to the trainable parameters, which is what is done during training, the gradient saliency technique calculates the derivatives with respect to the input (i.e. $\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dm}\frac{dm}{dx}$).

The problem with gradient saliency is that it does not distinguish between the signal that affects model output and distractors that the DNN is trained to filter out [35]. Explanations generated using the gradient saliency technique tend to be noisy and can also hide features that the model actually uses. Explaining which features make an input more or less of an object type is not as informative as explaining which features that make it the object type in reality [34].

3.2.2 Layerwise relevance propagation

Layerwise relevance propagation (LRP) was published in 2015 and is one of the first techniques that uses a theoretical framework to guide the development of local explanation heuristics [36, 34, 37]. The main benefit of the theoretical framework is that it provides a way to find local explanation heuristics that are suitable for many types of layers at all levels in DNNs, as well as local explanation heuristics for other types of machine learning models.

LRP starts by assuming that assigning relevance for how lower layers contribute to each output value should consider which activations that are necessary for the output value. Removing these relevant activations from the input should ideally cancel out that output value. For example, removing all features of a car in an image that is classified as *car* should mean that the model's output value for *car* is zero. In mathematical terms, this is called a root of the model function, and the idea in LRP is to use local explanation heuristics that are suitable to search for this root.

Although there is no known technique to optimally search for a root of model functions, there are some constraints to the search that have proven to be sufficient. For example, the activations of the root should be near the activations of the output value, relevant activations should be within the possible input space, and only the available relevance of the output value should be used for assigning relevance to activations. These constraints have proven sufficient to find local explanation heuristics that propagate relevance from the model output back to the input.

LRP assumes that the model function can be approximated using the mathematical technique Taylor expansion. Taylor expansion decomposes the model function into simple additive terms that can be directly mapped to neural network components. The additive terms mean that the model function can be decomposed into relevance scores for each activation that underlies the model output.

LRP is a family of local explanation heuristics that use these techniques for relevance propagation [37]. The heuristics are specifically adapted for different types of neural network layers and layer levels. Some heuristics can also propagate output relevance into positive activations that contribute to the model output and negative activations that detract from the model output. This may be useful to identify missing features that would make the model output more likely.

3.2.3 Shapley additive explanations

The Shapley additive explanations (SHAP) was published in 2017 [38], promising to improve a number of previous methods by pointing out a mathematical commonality between them and then proving that they could all be improved by use of a particular mathematical formula.

The formula in question was introduced by the Nobel laureate Lloyd Shapley in 1953 [39] in the field of game theory, a branch of economics. It calculates so called Shapley values, which serve to distribute the gains from some joint enterprise, or game, among the participating actors. The formula is designed to distribute the gains fairly, according to a set of soundness conditions, so that all gains should be distributed; actors who contribute more should have more of the gains; non-contributing actors should gain nothing; and it should be possible to sum up gains across games. In fact, Shapley showed that his formula is the only one that can possibly satisfy all the conditions.

In terms of explanations, the first step is to observe that the input dimensions to a machine learning model can be viewed as actors participating in the model's game of producing an output prediction. The output value can be seen as the total gain from the game, that is, to be distributed among the actors. To perform a fair distribution is to distribute the output value among the input dimensions in proportion to their contribution. In other words, Shapley values, thus applied, produce a saliency mask. This observation was made before the SHAP method, for example in [40] and [41].

The contribution of the original SHAP work was to observe that a number of earlier methods were all producing explanations that could be unified under a common linear form, called an additive feature attribution, meaning that they all share the property that the saliency values they produce sum up to the output value of the model to be explained. The authors of [38] then set up soundness conditions corresponding to the ones discussed above, and proved that the Shapley formula is the only way for a feature attribution method to satisfy all conditions. Since all the previous methods deviate in some way from the Shapley formula (typically by applying some heuristic without much the-

oretical foundation), the authors argued that the methods could be improved by adjusting them to the formula. Thus, SHAP is in fact a family of methods based on these adjustments. For instance, adjusting LIME (see Section 3.2.4) to accord with the Shapley formula yields KernelSHAP, a model-agnostic version of SHAP. Versions of SHAP that are based on model specific explanation methods inherit the same model specificity constraints.

3.2.4 Local interpretable model-agnostic explanations

Local interpretable model-agnostic explanations (LIME) generated a great deal of attention when it was published in 2016 [42], because it was one of the first explanation methods that could be applied to any model as a black-box. LIME explains a model's prediction on an input example by performing perturbations on the example and observing what happens.

Any machine learning model represents the relationship between its inputs and its outputs as some mathematical function, defined by the weights and structure of a neural network or by other parameters. This function in turn aims to capture some real world relationship, for instance, the relationship between a sequence of sounds and a sequence of words. The function modeled by a typical modern machine learning system is complex, which is why simply inspecting the weights of a neural network does not do much by way of explanation. LIME disregards the function as a whole, but instead tries to describe what the function does in the vicinity of the example that is to be explained. By perturbing the input in different ways, it is able to create a linear, and hence much simpler, model that behaves close to the complex one on examples that are similar to the one provided. The coefficients of this linear model then constitute a direct measure of which dimensions of the input have the greatest impact on the output of the model, or in other words, the coefficients are LIME's version of a saliency mask. Since all LIME had to do to the model was to feed it different perturbations of the input and observe its output, nothing had to be known about the model's internal workings.

3.2.5 Randomized input sampling for explanation of black-box models

Randomized input sampling for explanation of black-box models (RISE) is a model-agnostic local explanation technique that was published in 2018 [43]. Similarly to LIME, RISE generates explanations by perturbing the input and observing how the model reacts. No knowledge about the model's internal working is therefore necessary for generation of explanations.

RISE perturbs images by randomly generating masks that dim image pixels. The masks are generated by dividing the image into larger areas and randomly selecting which areas to include in the perturbed image. The model's output value for the perturbed image describes the extent to which the mask covers image areas that are important for the classification of that model class. Masks that cover many image areas that are important for the classification result in higher model output values compared to masks that cover fewer important image areas. By randomly generating many masks, RISE calculates the average importance of each image area. The image area's importance explains the model's classification.

A benefit of RISE is that it uses image areas of even size to generate explanations. The explanations therefore cover the same image regions as objects in the image. LIME, on the other hand, uses superpixels (continuous areas of similar pixel values), that may not capture correct image regions.

3.3 Hybrid explanation techniques

Hybrid explanation techniques provide insight by combining global and local XAI techniques. Instead of only using local XAI techniques on a case-by-case basis, hybrid explanation techniques automatically apply local XAI techniques on a large number of cases, typically whole datasets. Hybrid explanation techniques then compare all local XAI results to identify cases of where the model does not perform as expected. Such anomalies may inform further model development or indicate performance limitations to consider when using the model.

3.3.1 Spectral relevance analysis

The spectral relevance analysis (SpRAy) technique was introduced in [23]. SpRAy is a semi-automated technique that uses a whole dataset analysis approach to find cases where the model does not perform as expected. For example, in image classification, a general type of object, such as *dog* or *car* may appear in many forms and contexts, but similar object forms and contexts should have similar local XAI results. If the local XAI results are dissimilar for some cases compared to what is expected, this may indicate anomalous model behavior. Decision strategies that rely on spurious and artificial correlations that may not exist in the real world is also known as "Clever Hans" behavior. SpRAy consists of five steps to find anomalous model behavior:

- Compute relevance maps with LRP (see Section 3.2.2).
- Pre-process all relevance maps to a uniform shape and size.
- Perform spectral clustering on the relevance maps. Spectral clustering is an established technique that transforms a similarity matrix (measure of similarity between cases) into a representation that enhances the cluster-properties of the similarity matrix [44]. Clusters can then be detected in the new representation. The similarity is computed between relevance maps as the nearest neighbors from the Euclidean distance between pairs of relevance maps. The Euclidean distance between two relevance maps is computed from the difference in intensity for each color channel at every pixel.
- Identify interesting clusters. Spectral clustering computes measures (eigenvalues) that indicate disjoint or weakly connected clusters. Large gaps in eigenvalues indicate that the clusters are different.
- An optional step is to visualize the clusters using, for instance, t-SNE (see Section 3.1.1).

In [23], SpRAy was used to show that a previous generation of machine learning technique, the support-vector machine (SVM), learned spurious correlations in image classification. For instance, SpRAy showed that the classifier used four different strategies to classify images of horses, detect a horse and rider, detect a source tag in landscape or portrait oriented images, and detect hurdles and other contextual elements. This classifier was therefore unreliable in actual applications where source tags and contextual elements were not present. Adding the source tag to images of other objects, such as *car*, they could change the classification to *horse*.

4 Evaluating explainable artificial intelligence techniques

An often overlooked but important aspect of XAI is the ability to evaluate the proposed XAI techniques. Section 4.1 introduces evaluation criteria from a human factors perspective, where the user (e.g. operator or analyst) is central for measuring the effects of XAI when added to the AI system. Moreover Section 4.2 introduces tests that can be used to compare local XAI techniques, such as the ones presented in Chapter 3.2, using heuristics.

4.1 Human factors evaluation

A human factors evaluation of XAI techniques tests whether the explanations consider all factors that are important for users to fully utilize the AI system. For example, users may have different goals, needs, knowledge, experience, task contexts, use cases, etc. As in many types of system development, it is important to consider these factors throughout the development of the AI system, from system specifications to the final user testing. Since XAI techniques for DL is an emerging research area, initial users of the techniques are often system developers that are interested in evaluating model performance. Whether these XAI techniques are also useful for military users is largely still an open question. In [22], six metrics have been proposed to evaluate explanations:

- *Explanation goodness*: Consists of a checklist of important aspects to consider from a user perspective in the development of XAI techniques. The checklist is based on a thorough review of existing literature about explanations and includes seven important aspects of explanations, for example, whether the explanations help users understand how the AI system works, whether the explanations are satisfying for users, and whether the explanations are sufficiently detailed and complete.
- *Explanation satisfaction*: A scale to measure how users experience the explanations with respect to *explanation goodness*. The scale consists of eight items that are formulated as statements (the seven goodness aspects and one item regarding whether the explanation is useful for the users' goals). A validity analysis shows that the scale is reliable and can discriminate between good and bad explanations.
- *Facilitation of mental models*: Good explanations strengthen users' understanding of how the AI system works and why it makes a particular decision. Within cognitive psychology, such representation is called the user's mental model of the AI system. Four tasks are recommended to measure users' mental model of AI systems, for example, a cued retrospection task where users are asked to describe their reasoning after performing a task with the AI system, and a prediction task where users predict what the AI system will do. A comparison between the user's mental model and an expert's mental model shows the completeness of the user's mental model.
- *Promotion of curiosity*: Good explanations promote users' curiosity to investigate and resolve knowledge gaps in the mental model. It is recommended to measure curiosity by asking users to identify the triggers that motivated them to ask for an explanation. Some examples of triggers are: justification of the AI system's action, why other options were excluded,

or the AI system not behaving as expected.

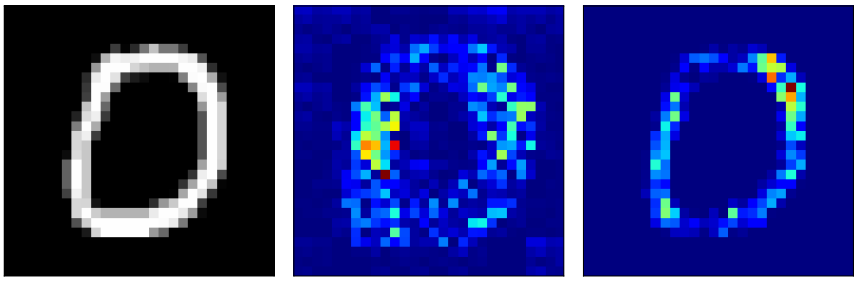
- *Trust in explanations*: A good mental model enables users to appropriately trust the AI system and use it within its operating boundaries. A scale with eight items to measure user trust in the AI system is recommended. For example, the items address user confidence in using the system and the system's predictability and reliability.
- *System performance*: The ultimate goal of XAI is to improve overall system performance compared to only using the AI system without XAI. Examples of performance measures include completion of primary task goals, user ability to predict the AI system's responses, and user acceptance.

Future studies will provide more information on how to interpret these metrics when evaluating XAI techniques for AI systems.

4.2 Evaluating local explanation techniques

The local XAI techniques described in Chapter 3.2 generate saliency maps to highlight the importance of each input dimension. The saliency maps are visualized differently depending on the type of data that is being processed by the model. For instance, heatmaps are typically used when processing images, whereas color coded characters and words are typical when processing text.

Figure 4.1 presents an example where saliency maps are visualized using heatmaps. In this case the heatmaps were generated for the digit 0 (Figure 4.1a), using the gradient saliency (Figure 4.1b) and LRP techniques (Figure 4.1c). Important dimensions (i.e. pixels in the image) are represented by warmer colors (e.g. red, orange, yellow, etc.), whereas non-important dimensions are represented by colder colors (dark blue, blue, light blue, etc.). A notable difference between the two techniques can be visually observed in the placement of the highlighted dimensions. The remainder of this section introduces techniques that can be used to quantitatively compare and evaluate local explanations generated by different techniques. Ultimately, the goal is to find out which explanation is the most accurate one.



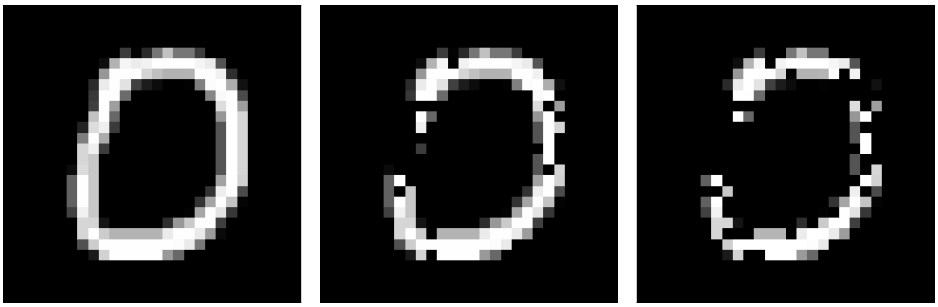
(a) Digit 0 from MNIST (b) Gradient saliency (c) LRP

Figure 4.1 – An MNIST image and its corresponding heatmaps generated using the gradient saliency and LRP techniques. The important dimensions, or pixels in the image, are represented by warmer colors (e.g. red, orange, yellow, etc.).

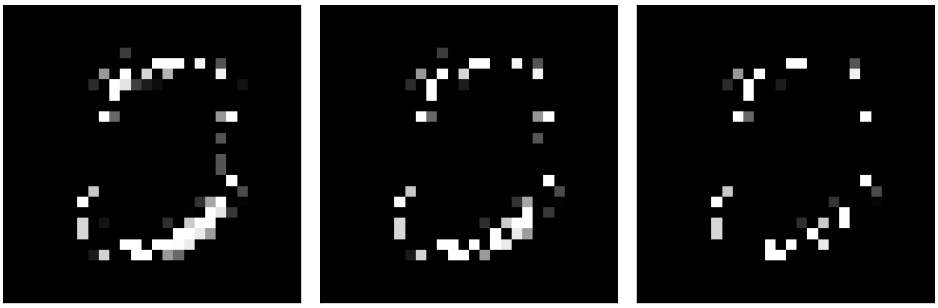
4.2.1 Deletion

Deletion [43, 34] is a metric that is calculated by measuring the model's ability to accurately make predictions while inputs are gradually distorted or deleted. Note that deleted in this case means converting the values of the inputs into something neutral (e.g. the background of an image). The deletion process is guided by the saliency maps generated by the XAI techniques so that values in more important dimensions are deleted prior to less important ones. The intuition of this metric is that an explanation is better if the performance drop is fast, as opposed to slow, when the deletion process progresses.

Figure 4.2 illustrates the deletion process using the gradient saliency map from Figure 4.1b. In Figure 4.2b, 50 of the most salient pixels have been deleted. At this stage it is easy to infer that the image is still representing a 0. In Figure 4.2f more than half of all pixels (400) have been deleted. At this stage it is much more difficult to infer that the image actually represents the digit 0.



(a) 0 pixels (b) 50 pixels (c) 100 pixels

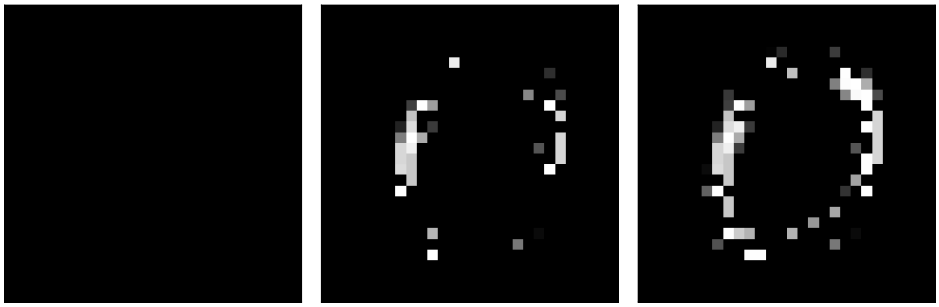


(d) 200 pixels (e) 300 pixels (f) 400 pixels

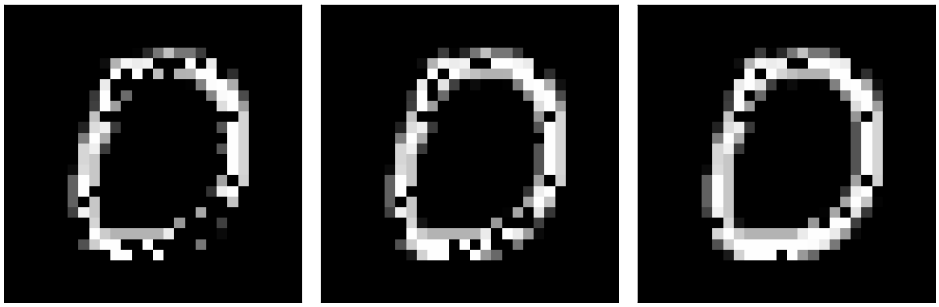
Figure 4.2 – Images generated from the deletion procedure for an MNIST image, where 0, 50, 100, 200, 300 and 400 pixels are deleted.

4.2.2 Insertion

The insertion metric [43] is the complementary approach to deletion. Figure 4.3 illustrates the insertion procedure for the same MNIST image used in the deletion example. Starting with the initial input (represented by a black image), the increase in accuracy is measured as more and more dimensions of the input are inserted as prioritized by the saliency map. Here, the intuition is that the accuracy of the model's prediction should increase when more information is inserted into the input. That is, when the increase is fast the explanation is better compared to when it is slow.



(a) 0 pixels (b) 50 pixels (c) 100 pixels



(d) 200 pixels (e) 300 pixels (f) 400 pixels

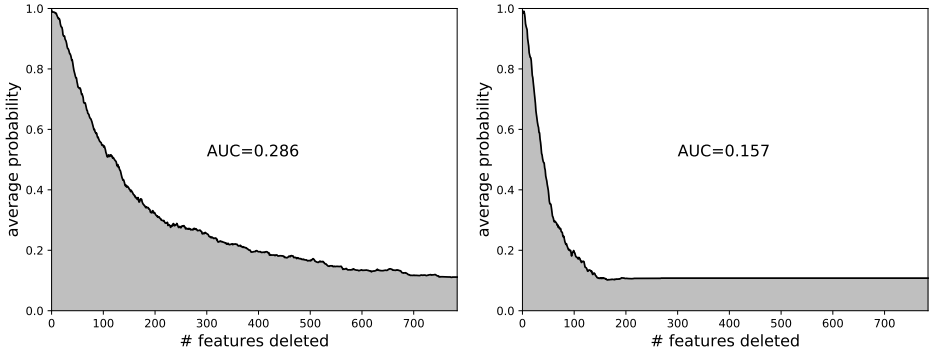
Figure 4.3 – Images generated from the insertion procedure for an MNIST image, where 0, 50, 100, 200, 300 and 400 pixels are inserted.

4.2.3 Evaluation metrics

To demonstrate the use of deletion and insertion, these procedures were applied using the gradient saliency and LRP techniques. In this case, the XAI techniques were evaluated using a classifier with a batch of 100 randomly sampled images drawn from the MNIST dataset.

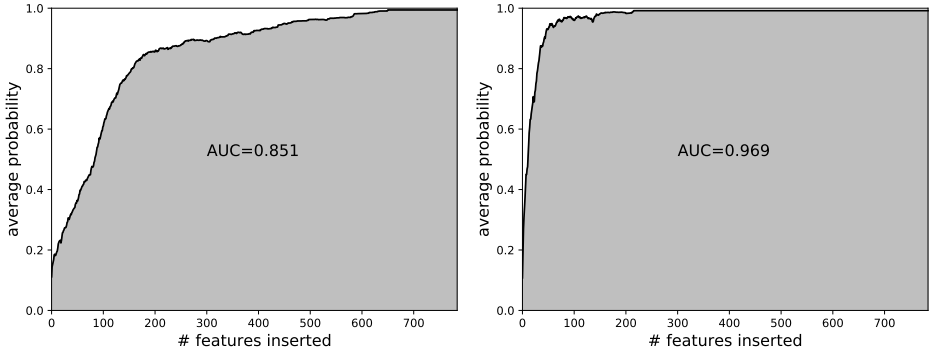
The results of the deletion and insertion procedures are presented in Figure 4.4 and Figure 4.5 respectively. The area under the curve (AUC) is a measurement that can be used to quantitatively compare XAI techniques with each other. For deletion, smaller AUC values are better than larger values. Similarly, for insertion, larger AUC values are better than smaller values.

In Figure 4.4 it can be observed that the decrease in the performance curve from the LRP technique is sharper and converges to a lower average probability value using the deletion procedure. This is in coherence with its heatmap, which highlights fewer features compared to the heatmap from gradient saliency (Figures 4.1c and 4.1b), indicating that LRP finds an explanation quicker with fewer features compared to gradient saliency. The same conclusion can be drawn from the results using the insertion procedure (Figure 4.5). Here, a rapid increase in average probability is observed after just inserting tens of features and reaching high performance after about 100 inserted features.



(a) Gradient saliency (b) LRP

Figure 4.4 – Deletion curves for gradient saliency and LRP.



(a) Gradient salicency (b) LRP

Figure 4.5 – Insertion curves for gradient saliency and LRP.

5 Experimental results: A case study on explaining natural language predictions

A common machine learning task within the area of natural language processing (NLP) is to have the AI system evaluate to what extent a text expresses negative, positive or neutral sentiment (i.e. *sentiment analysis*). A sentence such as "I'm so happy and grateful!" clearly expresses positive sentiment, whereas "I hope he meets his maker very soon" is clearly negative, and "He arrived yesterday" could be considered neutral. The positive example contains words that directly mark it out as positive, whereas the negative one requires a deeper understanding of language to be able to catch its decidedly hateful meaning. Thus, a text can vary both in terms of what kind and degree of sentiment it expresses, as well as how directly it does so. To understand how an AI system tries to make sense of the sentiment carried in texts that are fed to it, the same kind of techniques used to explain the classification of images in Chapter 4.2 can be applied.

5.1 The sentiment analysis predictor to be explained

The sentiment analysis model to be explained is a simplified version of the so called SentimentTagger model, which has primarily been used in-house to predict sentiment in tweets (that is, posts on Twitter). The SentimentTagger model consists of a combination of a DNN and a more traditional NLP module. In this work only the DNN part of the model was used. The DNN model was designed using a combination of recurrent (i.e. RNN) and fully connected (i.e. FCNN) neural network layers. The RNN part was implemented using a technique called long short-term memory (LSTM) that specializes in the modeling of distant dependencies between words or characters in a sentence, or even across sentences. For instance, in the sentence "The car, which I bought yesterday at a cheap price, broke down today.", the event "broke down" refers to "The car", even though they are separated by other text.

The particular LSTM used in SentimentTagger looks at the incoming text (a tweet) by breaking it down into its constituent letters. More precisely, it looks at characters, such as letters, but it also includes punctuation marks, whitespaces, emoticons, and so on. The model then extracts an intermediate representation that is good at modeling sentiment. This intermediate representation is then fed into the FCNN to produce the final sentiment prediction. The prediction is a continuous value between 0 and 1, where 0 is the most negative and 1 is the most positive. Hence, this is a regression model which in terms of explanation means that the explanation is not what contributed to predicting a particular class, but rather, what contributed to that particular output value.

The SentimentTagger prediction process is illustrated in Figure 5.1. Some examples of the model's predictions are also provided in Table 5.1. The predictions of the first three tweets in Table 5.1 agree well with the true sentiment value, as assigned by human judgment. The next three are examples of underestimation of positive sentiment, and the last three are examples of underestimation of negative sentiment. For some examples, such as number six, it might be argued that SentimentTagger is doing a better job than the human labelers. In all cases it would be beneficial to know what SentimentTagger based its estimation on.

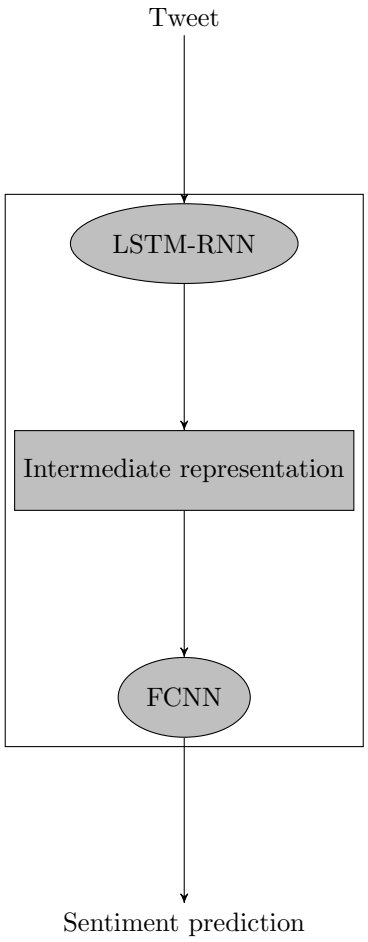


Figure 5.1 – Architecture of SentimentTagger. A tweet is fed into the LSTM-RNN, which produces an intermediate representation. This is then fed into the FCNN, which in turn produces the final sentiment prediction.

Table 5.1 – Examples of sentiment predictions by SentimentTagger on tweets.

Nr	Prediction	True value	Tweet
1	0.27	0.27	what do aquila, ajahnae, and euriechsa have in common besides ridiculously stupid,horrible,ugly, god awful names? tracey is not their father
2	0.37	0.32	@usr cn seems like you only appreciate good reviews. no apologies to my concern years ago.
3	0.71	0.72	i am always ready to smile.
4	0.40	0.84	3:45am and off to the hospital! elouise's waters have gone! labour littlesister superexcited
5	0.37	0.81	old bill free by the time the game with chelsea comes around? tears of joy tears of joy tears of joythat will be lively to say the leastafc
6	0.50	0.90	but the lady at the store said i had nice eyelashes so that is good, right? optimism? @usr
7	0.43	0.12	@usr norwegian quite simply the worst airline worstairline i have ever used! shocking appauling dire dismal beyondajoke useless
8	0.48	0.16	my irritation level is at an all time high today
9	0.47	0.15	these people irritate tf out of me i swear i am goin to sleep

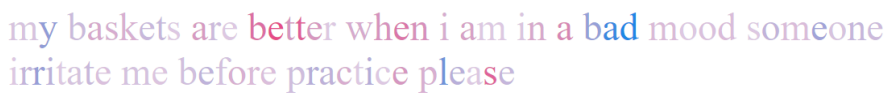
5.2 Explanation methods

To generate explanations for the predictions produced by SentimentTagger the model-agnostic LIME and SHAP techniques were applied. The version of SHAP (KernelSHAP) is actually a modification of LIME (according to the general formula proposed in [38]), which makes the comparison interesting. The reason for choosing model-agnostic methods in this case was that the concatenation of different types of neural networks makes applying model-specific methods non-trivial.

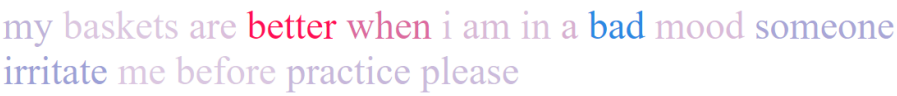
SentimentTagger analyzes tweets in terms of the characters they contain rather than at the word level. The most direct formulation of a saliency explanation is to indicate how much each character of a tweet contributed to the sentiment prediction on that tweet. An example of such an explanation is given in Figure 5.2a, where the tweet in question was predicted to have neutral sentiment (0.47) by SentimentTagger, whereas the value assigned by human judgment was slightly negative (0.31). So, what drove the prediction? Here, color coding is used to represent how each character contributed, either to increasing or decreasing the sentiment prediction. Blue means negative contributions (i.e. negative sentiment) and red means positive contributions (i.e. positive sentiment). Colors closer to transparent purple represent neutral sentiment.

The example seems to indicate that characters in the word “better” contributed positively and that characters in “bad” contributed negatively, whereas other characters provided a less clear picture. It is difficult to draw conclusions from saliency attributions to individual characters, since characters by themselves do not really mean anything. Consequently, while there may be reasons to have a sentiment prediction model work at the character level, explanations should probably be provided at an aggregated level that maps better to actual meaning.

If the character-level attributions are aggregated to each word containing the respective characters, the result is instead the visualization that is shown in Figure 5.2b. The picture that emerges is much clearer, and not only is it possible to see how strongly “better” and “bad” are driving the prediction, but the slightly positive role of “when” and the slightly negative role of “someone” and “irritate” can also be discerned. Finally, it can also be noted that the amounts of “redness” and “blueness” seem roughly equivalent, which explains why SentimentTagger decided on a neutral evaluation of sentiment. In the examples that follow, explanations aggregated to the word level are visualized, as in this case.



(a) Character level saliency



(b) Word level saliency

Figure 5.2 – A tweet where characters and words have been color coded according to their contribution to the sentiment prediction on that tweet. Red means contribution towards positive sentiment; blue means contribution towards negative sentiment. In this case the model predicted neutral sentiment (0.47), whereas the sentiment was judged slightly negative (0.31) by human labelers. The word level saliency visualization appears to more clearly map the importance of the semantics of the sentence.

5.3 Qualitative results

Table 5.2 shows nine examples of tweets that are interesting in different ways. The colors correspond to saliency attributions made by SHAP (and then aggregated to word level as explained above). The *Prediction* column lists the sentiment value predicted by SentimentTagger, and the *True value* column shows the value assigned by human judgment. The more red a word, the more characters composing it have collectively contributed to pushing the predicted value *up*. Conversely, the more blue a word, the more its characters have pushed the predictions *down*.

For the first three tweets, SentimentTagger agrees fairly well in its predictions with human judgment of sentiment (in the *True value* column). Despite the agreement, it is interesting to see what words SentimentTagger looked at to arrive at its predictions. In the first tweet, "stupid", "horrible", "ugly", "awful" and "not" drive the sentiment in a negative direction, but the word "father" is an even stronger negative driver. It could be queried whether SentimentTagger picked up on the combination of negative adjectives with "father", or if it is sufficiently sophisticated as to identify "not their father" as a damning statement. The third tweet is much clearer; "smile" does most of the positive work.

Tweets 4 and 5 are examples of SentimentTagger assigning negative sentiment to tweets that are actually quite positive. Words such as "hospital", "gone" and "tears" have been superficially interpreted as negative, whereas a proper understanding of context would negate that judgment. Tweet number 6 seems to show the same discrepancy between prediction and true sentiment. However, it could be argued that the insecurity expressed by the interrogative form actually has the prediction being closer to the truth than the assigned label.

In examples 7 through 9 the relationship is reversed, in that the mellow predictions have grossly underestimated the degree of *negativity* expressed in the tweets. Some missed negativity could perhaps stem from misspelled words, such as "appauling" (7), missing whitespace, such as "worstairline" and "beyondajoke" (7), and colloquial abbreviations, such as "tf" (9), even though a character level LSTM would be expected to deal better with minor misspellings and missing whitespaces than a word-level one. Other mistakes are more difficult to explain, such as "shocking", "dire", "dismal" (7) and "irritate" (9). Example 8 seems to indicate that SentimentTagger has missed the connection between "irritation" and "high".

Saliency attributions produced by LIME for the same tweets are shown in Table 5.3. While the SHAP attributions are largely intelligible, while not perfectly matching intuition, the LIME versions are largely perplexing. A few correspond to intuition, such as "ridiculously" (1), "appreciate" (2) and "optimism" (6), of which the first two were not highlighted by SHAP. Some directly contradict intuition, for example "useless" (7) and "smile" (3), the second also contradicting SHAP. However, most just appear arbitrary, such as "ajahnae" (1), "ago" (2), "will" (5) and "today" (8). Do these unintuitive explanations indicate faults in SentimentTagger that SHAP does not find, or are the more intuitive attributions by SHAP more accurate descriptions of what the LSTM is actually doing? The fact that KernelSHAP is a theoretically better grounded version of LIME would indicate the latter, but these qualitative results cannot provide any proof to that effect. For a more objective comparison between the two explanation methods, a quantitative analysis is deployed in the next section.

Table 5.2 – Selected tweets, colored by SHAP saliency values that have been aggregated to word level.

Nr	Prediction	True value	Tweet
1	0.27	0.27	what do aquila, ajahnae, and euriechsa have in common besides ridiculously stupid,horrible,ugly, god awful names? tracey is not their father
2	0.37	0.32	@usr cn seems like you only appreciate good reviews. no apologies to my concern years ago.
3	0.71	0.72	i am always ready to smile.
4	0.40	0.84	3:45am and off to the hospital! elouise's waters have gone! labour littlesister superexcited
5	0.37	0.81	old bill free by the time the game with chelsea comes around? tears of joy tears of joy tears of joythat will be lively to say the leastafc
6	0.50	0.90	but the lady at the store said i had nice eyelashes so that is good, right? optimism? @usr
7	0.43	0.12	@usr_norwegian quite simply the worst airline worstairline i have ever used! shocking appauling dire dismal beyondajoke useless
8	0.48	0.16	my irritation level is at an all time high today
9	0.47	0.15	these people irritate tf out of me i swear i am goin to sleep

Table 5.3 – Selected tweets, colored by LIME saliency values that have been aggregated to word level.

Nr	Prediction	True value	Tweet
1	0.27	0.27	what do aquila, ajahnae, and euriechsa have in common besides ridiculously stupid,horrible,ugly, god awful names? tracey is not their father
2	0.37	0.32	@usr cn seems like you only appreciate good reviews. no apologies to my concern years ago.
3	0.71	0.72	i am always ready to smile.
4	0.40	0.84	3:45am and off to the hospital! elouise's waters have gone! labour littlesister superexcited
5	0.37	0.81	old bill free by the time the game with chelsea comes around? tears of joy tears of joy tears of joythat will be lively to say the leastafc
6	0.50	0.90	but the lady at the store said i had nice eyelashes so that is good, right? optimism? @usr
7	0.43	0.12	@usr_norwegian quite simply the worst airline worstairline i have ever used! shocking appauling dire dismal beyondajoke useless
8	0.48	0.16	my irritation level is at an all time high today
9	0.47	0.15	these people irritate tf out of me i swear i am goin to sleep

5.4 Feature deletion analysis

As explained in Section 4.2.1, the deletion metric tests the performance of an explanation method by removing features in order of the saliency attributed to them by the XAI technique. A good XAI technique should attribute high saliency to features that are important to the prediction model's output, so that removing features in that order causes the model's performance to drop sharply. In the present case, the features to be deleted in order of saliency are characters, and deleting a feature in this case means replacing it with an empty character, such as a tab or a whitespace, and the prediction model is SentimentTagger. We applied deletion tests to both SHAP explanations and LIME explanations of SentimentTagger on a batch of 500 tweet examples, and then plotted how the prediction performance of the model falls as a function of the number of features (characters) deleted. Additionally, as a baseline, we performed deletion with a random mask, causing features to be deleted in random order. Since SentimentTagger is a regression model, its performance cannot be measured in accuracy. Instead, the R^2 metric that represents a measure of how well the trained model explains the variance in the test data was used.

Figure 5.3 shows the R^2 performance of SentimentTagger as a function of the number of deletions, as ordered by SHAP, LIME and the random mask respectively. It is immediately apparent that deletion tests favor SHAP, as its curve rapidly drops in order of its saliency attributions, whereas the corresponding curve for LIME is markedly less steep. LIME performs only slightly better than the random mask on the deletion test. SHAP consequently seems to be doing a better job at identifying a handful of features (characters) without which the model is at a loss to predict anything accurately. This is perhaps not surprising, as the Shapley formula is designed to do just that, whereas LIME relies on more technically motivated heuristics. The fact that it is even possible to completely extinguish SentimentTagger's performance by selectively replacing just a handful of characters with whitespaces may, however, be an interesting indication about the robustness (or lack thereof) of such models.

What is further noteworthy is that the R^2 values in the SHAP case actually drop below zero after the first ten or so deletions, and then move back towards zero as more deletions are performed. This means that the first set of deletions are actually causing the model to perform *worse* than a model that ignores its input and always makes the same prediction. As even more deletions are made, the model's predictions will converge towards the neutral prediction that corresponds to an empty tweet, which is equivalent to ignoring the input. Thus, the R^2 value converges back to zero.

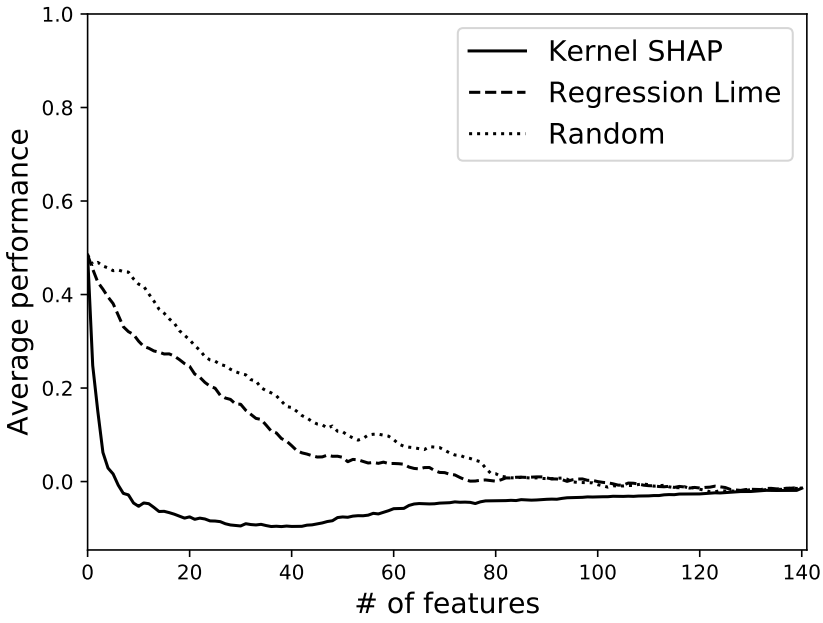


Figure 5.3 – Deletion analysis of SHAP and LIME explanations of the SentimentTagger predictions. Deletion in random order is used as a baseline. The graph plots the effect on model performance, as measured by the R^2 metric, as features are successively deleted (that is, characters are successively blanked out) in order of saliency. The steep initial fall of the SHAP curve indicates that SHAP is good at finding which features are most critical for model performance. The dip below zero indicates that strategic deletions can cause the model to make sentiment predictions that tend to *contradict* the true sentiment as labeled by humans. The slow decline by LIME indicates that LIME saliency values are less adept than SHAP values at finding which features are most critical for model performance, and only slightly better than random deletion.

6 Conclusions

Deep learning will be used to complement and replace functions in military systems. In fact, DL techniques are already being evaluated in military surveillance systems to automatically detect and track objects of interest in large volumes of imagery data [45]. There are several advantages with DL compared to traditional software techniques. Most importantly, DL can be used to model processes that are too complex to model using traditional software techniques. It can also facilitate active learning, where an AI system interacts with its users to acquire high-quality data that can be used to enhance the model in an operative system (i.e. after deployment).

Unfortunately, these advantages also introduce major challenges that need to be addressed, not only technically but also operationally. In this report, the focus was on the challenge of explainability. A major drawback of DL is that even though learning algorithms, model architecture and training data are known and well understood, the behavior of the model itself is not. This is typically not a problem in many real world civilian applications used for music recommendation and advertisement purposes. However, in the military domain the ability to understand and explain the behaviors of AI systems is critical. In this context, the decisions and recommendations provided by the AI systems may have a deep impact on human lives. This is valid at the tactical level where autonomous weapons and drones are used, as well as at the operational and the strategic level where long-term decisions are made by military leaders and political decision makers.

It might be argued that complex military systems such as fighter jets, submarines, tanks and decision support tools for command and control are also difficult to grasp. Although this is true, the techniques used to build these systems are inherently interpretable. Hence, if something goes wrong it is possible to systematically inspect the system to identify and correct the problem. This is not the case in DL. The main reason is that DNNs in real world applications often consist of millions or even billions of parameters. Hence, not even the creators of these models are capable of systematically addressing errors that may exist in the model.

In this report, several state-of-the-art XAI techniques that have been proposed to address the challenge of explainability were explored. Although some progress has been made, it can be concluded that XAI for DL applications in the military domain is still in its infancy. Ultimately, even though many XAI techniques have been proposed, they have not yet been evaluated in a military context. Hence, there is no guarantee that existing XAI techniques will enable the use of DL in high-stake military AI systems.

When developing AI systems for military purposes it is our recommendation that explainability and interpretability requirements are identified early in the acquisition and development processes. Foremost, it is important that such requirements are defined so that they are feasible and verifiable. That is, the requirements must be in line with what is practically possible to expect in terms of explainability.

In future work, we intend to develop an evaluation framework that can be used to support the development of XAI capabilities in military AI systems.

Appendix A Gradient descent optimization with backpropagation

This appendix demonstrates the training process described in Section 2.3.3. The model used in this example is represented by a computational graph as illustrated in Figure A.1. In this example, $f_\theta(x) = \omega x + b$ and $\theta = \{\omega, b\}$ represent the model and its trainable parameters respectively. x and $\hat{y}$ represent the input and its desired output (i.e. training data). The computational graph represents the squared error loss function $\mathcal{L} = (f_\theta(x) - \hat{y})^2$.

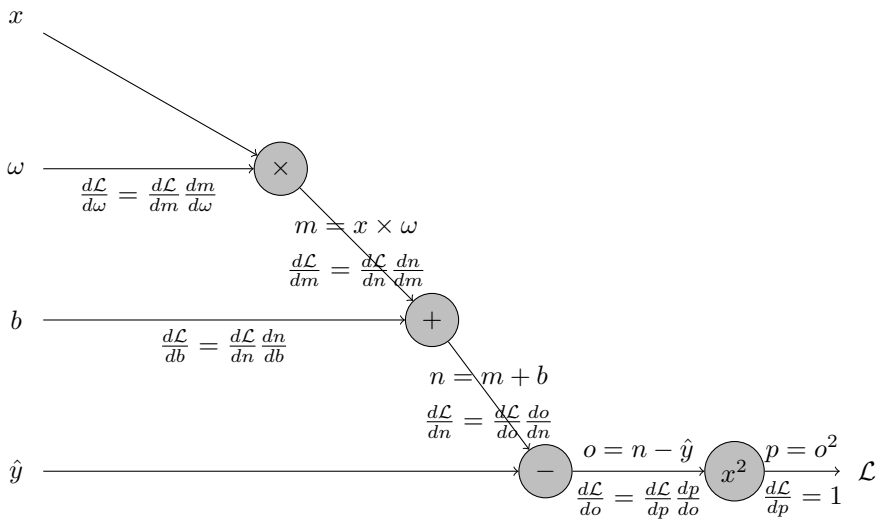


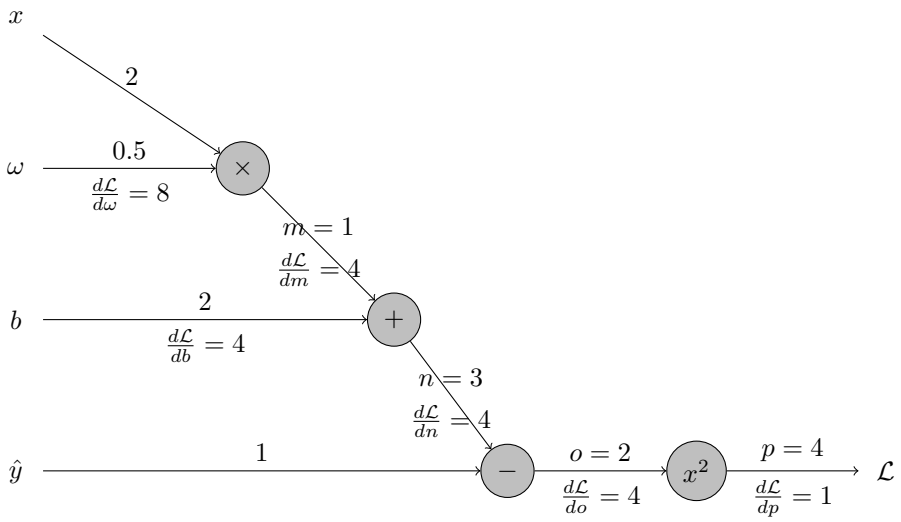
Figure A.1 – Computational graph representing the squared error loss function, $\mathcal{L} = (f_\theta(x) - \hat{y})^2$.

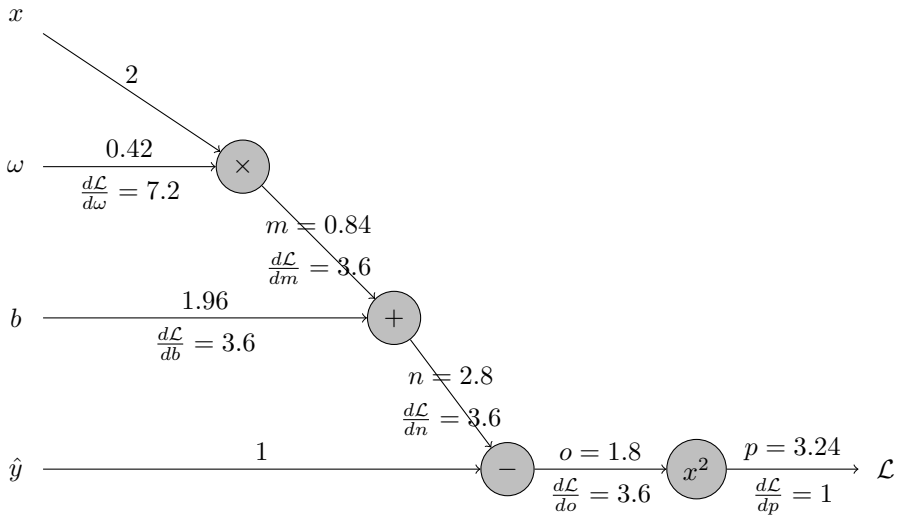
The general derivative rules for the subfunctions used in the graph are presented in Table A.1. Note that it is only the derivative of the loss with respect to the trainable parameters that are needed for training (i.e. $\nabla_\theta \mathcal{L}(f_\theta) = \{\frac{d\mathcal{L}}{d\omega}, \frac{d\mathcal{L}}{db}\}$). The backpropagation starts by setting $\frac{d\mathcal{L}}{dp} = 1$. From there it is easy to see how the chain rule propagates the error backwards (right to left) to find $\frac{d\mathcal{L}}{d\omega}$ and $\frac{d\mathcal{L}}{db}$.

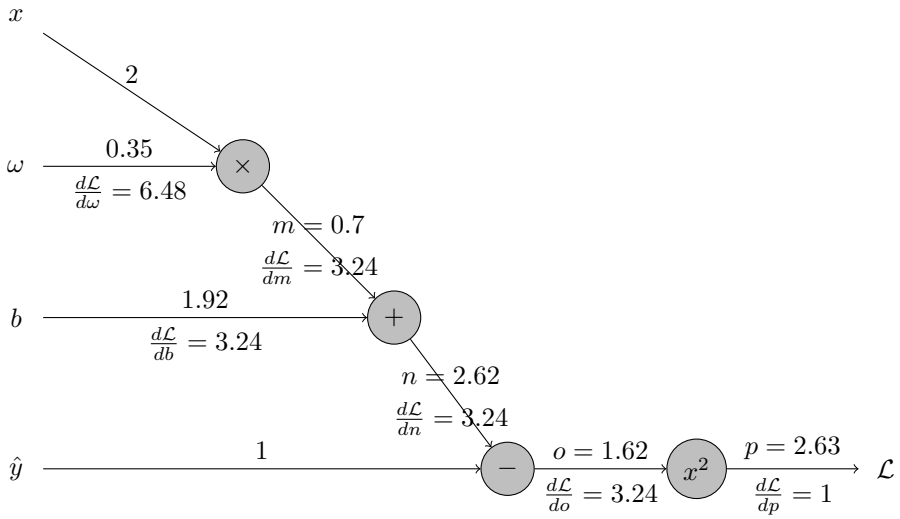
To demonstrate the training process, several iterations of GD was performed as illustrated below. In this case, training is performed with the settings: $\{x, \hat{y}\} = \{2, 1\}$, $\{\omega, b\} = \{0.5, 2\}$ and $\alpha = 0.01$. Note that the loss decreases in every iteration. This is an indicator that the DNN is learning.

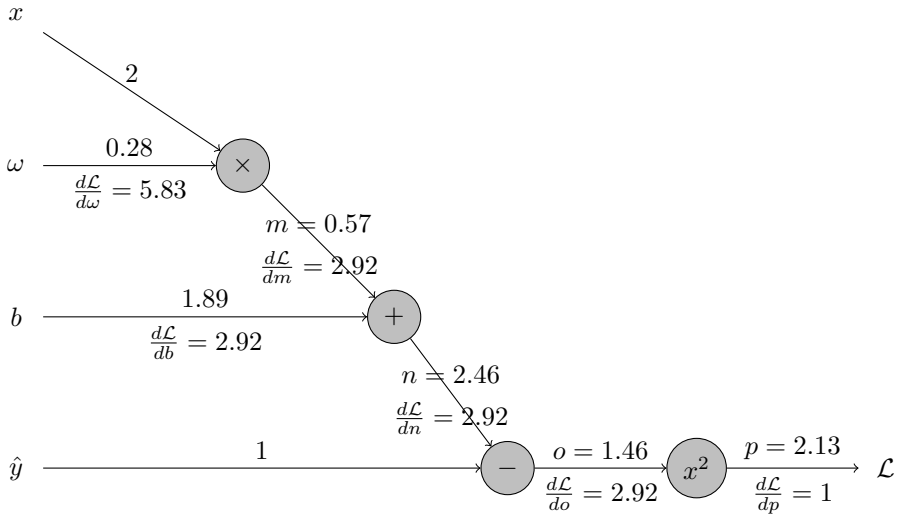
Table A.1 – General derivative rules for the subfunctions used in the computational graph in Figure 2.8. The derivatives of the subfunctions are identified with respect to each input (i.e. partial derivatives).

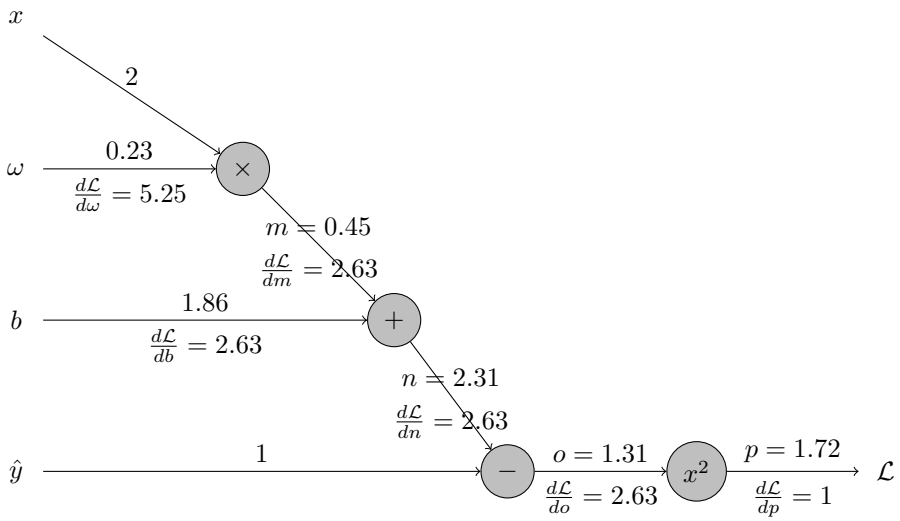
Operator	**Function**	**Partial derivatives**
$\times$	$f(x_1, x_2) = x_1 \times x_2$	$\frac{df}{dx_1} = x_2$ $\frac{df}{dx_2} = x_1$
$+$	$f(x_1, x_2) = x_1 + x_2$	$\frac{df}{dx_1} = 1$ $\frac{df}{dx_2} = 1$
$-$	$f(x_1, x_2) = x_1 - x_2$	$\frac{df}{dx_1} = 1$ $\frac{df}{dx_2} = -1$
x^2	$f(x) = x^2$	$\frac{df}{dx} = 2x$






x
2
ω
0.35
$\frac{d\mathcal{L}}{d\omega} = 6.48$
$\times$
$m = 0.7$
$\frac{d\mathcal{L}}{dm} = 3.24$
b
1.92
$\frac{d\mathcal{L}}{db} = 3.24$
$+$
$n = 2.62$
$\frac{d\mathcal{L}}{dn} = 3.24$
$\hat{y}$
1
$-$
$o = 1.62$
$\frac{d\mathcal{L}}{do} = 3.24$
x^2
$p = 2.63$
$\frac{d\mathcal{L}}{dp} = 1$
$\mathcal{L}$


x
2
ω
0.28
$\frac{d\mathcal{L}}{d\omega} = 5.83$
$\times$
$m = 0.57$
$\frac{d\mathcal{L}}{dm} = 2.92$
b
1.89
$\frac{d\mathcal{L}}{db} = 2.92$
$+$
$n = 2.46$
$\frac{d\mathcal{L}}{dn} = 2.92$
$\hat{y}$
1
$-$
$o = 1.46$
$\frac{d\mathcal{L}}{do} = 2.92$
x^2
$p = 2.13$
$\frac{d\mathcal{L}}{dp} = 1$
$\mathcal{L}$


x
2
ω
0.23
$\frac{d\mathcal{L}}{d\omega} = 5.25$
$\times$
$m = 0.45$
$\frac{d\mathcal{L}}{dm} = 2.63$
b
1.86
$\frac{d\mathcal{L}}{db} = 2.63$
$+$
$n = 2.31$
$\frac{d\mathcal{L}}{dn} = 2.63$
$\hat{y}$
1
$-$
$o = 1.31$
$\frac{d\mathcal{L}}{do} = 2.63$
x^2
$p = 1.72$
$\frac{d\mathcal{L}}{dp} = 1$
$\mathcal{L}$

Bibliography

[1] Government Offices of Sweden. National approach to artificial intelligence. https://www.regeringen.se/4aa638/contentassets/a6488ccebc6f418e9ada18bae40bb71f/national-approach-to-artificial-intelligence.pdf, 2018. [Online; accessed 11-september-2019].

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep Speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, volume 48, pages 173–182, 2016.

[3] Amazon. Amazon transcribe - automatically convert speech to text. https://aws.amazon.com/transcribe/, 2019. [Online; accessed 11-september-2019].

[4] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv e-prints*, page arXiv:1609.08144, 2016.

[5] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv e-prints*, page arXiv:1609.03499, 2016.

[6] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575, 11 2019.

[7] DeepMind. AlphaStar: Mastering the real-time strategy game StarCraft II. https://deepmind.com/blog/article/alphastar-mastering-

real-time-strategy-game-starcraft-ii, 2019. [Online; accessed 11-september-2019].

[8] Adriana Fernandez-Lopez and Federico M. Sukno. Survey on automatic lip-reading in the era of deep learning. *Image and Vision Computing*, 78:53–72, 2018.

[9] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1701–1708, 2014.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv e-prints*, page arXiv:1604.07316, 2016.

[11] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860, 2017.

[12] Sandeep Neema. Assured autonomy. https://www.darpa.mil/program/assured-autonomy, 2017. [Online; accessed 04-september-2019].

[13] John D. Lee and Katrina A. See. Trust in automation: Designing for appropriate reliance. *Human Factors*, 46(1):50–80, 2004.

[14] Robert R. Hoffman, Tim Miller, Shane T. Mueller, Gary Klein, and William J. Clancey. Explaining explanation, part 4: A deep dive on deep nets. *IEEE Intelligent Systems*, 33(3):87–95, 2018.

[15] Matt Turek. Explainable artificial intelligence (XAI). https://www.darpa.mil/program/explainable-artificial-intelligence, 2017. [Online; accessed 04-september-2019].

[16] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. iNNvestigate neural networks! *arXiv e-prints*, page arXiv:1808.04260, 2018.

[17] Raphael Meudec. Introducing tf-explain, interpretability for TensorFlow 2.0. https://blog.sicara.com/tf-explain-interpretability-tensorflow-2-9438b5846e35, 2019. [Online; accessed 11-september-2019].

[18] Facebook Inc. Captum: Model interpretability for PyTorch. https://www.captum.ai/, 2019. [Online; accessed 28-october-2019].

[19] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[20] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. Explainable AI: the new 42? In *2nd International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE)*, volume LNCS-11015 of *Machine Learning and Knowledge Extraction*, pages 295–303, 2018.

[21] Shane T. Mueller, Robert R. Hoffman, William Clancey, Abigail Emrey, and Gary Klein. Explanation in human-AI systems: A literature meta-review, synopsis of key ideas and publications, and bibliography for explainable AI. *arXiv e-prints*, page arXiv:1902.01876, 2019.

[22] Robert R. Hoffman, Shane T. Mueller, Gary Klein, and Jordan Litman. Metrics for explainable AI: Challenges and prospects. *arXiv e-prints*, page arXiv:1812.04608, 2018.

[23] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus Muller. Unmasking Clever Hans predictors and assessing what machines really learn. *Nature Communications*, 10(1), 2019.

[24] Kristof T. Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus Muller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature Communications*, 8, 2017.

[25] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017.

[26] Kelley M. Sayler. Defense primer: U.s. policy on lethal autonomous weaponsystems. *Congressional Research Service, IF11150, Version 1*, pages 1–1, 2019.

[27] U.S. Government. Humanitarian benefits of emerging technologies in the area of lethal autonomous weapon systems. *CCW/CGE.1/2018/WP.4*, pages 1–6, 2018.

[28] Yun He, Soma Shirakabe, Yutaka Satoh, and Hirokatsu Kataoka. Human action recognition without human. *European Conference on Computer Vision (ECCV) Workshop: Brave New Idea*, 2016.

[29] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[31] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9:2579–2605, 2008.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[33] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *International Conference on Learning Representations (ICLR)*, 2014.

[34] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[35] Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[36] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE 10*, 7:1–46, 2015.

[37] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 193–209. 2019.

[38] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4765–4774, 2017.

[39] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[40] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014.

[41] Stan Lipovetsky and Michael Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.

[42] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[43] Vitali Petsiuk, Abir Das, and Kate Saenko. RISE: Randomized input sampling for explanation of black-box models. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2018.

[44] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[45] Cheryl Pellerin. Project maven industry day pursues artificial intelligence for DoD challenges. `https://www.defense.gov/Newsroom/News/Article/Article/1356172/project-maven-industry-day-pursues-artificial-intelligence-for-dod-challenges/`, 2017. [Online; accessed 24-september-2019].

FOI, Swedish Defence Research Agency, is a mainly assignment-funded agency under the Ministry of Defence. The core activities are research, method and technology development, as well as studies conducted in the interests of Swedish defence and the safety and security of society. The organisation employs approximately 1000 personnel of whom about 800 are scientists. This makes FOI Sweden's largest research institute. FOI gives its customers access to leading-edge expertise in a large number of fields such as security policy studies, defence and security related analyses, the assessment of various types of threat, systems for control and management of crises, protection against and management of hazardous substances, IT security and the potential offered by new sensors.

FOI
Defence Research Agency
SE-164 90 Stockholm

Phone: +46 8 555 030 00
Fax: +46 8 555 031 00

www.foi.se