Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-182905-72287

Bc. Branislav Pecher

Interpretability of Neural Network Models Used in Data Analysis

Master's Thesis

Supervisor: Ing. Jakub Ševcech, PhD.

April 2019

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-182905-72287

Bc. Branislav Pecher

Interpretability of Neural Network Models Used in Data Analysis

Master's Thesis

Study program: Intelligent Software Systems
Field of Study: 9.2.5 Software engineering, 9.2.8 Artificial Intelligence
Place: Institute of Informatics, Information Systems and Software Engineering
Supervisor: Ing. Jakub Ševcech, PhD.

April 2019

DECLARATION

I hereby declare that this thesis was composed by me with the help of literature listed in this work and the consultations with my supervisor.

Bratislava, April 2019 .....

Bc. Branislav Pecher

ACKNOWLEDGMENTS

I would like to thank Ing. Jakub Ševcech, PhD. for his guidance and help in developing this master's thesis. I would also like to thank my family and friends for their patience and support during the development of this work.

STU FIIT
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Zadanie diplomovej práce

Meno študenta: **Bc. Branislav Pecher**

Študijný program: Inteligentné softvérové systémy

Študijný odbor: Softvérové inžinierstvo – hlavný študijný odbor
Umelá inteligencia – vedľajší študijný odbor

Názov práce: **Interpretovateľnosť modelov neurónových sietí využívaných na analýzu údajov**

Samostatnou výskumnou a vývojovou činnosťou v rámci predmetov Diplomový projekt I, II, III vypracujte diplomovú prácu na tému, vyjadrenú vyššie uvedeným názvom tak, aby ste dosiahli tieto ciele:

Všeobecný cieľ:

Vypracovaním diplomovej práce preukážte, ako ste si osvojili metódy a postupy riešenia relatívne rozsiahlych projektov, schopnosť samostatne a tvorivo riešiť zložité úlohy aj výskumného charakteru v súlade so súčasnými metódami a postupmi študovaného odboru využívanými v príslušnej oblasti a schopnosť samostatne, tvorivo a kriticky pristupovať k analýze možných riešení a k tvorbe modelov.

Špecifický cieľ:

Vytvorte riešenie zodpovedajúce návrhu textu zadania, ktorý je prílohou tohto zadania. Návrh bližšie opisuje tému vyjadrenú názvom. Tento opis je záväzný, má však rámcový charakter, aby vznikol dostatočný priestor pre Vašu tvorivosť.

Riaďte sa pokynmi Vášho vedúceho.

Pokiaľ v priebehu riešenia, opierajúc sa o hlbšie poznanie súčasného stavu v príslušnej oblasti, alebo o priebežné výsledky Vášho riešenia, alebo o iné závažné skutočnosti, dospejete spoločne s Vaším vedúcim k presvedčeniu, že niečo v texte zadania a/alebo v názve by sa malo zmeniť, navrhnite zmenu. Zmena je spravidla možná len pri dosiahnutí kontrolného bodu.

Miesto vypracovania: Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU v Bratislave

Vedúci práce: **Ing. Jakub Ševcech, PhD.**

Termíny odovzdania:

Podľa harmonogramu štúdia platného pre semester, v ktorom máte príslušný predmet (Diplomový projekt I, II, III) absolvovať podľa Vášho študijného plánu

Predmety odovzdania:

V každom predmete dokument podľa pokynov na www.fiit.stuba.sk v časti: home > Informácie o > štúdiu > harmonogram štúdia > diplomový projekt.

V Bratislave dňa 12. 2. 2018

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4
1

prof. Ing. Pavol Návrat, PhD.
riaditeľ Ústavu informatiky, informačných systémov
a softvérového inžinierstva

STU FIIT
SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Návrh zadania diplomovej práce

Finálna verzia do diplomovej práce [1]

Študent:

Meno, priezvisko, tituly: Branislav Pecher, Bc.
Študijný program: Inteligentné softvérové systémy
Kontakt: branop95@gmail.com

Výskumník:

Meno, priezvisko, tituly: Jakub Ševcech, Ing. PhD.

Projekt:

Názov: Interpretovateľnosť modelov neurónových sietí využívaných na analýzu údajov

Názov v angličtine: Interpretability of neural network models used in data analysis

Miesto vypracovania: Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU, Bratislava

Oblasť problematiky: strojové učenie, neurónové siete

Text návrhu zadania[2]

Modely neurónových sietí sa v súčasnosti využívajú v mnohých oblastiach, ako je napríklad medicína a bankovníctvo. Pri typickom používaní sa snažíme vytvoriť také modely, ktoré dokážu svoju úlohu riešiť čo najpresnejšie a teda s čo najmenšou chybou. Avšak presnosť nie je všetko, lebo takýmto spôsobom častokrát vznikajú modely, ktoré sa ľuďom javia ako čierne skrinky, ktorým nedôverujú. Ak im nie sme schopní vysvetliť ako sa náš model dopracoval k výsledku, tak ľudia môžu mať problém s jeho akceptovaním a používaním a uprednostnia jednoduchší, ale nepresnejší model. Napríklad interpretovať modely neurónových sietí na základe váh prepojení medzi neurónmi je veľmi ťažké, hlavne pri sieťach s vyšším množstvom skrytých vrstiev.

Analyzujte existujúce prístupy, ktoré sa používajú na vysvetlenie a zdôvodnenie výsledkov produkovaných modelmi strojového učenia bez závislosti od použitého modelu a na prístupy, ktoré sa zameriavajú na zlepšenie interpretovateľnosti výsledkov modelov neurónových sietí. Navrhnite metódu na analýzu údajov využívajúcu neurónové siete, pričom sa zamerajte na jej interpretovateľnosť a jednoduchosť vysvetliteľnosti. Navrhnutú metódu overte a demonštrujte jej použitie na modeli netriviálnej zložitosti.

---

[1] Vytlačiť obojstranne na jeden list papiera

[2] 150-200 slov (1200-1700 znakov), ktoré opisujú výskumný problém v kontexte súčasného stavu vrátane motivácie a smerov riešenia

Literatúra[3]

- 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, KDD '16, pages 1135–1144, New York, NY, USA, 2016. ACM.
- Montavon, G., Samek, W., and Müllerr, K.-R. (In press). Methods for interpreting and understanding deep neural networks. Digital Signal Processing, 73(Supplement C):1 – 15. DOI: 10.1016/j.dsp.2017.10.011. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1051200417302385.

Vyššie je uvedený návrh diplomového projektu, ktorý vypracoval(a) Bc. Branislav Pecher, konzultoval(a) a osvojil(a) si ho Ing. Jakub Ševcech, PhD. a súhlasí, že bude takýto projekt viesť v prípade, že bude pridelený tomuto študentovi.

V Bratislave dňa 7.1.2018

Pecher
Podpis študenta

Podpis výskumníka

Vyjadrenie garanta predmetov Diplomový projekt I, II, III

Návrh zadania schválený: áno / ~~nie~~[4]

Dňa: 12. 2. 2018

Podpis garanta predmetov

[3] 2 vedecké zdroje, každý v samostatnej rubrike a s údajmi zodpovedajúcimi bibliografickým odkazom podľa normy STN ISO 690, ktoré sa viažu k téme zadania a preukazujú výskumnú povahu problému a jeho aktuálnosť (uveďte všetky potrebné údaje na identifikáciu zdroja, pričom uprednostnite vedecké príspevky v časopisoch a medzinárodných konferenciách)

[4] Nehodiace sa prečiarknite

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Inteligentné Softvérové Systémy
Autor: Bc. Branislav Pecher
Diplomová práca: Interpretovateľnosť modelov neurónových sietí využívaných na analýzu údajov
Vedúci práce: Ing. Jakub Ševcech, PhD.
December 2019

Hlboké neurónové siete sa v posledných rokoch stávajú najpopulárnejším modelom pre riešenie rôznych úloh, ako sú napríklad klasifikácia obrázkov a textu, alebo určovanie sentimentu. Vo svojej podstate sú hlboké neurónové siete schopné sa naučiť reprezentáciu vysoko nelineárnych a komplexných funkcií, pričom naučené znalosti ukladajú vo forme váh naprieč mnohými vrstvami. To zapríčiňuje, že je ťažké porozumieť jednotlivým rozhodnutiam, ktoré boli spravené. Preto sú neurónové siete považované za čiernu skrinku, ktorej chýba transparentnosť, a preto ich prijatie a následné využívanie je sťažené v doménach, kde je cena chyby vysoká, ako je bankovníctvo alebo zdravotníctvo, keďže takýmto modelom je ťažké veriť. Aby sa zvýšila dôvera ľudí v modely hlbokých neurónových sietí, je podstatné, aby modely vedeli svoje rozhodnutia vysvetlovať. Tieto dôvody vedú k novým oblastiam výskumu v rámci hlbokého učenia. Jednou z napopulárnejších metód využívaných na vysvetlovanie rozhodnutí neurónových sietí je určovanie významnosti vstupných čŕt, vyjadrenej číslom, použitím atribučných metód. V tejto diplomovej práci sa zaoberáme návrhom novej atribučnej metódy založenej na zavedení porúch do dát, ktorá dokáže zohľadniť interakcie medzi jednotlivými črtami v dátach. To bude mať za následok, že dôvera v modely natrénované nad dátami so závislosťami sa zvýši. Zameriavame sa hlavne na vysvetlovanie rozhodnutí v doméne textu, ktorý je reprezentovaný za pomoci vnárania slov, tým, že sa pozorujú zmeny vo výstupe keď sú niektoré vstupné črty narušené. Rôzne prístupy na určovanie interakcie medzi slovami a na zavádzanie porúch do textu sú vyskúšané a overené. Výsledky z našich experimentov nám naznačujú, že naša metóda je lepšia v identifikovaní dôležitých čŕt, ktorých dôležitosť je inak schovaná kvôli interakciám v dátach. Avšak, v prípade keď chceme nájsť iba tú najdôležitejšiu črtu, alebo zopár takýchto čŕt, naša navrhnutá metóda nie je najlepšou voľbou.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Intelligent Software Systems
Author: Bc. Branislav Pecher
Master thesis: Interpretability of Neural Network Models Used in Data Analysis
Supervisor: Ing. Jakub Ševcech, PhD.
April 2019

In the last few years, the deep neural networks are emerging as the most popular method for many tasks, such as image and text classification or sentiment detection. Basically, the deep neural networks are able to learn to represent highly non-linear and complex functions, with its knowledge saved in weight across multitude of layers. Therefore, it is hard to understand specific decisions made. This makes the neural networks a black box models that lack transparency, and therefore their adoption and use in areas where the cost of errors is high, like banking or healthcare, is impeded due to low levels of trust. In order to increase the trust in deep neural network models and their use in industry, it is critical for models to be able to explain their decisions. These reasons lead to new research areas in the field of deep learning. One of the most popular approaches for explaining the decisions of neural network is using attribution methods to assign relevance scores to input features. In this master thesis, we propose a new perturbation-based attribution method that, unlike other attribution methods, can take into consideration the inherent interactions present in data. Thus, increasing the trust in model trained even on data with interactions. We are focusing on explaining the decision in domain of text with word embedding representation by observing the changes in output when the input features are perturbed. Different approaches for determining the interactions between words and for perturbing the text are explored and evaluated. The results from our experiments indicate that our method is better for finding the important features that can otherwise be hidden due to the interactions. However, when we want to find the most important feature, or a small number of them, our proposed method performs rather poorly.

Contents

Chapter 1

Introduction

Machine learning models are used in many different disciplines, like medicine for detecting tumors or other diseases, or in banking for detecting fraud or deciding, if the person is eligible for mortgage. When creating machine learning models, we aim to achieve the highest possible accuracy to guarantee that there would be no errors present and so can be used freely.

However, to use the machine learning models, we should be able to tell why the decision was made by the model. This way we ensure that the model is behaving like it should and is not prone to making a mistake. This is really important in high risk environments, like medicine, where a simple mistake can have severe consequence. If we cannot explain why the machine learning model is behaving the way it is, we cannot build trust in it and people will be inclined to use other models, for which the explanations can be provided even though they achieve lower accuracy.

Interpretability matters [12]. Only with interpretability can machine learning algorithms be debugged and audited. So even in low risk environments, like movie recommendation, interpretability in the research and development stage as well as after deployment is valuable. Having an interpretation for a prediction that went bad helps us understand the cause and fix it. There are many recorded instances when machine learning models misclassified some instances, because it was trained to look for things that are irrelevant in the decision process. For example, instead of looking for differences between dogs and wolves, the model can learn to look for snow in the image. Most of these mistakes are due to the faults in the training process or the training data [12].

Of course, we do not need an explanation for everything that happens, but mostly for the unexpected events. In addition, not all machine learning models need methods for interpretability, as they are easy to understand as they are. One such machine learning model is the decision tree that generates simple rules that are interpretable for us. On the other hand, there are many models that are hard to interpret. They are viewed as black boxes that no one understands and therefore are not trusted.

One of the hardest models to interpret, but also one that can achieve highest performance on most of the machine learning tasks, are the neural network models [17, 41]. Their accuracy, but also hard interpretability, is due to the fact that they perform highly non-linear transformations in the data and all the knowledge is saved in the connection weights in the neural networks.

Due to the recent advances in the deep learning, which caused the neural networks to have much more neurons and much more hidden layer, the need for methods for an easy interpretation of neural network decisions is increased [17]. Increasing the accuracy of the neural networks without the proper tools for interpreting their decisions is useless, as that way they will never be used in the production environment. Many methods for providing the explanations of the neural network decision have already been introduced. However, most of them are unusable for the deep neural network models. Therefore, the main focus of the research in the deep learning area is to develop a fast method that provides accurate explanations for neural networks decisions.

One of the most used techniques for explaining the decisions of neural networks are the attribution methods. The attribution method assigns one number, an attribution, to each input feature. This attribution represents a relevance, or a contribution, of the input feature towards a specific decision. These relevance scores can be either positive, reflecting that the feature supports the decision, or negative, reflecting that the feature is suppressing the decision. When arranged together, the relevance scores produce an attribution map that can be easily visualised in a form of a heatmap that is easy to read by humans. One of the negatives of the attribution methods is the fact that it is difficult for them to deal with interactions between features.

The main focus of this thesis is to create a new perturbation-based attribution method that takes into consideration the inherent interactions between different features in data. While it can be used for any kind of data, we are focusing on explaining the decisions on textual data with word embedding representations.

In chapter 2, a simple description of neural networks theory is presented, which shows the complexity present in the neural network models. Chapter 3 deals with the problem of interpretability, why it is important and what approaches are used when it comes to neural networks. In chapter 4, we discuss the findings from the performed analysis on problem domain. In chapter 5, we present the hypotheses we have set. The design of our method, and how we want to evaluate it is in chapter 6 and the implementation details of our proposed method are in chapter 7. Chapter 8 presents the performed experiments. More specific implementation details and requirements of our method, and the experiments performed, are included in appendix A.

Chapter 2

Artificial Neural Networks

Artificial neural networks were inspired by the biological neural systems, with individual neurons and connections between them. But since their creation, they have diverged from a simple representation of human brain and has become one of the most popular model for machine learning tasks, because of their ability to perform really well. Neural network models belong to the family of gradient optimized models, such as linear regression models or logistic regression models [17, 4].

The first ever neural network model was the McCulloch-Pitts model. It was an extremely simple artificial neuron model, where the inputs and outputs could either be zero or one, while each input could be inhibitory (negative) or excitatory (positive). A sum of inputs was computed and if it was larger than a given threshold, the output was one, otherwise it was zero [17, 4].

This model was later extended to create the Rosenblatts perceptron, which is the basis of todays models. The perceptron added a few things to the original model: each input connection has a weight associated with it and so it is not limited to be of value zero or one. A bias term is also present as an additional input. Same as the original model, the perceptron performs a sum of inputs, however, in this case they are multiplied by their connection weights. After the sum is computed, it is run through an activation function which produces an output [17, 4].

A single perceptron can be used as a linear classifier, similar to the logistic regression. It can be interpreted as a function in the form: $y = f(w_0x_0 + w_1x_1 + ... + w_nx_n)$, where the single perceptron models a straight line separating the classes of the output, with the bias term serving as a shift of the line and the inputs serving as a rotation of the line. The f function can be any differentiable function as the training is done by updating the weights according to the gradient of the output. The most commonly used activation functions are sigmoid, tanh and rectified linear unit and maxout [4].

However, there was a critique for the single perceptron models, as they could not model

every possible function. A single perceptron model can only be used for linearly separable problems, which causes difficulties, as not many problems of that variety exists. Therefore, instead of just simply using one perceptron model, multiple perceptrons, marked as neurons, are stacked together in one layer. Additionally, multiple layers are connected together in a graph, often acyclic one but also cyclic in special architectures like reccurrent networks, producing multilayer perceptron models. These are the models we use today. The first layer is called the input layer, the last layer is called the output layer and the layers between are called hidden layers. It was proven that any neural network with one or more hidden layers is an universal approximator and therefore, given a continuous function $f(x)$, there exists a neural network that can approximate it. However, deeper networks with more hidden layers perform much better than networks with single hidden layers even though they have the same representational power. Nowadays, the deep neural network can learn representation of highly non-linear functions. This is allowed by the stacking of layers and the abstract representation of learned information saved as network parameters [17, 4].

2.1 Simple Feedforward Neural Networks

Feedforward neural networks, also called multi-layer perceptrons, are the most important and typical deep learning model. They form a basis of many important machine learning applications. The goal of the network is to approximate a function f by defining a mapping $y = f(\theta, x)$, where the y is the output and θ are the parameters, in this case weights, in the feedforward neural network that are learned [17].

Feedforward neural networks are typically represented by composing together many different functions, forming a directed acyclic graph that describes how these functions are composed together. The feedforward aspect of the network is caused by the flow of information through the network in only one direction, from the input layer, through hidden layers and finally to the output layer. There are no feedback loops in the architecture of the feedforward neural network [17].

There can be multiple layer types in feedforward neural network, however, almost always a fully-connected layer is used. In this layer, each neuron is connected to all of the neurons from the previous layer, but neurons on the same layer have no connections between them. By increasing the number of layers, the capacity of the neural network improves, which increases the space of representable functions. For this reason, high number of layers can lead to overfitting on the given data and therefore need to be addressed with stronger regularization. This overfitting can also be reduced using shallower networks, but it is not preferable as they cannot perform as good as deep networks [4].

Simple feedforward neural networks are similar to linear models. They make use of a

loss function, or cost function, to determine the error rate of the network and better train the model. The typically used loss function is the mean squared error for the regression tasks, and the most likelihood cost function for classifications tasks. However, in recent years a cross-entropy loss function became most popular for both tasks [4, 17].

The training of the feedforward neural network is performed by changing the weights of the connections between neurons. For this, a gradient based learning is used, which is similar to the learning in the linear regression. The biggest difference is that the nonlinearity in neural networks causes most of the loss functions to become nonconvex. Therefore, they are trained using an iterative gradient-based optimizers that just drive the cost function to its minimum instead of using linear equations or convex optimization algorithms. This learning has no guarantee of reaching global minimum and is highly sensitive to the values of initial parameters and so it is important for feedforward neural networks to have their weights initialized at small random values [17].

When using the feedforward neural network, the initial information from input x flows forward through the network and is propagated to the hidden units and finally to the output unit producing an output $\hat{y}$. This is called forward pass. On the final layer, a scalar cost is produced. However, to be able to train the neural network, the produced scalar cost must be propagated backwards through the network [17]. This is allowed by using the back-propagation algorithm proposed in [36]. This back-propagation algorithm computes the gradient of our cost function, first on the output unit, with regard to individual inputs from other neurons and adjusts its connection weights accordingly. This is done layer by layer and neuron by neuron from the last output layer, through hidden layers and finally to the input layer. For the computation, this algorithm uses the chain rule of calculus to compute derivatives of functions formed by composing other functions, which is applied recursively [17].

There are many higher order optimization methods that can be used in neural networks, such as Adagrad, RMSprop or Adam optimization or using a Nesterov momentum, which improve the learning process of the neural network, but also increase the complexity of computation [4].

2.2 Convolutional Neural Networks

Convolutional neural networks are an extension of simple feedforward artificial neural networks. They are still made of neurons with trainable parameters. Each neuron still gets an input, performs a dot product and outputs a value. What is different is the architecture of the network [3].

However, they are mostly used for tasks involving input with a much more complex structure, such as image recognition or classification, as they are able to achieve much higher

accuracy on these tasks. This is due to the fact that they explicitly assume that the input is an image and so their architecture is specialized for it [3].

This much better performance is evident from the results in the popular ImageNet Large Scale Visual Recognition Competition (ILSVRC), where convolutional neural networks outperform their multi-layer perceptron counterparts by a large margin [25]. The first convolutional neural network, which really outperformed non-convolutional approaches is the AlexNet [25], which achieved an error rate of 16% in the 2012 ImageNet challenge, while the network in second place achieved 26% error rate [49]. From that moment, the competition was dominated by the convolutional neural networks like the ZF Net [49], which improved upon the AlexNet by identifying its problems using approaches for interpreting the convolutional neural networks. Further improvements were introduced by the GoogLeNet [45] or VGG Net [43], where these networks introduced novel parts to the architecture.

One of the reasons why convolutional neural networks are better for tasks involving image recognition or classification is the fact that typical images can be viewed as high dimensional data. Every pixel can be considered as another input attribute and so a fully-connected layer has a large number of parameters, weights. This large number of parameter greatly increase the demand on the size of training data set and memory required for the simple non-convolutional models [27].

Convolutional neural networks do not need another manual or automatic feature extractor for them to be used effectively. In image recognition tasks, a feature extraction step is important for the maximalisation of the output precision. When it comes to simple feedforward neural networks, a manual or automatic feature extraction step must be first performed and its output used as an input for the classificator. However, convolutional neural network performs the feature extraction step using their hidden layers and so raw images can be used as an input, without any further increase in their parameter count or the required size of training dataset [27].

Last but not least, simple feedforward neural networks ignore the topology of the input data. The individual attributes in the input data can be presented in any fixed order, without it affecting the outcome of training. However, images have a strong two dimensional local structure, where individual attributes, or pixels, are highly correlated with their neighbourhood. These high local correlations are exploited in the convolutional neural network architecture. The receptive field of hidden units is constricted to be local and so local features, like corners or edges are extracted, before recognizing higher order features [27].

All in all, the main difference between convolutional neural network and simple multi-layer feedforward neural networks is in its architecture, where neurons in hidden layers are connected only to a small region of previous layer. In addition, neurons in layers are arranged in 3 dimensions, width, height and depth, while each layer transforms a 3D input volume to a

3D output volume of neuron activations. They introduce a further abstraction and complexity of representation of learned knowledge. Thus, the convolutional neural network can learn a much more complex representation of data than the simple feedforward neural networks [3]. There are 3 main layer types used in convolutional neural networks:

- Convolutional layer
- Pooling layer
- Fully connected layer

The architecture of the first convolutional neural network, LeNet-5, is depicted in figure 2.1.

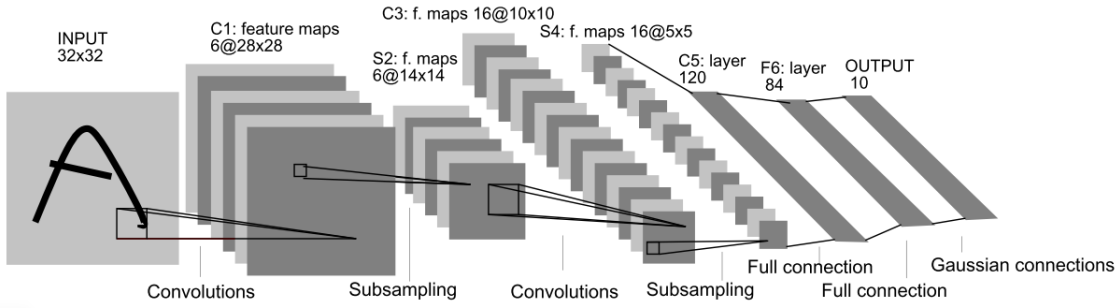


Figure 2.1: *Architecture of LeNet-5 convolutional neural network, consisting of a set of convolutional, pooling and fully-connected layers. Each plane represents one feature map [27].*

Convolutional Layer

Convolutional layer is the main building block of the convolutional neural networks and performs most of the computational heavy lifting [3]. Each convolutional layer in convolutional neural network consists of a set of learnable filters, which are small spatially, but extend through the full depth of the input volume. These filters can be interpreted as small windows, that are slided across the whole width and height of the input during forward pass. This sliding produces a 2-dimensional activation map, called feature map that gives the response of the specific filter at every position. The output volume is then produced by stacking of the feature maps along the depth dimension, as each is of the same width and height. The width and height of each feature map, and therefore also of the output volume, is determined by the size of stride for the sliding of the filters, the size of filter and the use of zero-padding. The number of filters is the determining factor of the depth of the output volume [3, 27].

The local connectivity of neurons in the convolutional layer is achieved by limiting the width and height of the filters to a number smaller than the width and height of the input

volume. The size of filters for each layer can be controlled by a hyperparameter called receptive field of neuron [3, 27].

To reduce the number of required parameters, weights and biases, a parameter sharing is used. It is based on the fact that, if a specific filter is relevant for one part of the input, it should also be relevant for the other parts as well due to the translationally-invariant structure of images. Therefore, each filter only has one set of parameters, with their number determined by the width and height of the filter. This eliminates the need to relearn the detection of important features at every location. Thus, the convolutional neural networks are invariant to rotation and translation in images [3, 27].

The convolutional layer is typically followed by an element-wise activation function, similarly to the simple feedforward neural networks. Typically used activation functions are rectified linear unit or sigmoid [3].

Pooling Layer

The pooling layer is in charge of down-sampling the spatial dimension of the input. By reducing the spatial size of the representation, the amount of parameters and computations is also reduced. In addition, this action also controls the over-fitting of the network by reducing the resolution of the feature map and therefore reducing the precision with which the position of the distinctive feature is encoded in the feature map [3, 27].

Pooling layer works similarly to the convolutional layer. A small window of a fixed size is defined. It is used to stride through the input volume, but instead of computing a dot product, a simple function is used. The function used determines the name of the pooling layer. Typically a max-pooling layer is used, which simply takes the maximum of its inputs. Additionally an average pooling, which takes the average of its input, or an L2-norm pooling, which calculates euclidean distance between inputs, can be used. The output of the function is then multiplied by a trainable coefficient and optionally can be passed to an activation function [3, 27].

Fully-connected Layer

Fully-connected layer is the equivalent of the regular layers present in the simple feedforward neural networks. Each neuron in fully-connected layer is connected to every neuron on the previous layers. Therefore, they are usually only used as a last layer of the convolutional neural network to output the class probabilities [3, 27].

The only difference between fully-connected layer and convolutional layer is that neurons in convolutional layer are connected only to a local region of the input and share weights. Other than that they are the same. Therefore, it is becoming very popular to convert fully-

connected layers to convolutional layers by defining a filter size equal to the whole image, with no zero padding and stride value of 1. By performing this conversion, it is possible to use the convolutional neural network on images of bigger size than the size they were trained on, by sliding the original neural network across different spatial positions in larger image in one forward pass and then averaging the output class probabilities [3].

Chapter 3

Interpretability of Neural Networks

Interpretability matters. In order to build trust in intelligent systems and move towards their meaningful integration into our everyday lives, we must build transparent models that are able to explain why they predict what they predict, in addition to doing the prediction correctly. Therefore, the accuracy of the model is not the most important factor of it, but also its interpretability is important. This is most apparent in the case of using machine learning models in the environment, where small mistakes can lead to catastrophic consequences, for example medicine. If machine learning model can not explain why it made the specific prediction, it cannot be trusted and therefore it will not be used [12].

Additional need for interpretability originates not only from the need to understand and therefore develop better models, but also from the law perspective. Recently, a law[1] was adopted by the European Union that, among other things, introduces the *right to explanation*. What this means is that every person has the right to demand the logical reasoning behind the decision that directly or indirectly affected them. This is however problematic, if all the machine learning models used are black boxes that cannot explain their decisions [18].

There is however a trade-off for interpretability of models. If we want to make a machine learning model more interpretable, it often suffers in its accuracy and other way around. This is apparent in the recently developed deep models that achieve great performance through greater abstraction and tighter integration at the cost of interpretability. For example recent deep residual networks (ResNets) are over 200 layers deep, which makes them especially hard to interpret [41].

There are different machine learning models with different levels of the interpretability-accuracy trade-off. Some machine learning models might not need explanations, because they are used in a low risk environment, meaning a mistake has no severe consequences. Or the method has already been extensively studied and evaluated or is easily interpretable as

[1] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), OJ L 119, 4.5.2016, p. 1–88

is, for example decision trees or linear models. On the other hand, there are models that are considered as black boxes that are really hard to interpret and therefore they are not trusted and are not used, even though their performance is much better [12, 18].

The neural networks are one such model. They perform a highly non-linear transformations in data by using multiple hidden layers with many neurons, which causes them to be highly abstract. In addition, the learned interactions in the data are saved in the form of weights between individual neurons across multitude of layers. For these reasons, they are able to achieve high accuracy for many machine learning tasks, simply by adding layers. This, however, also causes them to be viewed as black boxes that are hard to explain. In addition, it has been shown that deep neural networks can be easily fooled into misclassifying inputs with no resemblance to the specific class by making imperceptible alterations to its pixels [30].

Therefore, one of the main focus of the research in the field of deep learning and neural networks is the creation of new methods for increasing the interpretability of the deep models, while maintaining their high accuracy [41]. The deep neural networks can be looked at from different perspectives. We can try to explain the processing done by the network, which answers the question of why does a particular input lead to specific a output. Another approach is to look at the representations learned by the neural network, which answers the question of what does the network contain. The third approach is to create an explanation producing system with architecture designed to interpret its own behaviour, either by explaining the processing or learned representation. We mostly focus on explaining the processing of neural network, which can be divided into 4 categories of approaches, that are often related [15]:

- **Model-agnostic** aproaches used for simplifying the model and generating explanations from this proxy model.
- **Automatic-rule extraction** approaches used for generating simple rules for the decision process.
- **Visualisation** approaches that generate visualisation of either the processing or representation learned by the network.
- **Attribution** approaches that generate attributions of input for the given output.

One last problem that not many researchers deal with when it comes to interpretability is defining what is meant by the words *interpretation* and *explanation*. Many researchers are using the two words interchangeably, even though they definee different things according to [15]. The goal of *interpretability* is describing the internal workings of the model in human understandable format. To get to the *explainability*, the approach must satisfy the goal of completeness. The goal of completeness is to describe the workings of the system in an accurate way. Nonetheless, achieving both interpretability and completeness simultaneously

is hard. There exists a trade-off between these two, as interpretable explanations often lack in predictive power. This can be evident in the article by Ribeiro *et al.* [35], where by *explaining the prediction* they mean to present a visual or textual artefacts that provide qualitative representation between components and models prediction. They find it important to generate explanations that can easily be interpreted by humans, even though they are not necessarily complete, by limiting the number of generated 'artefacts' to only the most important ones. In this work, we take similar approach and therefore use the *explainability* and *interpretability* interchangeably, which means to **generate a visual representation of relevance of models components to its output that is understandable for humans**.

3.1 Model-Agnostic Generation of Proxy Models

Model-agnostic approaches hold a special place in the interpretability of the neural network models. The main idea behind them is to treat the examined model as a black box that transforms an input to output. Using this transformation, an approximation of the complex model is created by training a surrogate model, which is much simpler. This surrogate model is usually one of the more interpretable types of models, like simple linear model or a decision tree. This approximated simple model is then assumed to be indicative of the internal workings of the complex model and therefore the explanations produced by this proxy model are representative of the explanations of the complex one.

3.1.1 Local Interpretable Model-Agnostic Explanations

Local Interpretable Model-agnostic Explanations (LIME) approach, introduced by Ribeiro *et al.* [35], is one of the most notable representatives of this approach. This approach is used to produce explanations for individual predictions made by the model, in order to increase the trust of humans in a specific model. The inspection of individual predictions and their explanations, in addition to accuracy metrics, is considered a worthwhile solution of increasing trust and simplifying the task of deciding which model to use in the production. The main idea is to provide a set of weighted features that are the most influential on the output. This process can be illustrated as in figure 3.1.

The LIME algorithm can explain predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model [35]. As we can see from the citation of the creators of the LIME approach, there are two main points of focus for his approach: generating a *faithful* explanation using a *local approximation* of the model.

In order to generate the faithful explanations, a number of desired characteristics was set prior to the development of the approach. First criterion is that the generated explanations

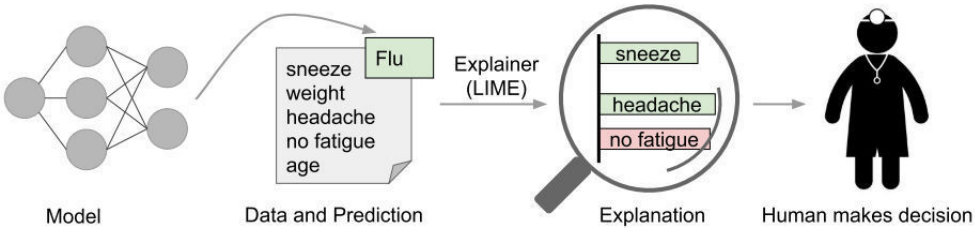


Figure 3.1: *LIME as used for explaining individual predictions. Model is predicting that a patient has a flu. The LIME approach highlights the features that led to the prediction, where 'sneeze' and 'headache' is promoting the decision and 'no fatigue' is prohibiting the decision. Using the explanation, a doctor can determine if the model can be trusted or not [35].*

must be interpretable. What it means is they should provide qualitative understanding between input variables and the response. In addition, the explanation must take users limitation into account and therefore it should be easy to understand by humans. This can be satisfied by presenting only small number of weighted features [35].

Another criterion is the local fidelity of explanations. What it means is that the explanation should be indicative of the complex model, at least in the local vicinity of the observation. Last but not least, the explanation approach should be model-agnostic, as this provides also the flexibility of explaining future classifiers.

The local approximation part of the studied complex model is performed by training a surrogate model. For the specific input, its neighbourhood is perturbed, generating new observations. This perturbation is done uniformly around the observation and each new data is weighted by a distance to the observation metric. Each new observation is passed to the black box classifier to determine its output. This generates data which describe the behaviour of the complex model in the surroundings of the examined decision. The new observations are then used to construct a much simpler surrogate model on the neighbourhood of the decision and used for generating the explanation for complex model. This simple model can either be a linear classifier, a decision tree or anything else that can provide explanations easily. It was shown that the method can be used to identify regions that are most influential for the specific decision. One such process can be seen in figure 3.2, which represent the explanation of decisions on the Googles Inception neural network [35].

In addition, a method for global interpretation is proposed, which uses the local interpretations. Providing a global perspective on the model is important to improve the trust in the model. For this purpose, a set of local interpretations are used as a mean to create a global representation. These local interpretations are not picked at random. The observations that are picked must be selected to cover the most important components, but still not be redundant between each other. To achieve this, a maximization of a weighted coverage function is performed using a greedy algorithm.

Figure 3.2: *Explanation of image classification of 3 top classes, on the Google Inception neural network produced by LIME approach. From the left the images represent: an original image; explanation for the class 'Electric Guitar'; class 'Acoustic Guitar'; the class 'Labrador' [35].*

3.2 Extracting Rules from Trained Neural Networks

Methods of rule extraction from neural networks deal with producing a description of the neural network hypothesis that is comprehensible and yet closely approximates the networks predictive behaviour given a trained neural network and the data it was trained on [11]. The extracted rules can be in different forms: simple **IF-THEN** rules with multiple conditions that need to be satisfied and one implication. **M-of-N rules**, where if M instances of set of N instances are satisfied then the outcome is usable. Or in the form of **decision trees** typically used in machine learning [19].

There are three different sets of approaches that can be used for rule extraction: *decompositional approaches*, *pedagogical approaches* and *eclectic approaches* [19, 7].

Decompositional approaches work by splitting the network into individual neurons and extracting rules from them. These rules are then aggregated across layers and used for the whole network [19, 7]. One such approach is called KT proposed in [13] and it is used for extracting IF-THEN rules. For each hidden and output unit, a set of positive attributes and negative attributes is generated. The set of positive attributes is first searched for the combination of those, whose summed weights exceed the threshold of the neural network. Afterwards, the set of negative attributes is searched in similar way. The attributes from both searches are then aggregated into a one IF-THEN rule and used for interpretation.

Another such approach is the DeepRED [52], which was specifically designed to be used on deep neural networks. This method decomposes the neural network into a decision tree. It is an extension of a previously specified algorithm that could have only been used on shallow networks to work on any number of hidden layers. Multiple strategies are used to generate this decision tree, a C4.5 algorithm for training the tree and a pruning algorithm to remove unnecessary inputs. The decision tree generated this way very closely approximates the neural network, which causes it to be quite large and therefore not really interpretable.

Pedagogical approaches treat the neural network as a black-box, as their main focus is

finding an output rule for a corresponding input. The internal structure of neural network is not taken into consideration [19, 7]. One such approach is called TREPAN [10]. This approach extracts a M-of-N split points and decision tree by using a query and sampling approach. What it basically does is training a decision tree on the input-output mapping of the neural network [10].

Eclectic approaches combine the previous two approaches [7, 19]. One of the representatives is called Rex-CGA [22] and is used for multilayer perceptrons. It uses a clustering genetic algorithm to generate a set of clusters of hidden units activation values. After finding these clusters, it treats them as individual neural networks and use an input-output mapping rule generation from pedagogical approaches to generate rules for then. They are then aggregated across the networks and used for the whole neural network.

Each set of approaches has its own positives and negatives. The decompositional approaches are much more transparent but because they work layer by layer, they are highly dependent on the architecture of neural network and can be rather slow and memory consuming and therefore not scalable. The pedagogical approaches on the other hand are much more flexible, as they are architecture independent, however they usually perform worse. The eclectic approaches combine the decompositional and pedagogical ones, so they take the positives and negatives from both of them, and therefore are slower than pedagogical but faster than decompositional and more accurate than pedagogical but less accurate than decompositional. Another negative for most of the rule extraction approaches is the fact that they perform rather poorly on deep neural networks as they were created in time, when the deep learning was not so widespread [7, 19, 52].

3.3 Visualising What Neural Network has Learned

Methods for visualising the neural network models are numerous and are based on the fact that, for most people, visual representations of structures and relationships are much easier to understand than rows of numbers and data. In machine learning, it is popular to perform visual analysis of data before training a model, which leads to getting the idea about what relationships are in the data. Then, after visualising what model has learned, if there are these relationships also present in the trained model, it enhances our trust in the model, as we see it has learned something that was apparent to us in the data [51].

Most of the methods for visual interpretation of neural networks are aimed at convolutional neural networks and images, as they are far more simple to design to provide the necessary explanations we need. These methods, most of the time, display the input image, but with modifications that highlight the parts of the image that are used for the classification of the image [50, 41, 49]. We can say that these visualisation techniques are used to provide a

visual explanation for the decisions of the trained model. However, to provide a good visual explanation, the visualisation approach should be class-discriminative, therefore localize the target category in the image, and high-resolution, therefore capture a fine-grained detail [41].

However, there are also methods that visualise the structure of the neural networks, the individual hidden layers, neurons on them and connection weights between them [1]. The problems with these methods are that they are not really usable for neural network models that contain high number of hidden layers and neurons and therefore they are not often used.

The last category of methods for visualising neural networks are the methods that visualise the activations of neurons on individual layers of the network, usually the last layer, along with the training process of the network [34]. These methods can be used for images, but also for tabular data, however they are a bit harder to interpret than the methods used purely for convolutional neural networks and image input data.

3.3.1 Visualising Interesting Parts of Images Used for Prediction

One example of the methods that visualise interesting parts of images is the method proposed in [50]. This visualisation method exploits the remarkable localization abilities of convolutional neural network that is apparent even despite being trained on image level labels. The main focus is on generating a class activation map using a global average pooling, that was introduced in recent year to the convolutional neural networks to remove most of the fully-connected layers but still retain performance.

For this method to work, a change in architecture must be introduced to most of the convolutional neural networks. This change includes the addition of the global average pooling layer before the output layer, which is essential for this method. Afterwards, a class activation map can be computed using following equation:

$$M_c(x, y) = \sum_k w_k^c f_k(x, y) \tag{3.1}$$

where $M_c(x, y)$ represents the activation of class c for pixel located on position x and y respectively in the input image, the w_k^c represents the corresponding connection weight to class c for neuron k and the $f_k(x, y)$ represents the activation of neuron k in the last convolutional layer at spatial position (x, y). Basically, the class activation map represents the importance of image regions for the classification and it is computed by projecting back the weights of the output layer on to the convolutional feature map [50]. The generated class activation map is illustrated in figure 3.3.

The downside of this method is the fact that the architecture of the neural network model must be changed by removing most of the fully-connected layers and adding a global pooling layer just before the output layer. This, however, is often not possible and therefore a much

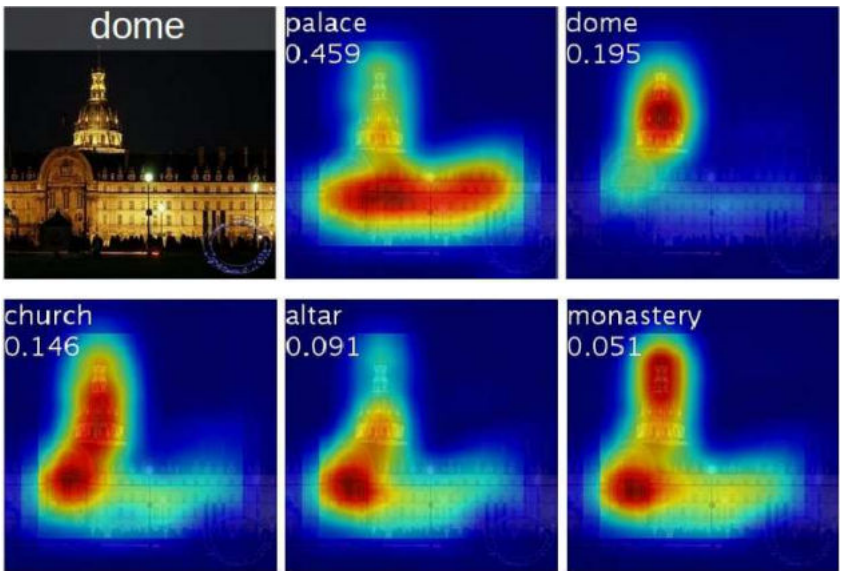


Figure 3.3: *Class activation map of top 5 predicted labels for the given image with a dome present in it. The predicted class along with its probability is shown above each class activation map. The highlighted parts show which part of image activate which class, with the red color specifying the highest activation and blue color the lowest [50].*

more general method was introduced in [41], which can be used on a much wider variety of convolutional neural networks, for example those with fully-connected layers, those that are used for captioning, or for those with multi-modal inputs or reinforcement learning, without the need for the architectural change or re-training.

This method is a direct extension of the class activation method and is based on computing a gradient and therefore called gradient weighted class activation map. However, instead of using gradients with respect to the output of the whole network, which is usually produced by fully-connected layer, this method uses the output of the last convolutional layer. The reason is that the convolutional layers still retain a spatial information, which is lost in the fully-connected layers, and so it is the best compromise between high-level semantics and detailed spatial information. It uses a gradient information flowing into this last convolutional layer to understand the importance of each neuron for a decision of interest [41].

There are two steps that are used for computing the gradient weighted class activation map that use the following two equations:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\delta y^c}{\delta A_{ij}^k} \tag{3.2}$$

$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k) \tag{3.3}$$

First a gradient of the probability for class c, y^c, is computed with the respect to the feature map A^k of a convolutional layer. These gradients are then global average pooled to obtain the

neuron importance weights α_k^c that captures an importance of feature map k for class c. A weighted combination of forward activation maps is performed and is followed by a ReLU function to produce the importance map. ReLU is applied because we are only interested in positive impact on the class of interest and not the negative one. For this method to work, the y^c only needs to be a differentiable activation function [41]. The produced activation maps are shown in figure 3.4.

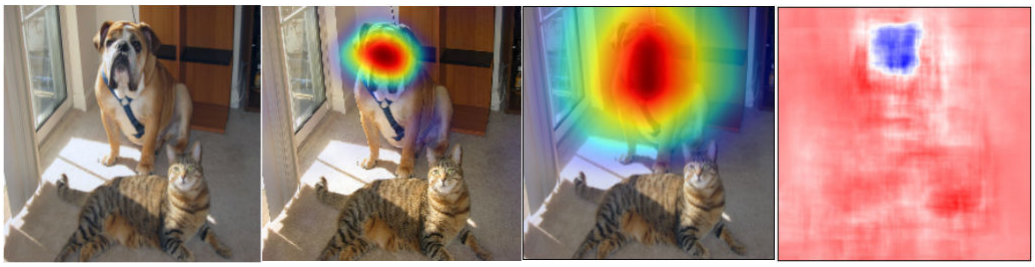


Figure 3.4: *Output of the gradient weighted class activation map (Grad-CAM) approach. The important parts of the image for classfication are highlighted. From the left: original image; class discriminative regions for class 'dog' in simple convolutional neural network; class discriminative regions for class 'dog' in ResNet-18 layer; occlusion map for class 'dog' generated by the Grad-CAM method [41].*

3.3.2 Visualising the Activations of Neurons and Training Process

One method that deals with visualising the neural network neuron activations and visualising the process of training the neural network is presented in [34]. It was designed to be used with convolutional neural networks, but also with multi-layer perceptrons and was used for image datasets, but it can be used also for other types.

It is based on the extraction of the hidden layer activations from the network for a given dataset. This method can be divided into two parts: creating projections from the extracted activations from the network and depicting the relationships between neurons from which these activations were extracted. This method goes as follows [34]:

1. A subset of observations is selected from the test set and fed to the neural network.

2. For each observation, the activation of neuron on the observed layer is extracted and added to the observation.

3. The observations along with their activations are projected to the two-dimensional space using a fast approximate implementation of t-distributed stochastic neighbour embedding (t-SNE) using default parameters.

4. The projections are then visualised as scatter plots with points colored according to their class assignment.

This helps understand the relationships between learned representations of different observations. However, it does not show relationship between neurons or their interaction that leads to fulfillment of the discriminative task. Therefore it is extended by neuron projections. In neuron projections, the neurons are also visualised as scatter plots based on similarity between them, which is computed as Pearsons correlation coefficient. The color of the neuron in the scatter plot is determined by the power of the neuron for discriminating specific class in comparison to the rest. Similarly to observations, neurons also have to be projected to two-dimensional space, in this case, using multidimensional scaling (MDS) [34]. These visualisation are shown in figure 3.5.

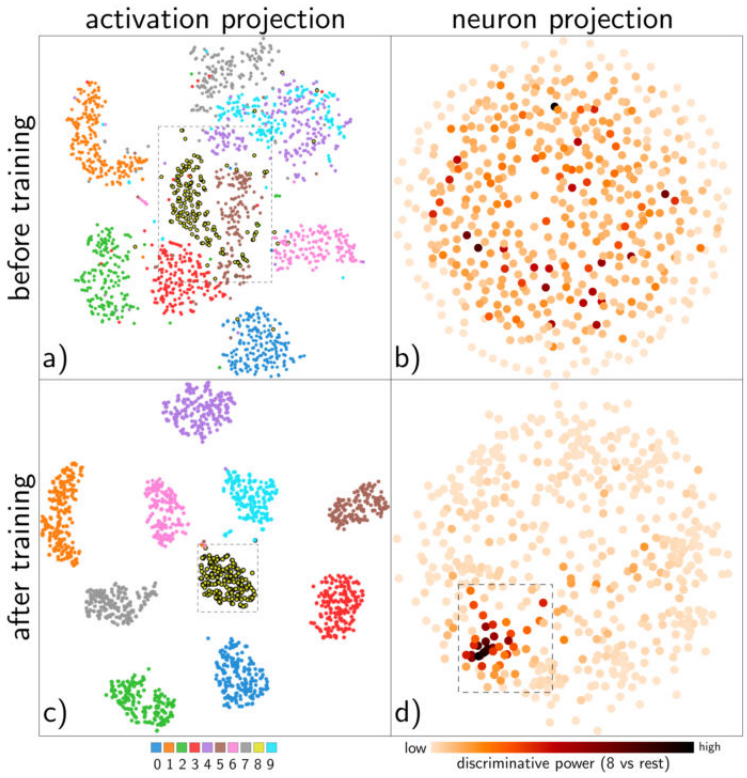


Figure 3.5: *Visualisation of activations and neuron projections of the last layer of convolutional neural network trained on MNIST dataset. The plots a) and b) represent the visualisation before training, while the plots c) and d) are after training. The plots a) and c) show the observations and their activations colored according to their class. The plots b) and d) represent the neuron activations, with the position determined by the similarity between neurons and color determining the discriminative power for specific class in comparison to the rest, in this case the discriminative power is for class 8 [34].*

In addition to visualising the activations, this approach can be used to visualise the training process of the neural network. It provides a way to visualise the inter-layer evolution, e.g. activations between layers, of the activations of the observations, but also the inter-epoch evolution, activations between individual epoch of training. The visualisation is color-coded similarly to activation projections, where individual colors show individual classes. In addition,

an arrow is also shown that goes from pure black to white, portraying the change through training. This approach can be viewed as performing the previous activations multiple times during training and aggregating them into a single diagram [34]. This technique is illustrated in figure 3.6.

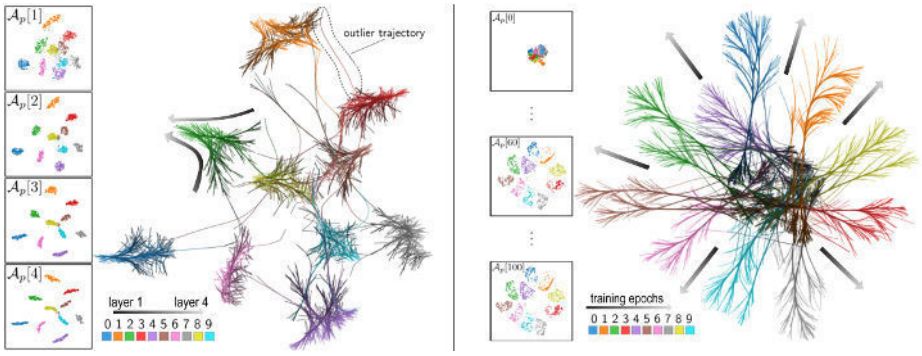


Figure 3.6: *Visualisation of the training process of the neural network. The plot on the left shows the evolution of activations between individual layers, with colors representing individual classes and the brightness represents the layer number, with brighter trails indicating later layers. The plot on the right shows the evolution of activations between individual epochs of the last layer, with colors representing individual classes and the brightness representing the number of epoch, with brighter trail indicating later epoch [34].*

3.4 Generating Input Attribution for the Decision

Attribution methods deal with assigning an attribution value to each feature in a network for a single observation. The attribution value represents a relevance or a contribution of the feature. Consider a neural network with inputs defined as $x = [x_1, x_2, ..., x_N]$, with N representing the number of features. This neural network is used to produce an output vector in form $S(x) = [S_1(x), S_2(x), ..., S_C(x)]$, where C represents the number of output classes. The goal of attribution method is to determine the contribution of each feature in the input vector x to the output S_c, when given an output neuron c [2]. For example, when it comes to classifying an image, the output neuron is the neuron representing correct class for classification and the contributions are calculated for each pixel in the input image.

One of the main positives of using the attribution methods is the simplicity of visual representation of the generated attributions. When examining an individual observation, the attributions for each features can be arranged together to create an attribution map. This attribution map can be easily displayed using a heatmap, with red color indicating the promotion of the class and blue color indicating a suppression of the correct class [2].

The attribution methods have their roots in the sensitivity analysis approach, as it was the name of the first approach that was developed. The sensitivity analysis approach involves a

series of methods quantifying how the uncertainty in the output is related to the uncertainty in its input. In other words, it assesses how sensitive is the output of the model to the fluctuations in its input and parameters that it was built with [48, 38]. This, in fact, is the same thing as generating attributions, as it determines which input features are the most contributing for achieving high accuracy in output.

The sensitivity analysis approach was first designed to be used for statistical regression models. The regression models, or linear models can be expressed as

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon \tag{3.4}$$

with $\mathbf{Y}$ representing vector of dependent variables, $\mathbf{X}$ representing matrix of independent variables, β representing vector of coefficients and ϵ representing vector of errors. This was one of the requirements for it, as the way it computed the importances was by using gradients.

Nonetheless, this requirement is fulfilled by the neural networks too, as the individual neurons can be expressed as

$$\mathbf{Y} = f(\mathbf{X}, \mathbf{w}) \tag{3.5}$$

with $\mathbf{Y}$ and $\mathbf{X}$ representing same variables as in regression model and $\mathbf{w}$ representing the matrix of weights of the network, which is just the extension of vector of coefficients from regression models. The only requirement is for the function f to be differentiable, which is also one of the requirements in most neural networks and therefore it is fulfilled as was determined in chapter 2. Therefore, sensitivity analysis can still be applied to neural network models [47].

While the requirement of the specific form of the model representation is fulfilled by the neural network, it is not really neccessary. This requirement is only necessary for the methods stemming from the sensitivity analysis. There are, however, additional advances in attribution approaches, that do not require it.

The specific way the attribution is computed in neural network model is determined by the type of method used. There are three main categories of methods that can be used:

- **Weight magnitude analysis** methods using weights to determine the importance of features.
- **Gradient** methods that use the gradient to backpropagate the importance.
- **Input perturbation** methods that perturb the input to determine the importance of features.

We can say that using this specification of attribution methods, also the LIME method described in previous sections (3.1.1) can be considered a model-agnostic attribution method.

Additionally, the methods visualising the interesting parts of the image can be considered a type of attribution method as well.

By determining the feature importances using the attribution method, we can verify that the model does what it is intended to do and therefore determine its stability and applicability in practice [23, 48, 38, 2].

3.4.1 Weight Magnitude Analysis Methods

These methods are defined by observing the connection weights between the input and hidden nodes. The idea behind this is that the variables with higher influence on the output node result should also have larger connecting weights between the input and the hidden nodes [33].

One of the most common approaches is to sum all the weight magnitudes. For each input node, the sum of its output weight magnitudes from each of the hidden layer nodes is the relative influence of that hidden node on the output. To determine relative importance for each variable, it is done for all input nodes, where one input node represent one variable. However, a normalization of the weights must first be done. It is done by dividing weight magnitude of each of the input nodes by the largest connecting weight magnitude between the input and the hidden layer. After normalization, the weight magnitudes from each input node to the nodes in the hidden layer are subsequently summed and ranked in a descending order. The rank indicates the relative influence of the input variable on the output and its formula can be denoted as:

$$I_i = \sum_k \frac{w_{ik}^0}{max_{i,k}(w_{ik}^0)} \tag{3.6}$$

where w_{ik}^0 denotes the connection weights between the *i*th node on the input layer and *k*th node on the second layer and I_i denotes the importance of the *i*th input variable [33].

Another approach, called neural interpretation diagram, is used to calculate relative magnitude of each connection weight and then using it for visualisation. It is a tool for providing a visual representation of the connection weights among neurons, where the calculated relative magnitude of connection weight magnitude is represented by the line thickness, where thicker line indicates larger weights. The connections are also shaded according to the sign of their magnitude, where black lines represent positive or excitation signal and grey line represent negative or inhibitory signal. The neural interpretation diagram is displayed in figure 3.7. By calculating the connection weight magnitude, visualizing it and tracking its direction between each neuron, this technique enables to identify individual and interacting effects of the input variables on the output. However, the interpretation of this neural interpretation diagram can become really challenging with increasing number of input variables and hidden layers. It is also quite difficult task even when it comes to simple neural

networks due to the complexity of connections among neurons as the magnitude and sign of weights can change from layer to layer [32, 31].

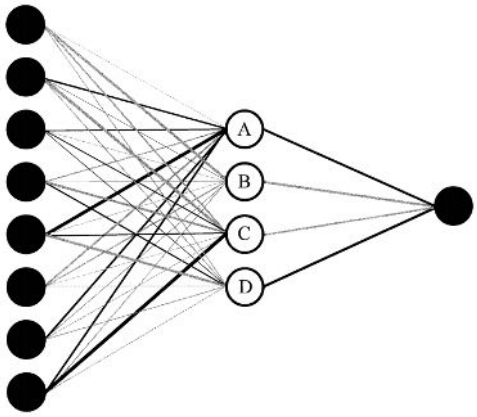


Figure 3.7: *Neural interpretation diagram for neural network with 8 input variables, one hidden layer with 4 neurons and one output neuron. The thickness of the lines determine the magnitude of the weight connection proportional to other connection weights. The colour determines the sign of connection, black is for positive (excitation) connection and grey is for negative (inhibition) connection [31].*

All the methods that were described so far, from the weight magnitude analysis methods, considered that connection weight between neuron from previous layer to the neuron on the current layer had the exact same effect on the size of connection weights between neurons on current layer and the next layer. However, in most cases it is not the case. The approach that takes this into consideration was first proposed by Garson [14] and later modified by Goh [16]. It is called the weight deconstruction or partitioning method, as it partitions the connection weights between each neuron in hidden layer and each neuron in output layer into components associated with each neuron in input layer. Consider a neural network with large connecting weight between one hidden layer neuron and the output neuron. If the neuron on the input layer has small connecting weight to that specific neuron on hidden layer, its importance will be determined to be much smaller than input neuron with large connecting weight, as its contribution to the output connection weight can be considered as much smaller. The computation consists of these steps:

1. For each input neuron i the absolute value of the connection weights between hidden and output layer is multiplied by the absolute value of the connection weight between input and hidden layer. This is done for each variable j and the products P_{ij} are produced.

2. For each hidden neuron, the product P_{ij} is divided by the sum for all input variables, getting the product Q_{ij}.

3. For each input neuron, its Q_{ij} products are summed to produce product S_j.

4. Each S_j product is divided by the sum of all input variables ($\sum_i S_i$). This expresses the relative importance or distribution of all output weights attributable to the given input variable.

However, it is important to note that this algorithm uses the absolute values of connecting weights when calculating the variable importances and therefore it ignores the sign of the relationship between input and output [31].

The last approach, proposed in [31], is to use randomisation test for the connection weight selection. It eliminates null-connection weights that do not differ too much from random weights, which significantly simplifies the interpretation of the neural network, for example when it comes to the neural interpretation diagram. This approach is very similar to other pruning approaches. This approach consists of these steps:

1. A number of neural networks with random weights are constructed using the original data.

2. The best network is selected and following specifications are recorded: its initial weights, the connection weights after training, their product, their overall value and the relative importance for variables.

3. Original response variable is randomly permuted producing y_{random}.

4. New network is constructed using y_{random} and random weights.

5. Steps (3) and (4) are repeated a number of times while recording weight specified by step (2).

Then, the significance of each connection weight and relative importance of attribute can be calculated as the proportion of randomised values equal or greater than the observed values.

The problem with these methods is the fact that they can not be really effectively used for neural networks that are more complex and contain large number of hidden layers. Another important thing is the fact that the input data must be first normalized for them to be used effectively as the large difference in the scale of variables can lead to weights having different magnitude, which does not represent the importance of the variable [40]. Another downside is the fact that to use these methods it is required to know the architecture of the neural network [47].

3.4.2 Gradient Methods

The distinction of these methods is that they are computing an approximation of the attribution in a single forward and backward pass through the network, one layer at a time, starting

from the output layer and proceeding backwards to the input layer. There is one fundamental requirement for these methods. The activation function must be differentiable, as the basis for them is found in using the gradient [21, 2].

Sensitivity Analysis

The sensitivity analysis represents the first and most simple gradient method. The feature importances are computed by applying a simple backward chaining partial differentiation rule. First the output relevances, with respect to variations in the values in neurons of layer N are calculated and then a backward chaining is employed to calculate the output relevance in the layer input variables. The equation for calculating the influence of each input variable on the output for a simple feedforward neural network with one hidden layer, one output unit and *j* input variables can be denoted as:

$$I_i = \sum_k O(1-O) w_{k1}^2 v_k^2 (1 - v_k^2) w_{ik}^1 \; ; \; \forall i = 1, 2, ..., j \tag{3.7}$$

Where O denotes the value on the input node, w_{k1}^2 denotes the outgoing weights of the *k*th node in the second hidden layer, v_k^2 denotes the output value of the *k*th node in the second hidden layer, w_{ik}^1 denotes the connection weights between the *i*th node on the first layer and *k*th node on the second layer and I_i denotes the importance of the *i*th input variable [21, 33].

The augmented equation method can also be found in interpretation of the deep neural network models. It is based on the models locally evaluated gradient or some other local measure of variation. A common formulation of it in deep neural networks is as follows:

$$R_i(x) = (\frac{\partial f}{\partial x_i})^2 \tag{3.8}$$

where the gradient is evaluated at the data point x_i. The most relevant input features are those to which the output is most sensitive. The technique is easy to implement for a deep neural network, since the gradient can be computed using backpropagation.

However, there is a problem that this method is strongly exposed to derivative noise that characterizes complex machine learning models and the accuracy of these methods is often questioned. This problem is evident in figure 3.8, where the heatmaps are discontinuous, scattered and do not focus on class-relevant features. Instead it indicates what pixels make the digit belong more or less to target class rather than what pixels are specific for the target class [29]. Additionaly, this method can compute only the positive relevance of the features. It is, however, also desirable to show the features that are prohibiting the decision [2].

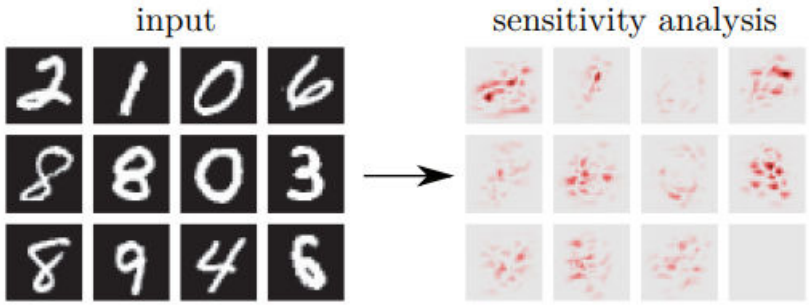


Figure 3.8: *Sensitivity analysis applied to a convolutional deep neural network trained on MNIST data set, and resulting explanations (heatmaps) for selected digits. Red color indicates positive relevance scores [29].*

Integrated Gradients

The integrated gradients method was built upon predefined axioms, or desirable characteristics, that allow for better empirical evaluation of attribution methods. There is a significant challenge in developing an attribution method, due to the fact that they are hard to evaluate empirically. It is hard to separate the errors of model from errors of the attribution method. For this reason, 2 axioms are set in this work and an attribution method is designed to satisfy both of them. No other gradient attribution method satisfies these axioms [44].

First defined axiom is called *sensitivity(a)*, which is usually broken when using simple gradients. The definition of *sensitivity* is that for every observation, if the specific observation differs in one feature from a baseline observation and the prediction for these two observation is different than the differing feature should have a non-zero attribution. It is common to inspect the products of model coefficients, weights, and feature values when debugging a model. Gradients are a natural analog of the model coefficient and therefore using a product of gradient and feature values is a reasonable starting point for any gradient based attribution method. The problem with gradients is that they break the first axiom, *sensitivity*. This is due to the fact that the prediction function may flatten at input and have zero gradient despite getting different input values. This causes simple gradients to focus on irrelevant features [44].

The second defined axiom is called *implementation invariance*. What it means is that if two different networks are functionally equivalent, e.g. their outputs are equal for all inputs, then the attributions should always be identical too, even when the networks are wildly different. The functional equivalence is often present in neural network architectures, where two different sets of parameter values lead to the same function learned. Therefore, the attribution method should not be dependent on a simple parameter initialization. It is proven that gradients have this property and therefore are implementation invariant [44].

Using these two axioms, a new attribution method is proposed, the integrated gradients. This method first defines a baseline input $x^{'}$, which can be a black image or a zero word. For a given input x, the method considers a straightline path from the baseline to this input and a gradient is computed at each point along the path. These gradients are then cumulated using an integral along the path. The integrated gradient is then defined along the i^{th} dimension for input x and its baseline $x^{'}$ as [44]:

$$(x_i - x_i^{'}) * \int_{\alpha=0}^{1} \frac{\partial F(x^{'} + \alpha * (x - x^{'}))}{\partial x_i} d\alpha \tag{3.9}$$

As computing the integral over the gradients can be problematic, an approximation is usually used instead. One efficient approximation is to use summation over gradients at sufficiently small intervals of the input perturbation [44]. Therefore, this method is a kind of hybrid between the perturbation and the gradient based approach, as an input is being perturbed, however a gradient is computed at each perturbation.

One important part of using this method is selecting a good baseline. It should have a near zero score in the classifier. In addition it should not be just simply an adversarial example, but a complete absence of signal in input [44].

DeepLIFT

The DeepLIFT approach explains the difference in output from a reference output in terms of difference of the input from the reference input. This reference input/output can be viewed as the baseline input from the integrated gradients method. It must also be chosen appropriately to the problem at hand. To obtain insightful results, both references must be chosen by relying on domain knowledge or trying the method using multiple references [42].

It uses a *summation-to-delta* property. Given a target output neuron of interest t, a set of inputs $x_1, x_2, ..., x_n$ that are neccessary but also sufficient to compute t, the *summation-to-delta* property can be expressed using an equation:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta t} = \Delta t \tag{3.10}$$

The Δt represents the difference from baseline for the output neuron. The $C_{\Delta x_i \Delta t}$ is the contribution score, which can be thought of as a difference from output that is blamed on the difference from reference of x_i. One of the positives of this contribution scores is that it can be non-zero even if the gradient of output in regards of the input is zero.

For the assignment of contribution scores to the input features, a set of rules is defined. These rules can be used to find the contribution of any input to a target output using a backpropagation. These rules are using the *summation-to-delta* property and a differentiation between

positive and negative contributions. The computation of contribution is also dependent on the activations used, for example when considering a softmax or sigmoid activation on the output, it is preferable to compute the contributions using the layer preceding this final non-linearity instead [42].

Layer-wise Relevance Propagation

The layer-wise relevance propagation is a technique for estimating which input features are important for achieving a certain decision. The main idea is in redistributing the output prediction score, for each possible class separately, back to the input space using a backward propagataion procedure. This backward propagation procedure should satisfy a layer-wise conservation principle. Using this principle, not only the last input features get the relevance scores from output, but also all the intermediary layer neurons are assigned a relevance scores. It also holds that the sum of all relevance scores on each layer should be equal to the output prediction score [8, 6]. Lets consider a neural network with one hidden layer. The input layer contains k features, denoted $x = [x_1, x_2, ..., x_k]$. The hidden layer contains n neurons, denoted $v = [v_1, v_2, ..., v_n]$. And the output layer consists of m output neurons, denoted $y = [y_1, y_2, ..., y_m]$. Then we can depict the conservation principle with following equation, where R_k, R_n and R_m represents the relevance scores for input features, hidden layer neurons and output neurons respectively:

$$\sum R_k = \sum R_n = \sum R_m \tag{3.11}$$

The redistribution process can be defined in multiple ways, and for each different model it has to be specifically tailored for it. One possibility is to use a first order Taylor decomposition [8]. However, specifically for the neural network models, the Taylor decomposition is not used. Instead, a kind of message sending is introduced. When the relevance is known for neuron on upper layer, its decomposition can be redistributed on the lower layer by using the ‘message sending‘ from this lower layer [8, 6]. In other words, the connecting weights between the one neuron on the upper layer to all neurons on the lower layer is used as the means to redistribute the relevance score. Therefore, the decomposition can be denoted using following equation:

$$R_{i \leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j} * R_j^{l+1} \tag{3.12}$$

The $R_{i \leftarrow j}$ represents the message sent from lower layer i to upper layer j. $R^{(l,l+1)}$ and denotes the relevance score of lower layer neurons from the upper layer neuron $l + 1$ and $R^{(l+1)}$ denotes the relevance scores of one upper layer neuron. The expression z_{ij} can be calculated using formula $z_{ij} = X_i * w_{ij}$ and represents the activation of specific neuron. Alternatively, z_j can be expressed using equation $\sum_i z_{ij} + b_j$ and represents the activation of neuron on the

upper layer from all neurons from the lower layer [8, 6].

Using these equations, the relevance can be propagated using a backward propagation procedure in the neural network to the input layer and therefore on all the features. Note that there is no bound on the relevance and therefore the relevance scores are portraying the relative importance of features between each other [8, 6]. One thing that needs to be done, however, is to set the relevance scores on the output layer arbitrarily, either by using the output probabilities, or any other number [6].

One problem using the propagation rule is that for small values of z_j, the relevance message can take unbounded values which produces a numerical instability in the model. To counter this, an ε stabilizer term is introduced. When the LRP is modified in this way, it is denoted as ε-LRP [8, 6].

As is evident from the way the relevances are computed, this approach is dependent on the architecture of the model. If introduced to a model with other than $tanh$ or $ReLU$ activation, it fails spectacularly, as the decompositional approach was designed only for these 2 functions [8, 6]. Additionally, if we want to use this approach on other types of network, for example convolutional or recurrent, the decompositional equation must also be changed [6].

Comparison of Described Methods

One of the positives of these approaches is that they are much faster, as they require only one forward and one backward pass of the neural network.

On the other hand, they are highly dependent on the architecture of the neural network. Some of these approaches, for example can not deal with the ReLU activation or some of them can only deal with them and not the others. Another problem is that the generated feature importances are affected by interaction between different input features [2].

While the mathematical formulation for Integrated Gradients, DeepLIFT and LRP methods is wildly different, their generated feature importances are very similar. This is mostly evident when it comes to the Integrated Gradients and DeepLIFT, which are highly correlated, suggesting that DeepLIFT is faster approximation of the Integrated Gradients method. However, it fails in presence of multiplicative interactions between features. The LRP method produces a bit different outputs, though it is a simplest of the three methods. In addition, it fails when presented with other activation function than ReLU [2]. The comparison between the three methods is illustrated in figure 3.9. Additional illustration of visual comparison, along with the mathematical formulation of methods and their use on different activation function is in figure 3.10.

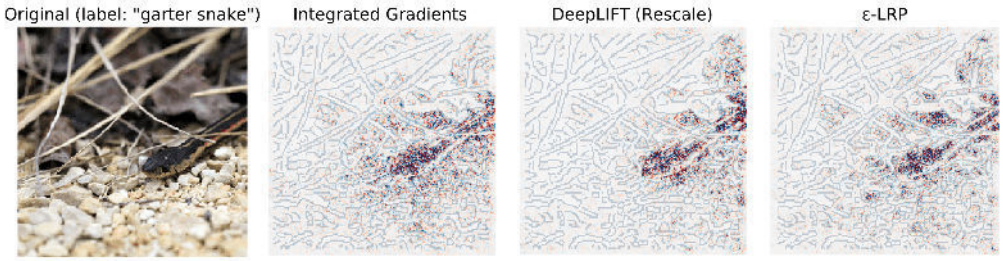


Figure 3.9: *Comparison of attribution maps generated by the three gradient based methods, Integrated Gradients, DeepLIFT and LRP. We can notice that attributions map of LRP is different from the other two, which are more similar [2].*

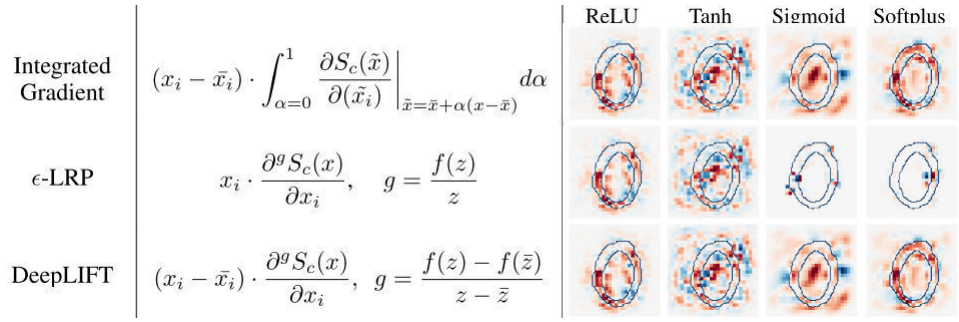


Figure 3.10: *Visual comparison of the three gradient based methods, along with their mathematical formulation and usage on neural networks with different activations. We can see that the attribution maps for them are very similar. Notice, how the LRP method can not deal with the sigmoid and softplus activation functions [2].*

3.4.3 Input Perturbation Methods

The last set of approaches, the input perturbation methods, test the rate of change in the output when the input is directly changed. It is much more straightforward and different than the previous two sets of approaches, as it observes the direct change of input and not just the change in weights. One important assumption is required, for the input variables to be continuous, which guarantees that the variable is still meaningful when it is perturbed. For example, if we had categorical variables with values from 1 to 5, setting it to value 2.5 would not make sense [47, 2].

The basic idea for these methods is that a change is introduced to one of the input variables, while keeping the other input variables untouched. These changes could take form of $I_n = I_n + \delta$ or $I_n = I_n \times \delta$, where the I_n represents the observed input variable and δ represents the introduced change. This is done for every variable, and optionally for multiple values of δ, while making note of the change in the output. Afterwards, the input variable that introduced the highest change in output is considered to be the most influential relative to other input variables [47].

The change in the output can be measured in multiple ways. A baseline error for neural network model can be established for the observation by passing it to the neural network and then a change in this error after perturbing specific input variable can be measured [5]. Another possibility, when it comes to the classification task, a change in the probability of the correct class can be measured and used, optionally with including the changes in other classes [49]. Last possibility is to use one of the sensitivity measures. These include following measures [9]:

- **Range**: $S_r = max(\hat{y}_{a_i}) - min(\hat{y}_{a_j}), i, j \in \{1, ..., L\}$
- **Gradient**: $S_g = \sum_{j=2}^{L} |\hat{y}_{a_j} - \hat{y}_{a_{j-1}}|/(L-1)$
- **Variance**: $S_v = \sum_{j=1}^{L} (\hat{y}_{a_j} - \bar{y}_a)^2/(L-1)$
- **Average Absolute Deviation**: $S_d = \sum_{j=1}^{L} |\hat{y}_{a_j} - \tilde{y}_a|/L$

The $\hat{y}_{a_j}$ represents the output for jth change in the ath input variable, $\bar{y}_a$ represents the mean of ath variable, $\tilde{y}_a$ represents the median of ath variable and the L represents the level or the number of changes for the variable, e.g. the number of different values that the δ variable can attain [9].

For all the mentioned methods for calculating the change in output, it holds that the higher the value then the more relevant input variable is. Therefore, we can calculate the relative importance of the input variable as:

$$r_a = \delta_a / \sum_{i=1}^{M} \delta_i \tag{3.13}$$

Where the r_a represents the relative importance of the input variable a, the δ_a represents the change in the output or the sensitivity measure calculated for the input variable a, and the M represents the number of input variables [9].

The most commonly used perturbation approaches can be divided into two distinct categories, which define how the input variables are changed:

1. *Local perturbations*
2. *Global perturbations*

One of the positive traits of these methods is the fact that the architecture of the neural network does not need to be known, as these methods take the neural network model as a black box and so are only interested in the input-output mapping of changes [47].

The problem for it comes from the fact that in many approaches, there is an implicit assumption that the input variables are not dependent on each other. This causes errors when

encountering data with dependent variables, as when one of the variables is removed, its value can still be reproduced from its dependent variables, and so it is considered to not be important. This can be partially prevented by using global perturbations. However, there it is still limited as it is based on randomisation approach [5]. Another problem for perturbation method is that they tend to be slow. As a change is introduced to each feature, the number of operations is proportional to the number of features. For example, when used on images, where each pixel is considered to be a feature, generating one explanation using perturbation method can take hours [2]. In addition, the produced feature importances are highly dependent on the number of features perturbed.

Local Perturbations

Local perturbation methods are defined by changing of value of only one input variable, while others remain fixed. In the most basic case the other variables are usually fixed at the representative value for them, for example median, mean or 1^{st} or 3^{rd} quartile [5]. The term $'$local$'$ in this case refers to the fact that the change, or derivative, of the output is calculated at a single point in hyperspace of the input variables [37]. This method is also referred to as a $'$One-at-a-time$'$ method [23].

The most common case for the local perturbation approach goes as follows [9, 26]:

1. Generate a new observation x_{mean}, with each attribute in the x_{mean} equal to the mean of the various attribute values for all records in the data set.
2. Find the network output for input x_{mean} and call it $output_{mean}$.
3. For each attribute, vary x_{mean} from minimum to maximum to reflect its range of values. For each attribute value find the output of the neural network and compare it to $output_{mean}$ producing a $diff_{a_i}$, difference from mean for attribute a and ith value.
4. Sum all of $diff_{a_i}$ values producing δ_a value.
5. Rank the attributes in descending order depending on their δ_a value.

The rate of change of a specific attribute value is often determined by their minimum and maximum value. The minimum-maximum interval is divided by number L and the result is the value that is added to the minimum value each time. Therefore, the computation is done L times, always with different value in the attribute interval and so the final feature importance is computed as average across these values [9].

There are some problems with this approach. Specifically the fact that only a small portion of a highly-dimensional hyperspace is scanned, due to the fixing of other variables on their mean value, when they are not perturbed. Another problem is, if the data contains dependent

attributes on each other. Varying only one of them is not a logical, nor representative solution, as the value of the variable is still contained in other dependent attributes. This is partly solved by using other approaches from local perturbation or the global perturbation approaches [5, 37].

One method from the local perturbations type that can deal with the problem of not exploring the whole highly-dimensional hyperspace of the input variables is called ′Leave-one-covariate-out′ (*LOCO*) [28]. It was originally described in the context of regression models, however, its default idea is model agnostic and therefore can be used in other models as well, because it can be implemented in various ways. It creates local interpretations for each row in the data by scoring this row once and then again for each input variable in the row. The general idea of the *LOCO* goes as follows:

1. Take each row and calculate output value for it by introducing it to the neural network model.

2. For each input variable, set it to missing, by assigning to it some appropriate value, for example mean, median or zero, therefore leaving it from the prediction.

3. Introduce each row with its one missing input variable to the neural network model. Calculate the output and compare it to the output from the run where all variables were present.

4. Rank the input variable for the specific row in descending order by their absolute impact on the change of the output.

This way, the approach produces variable importance for each row in the data. However, it can also be used to determine the relative importance of input variables for the whole model, by averaging the variable importances across the entire data set.

Global Perturbations

When it comes to the global perturbation approaches, the variation of a singular input variable is considered, while the values of other input variables are allowed to vary. This way, the highly-dimensional hyperspace of input variables is more thoroughly explored, resulting in much more meaningful interpretations, as it can capture some of the corner cases in the data set [5].

There exists diverse global perturbation methods, but the following ones are used the most often [23, 9]:

- *Simple Global perturbation* - it is very similar to the local perturbation approaches, with the exception that multiple input variables are simultaneously varied in their minimum-maximum interval instead of just one. The others are still held at their respective

value, e.g. mean or median. This approach better captures the interactions between input variables, however, it requires more computation and still cannot capture all the interactions.

- *Data-based perturbation* - it is also very similar to local perturbation, with the exception that instead of using the means for other values, a set of random samples is taken from the data set and a local perturbation is performed on each of them. When taking the whole original dataset, this method is closely related to the *LOCO* approach.

- *Monte-Carlo perturbation* - similarly to the Data-based perturbation, a set of random samples is generated from the original dataset. The exception is that instead of taking samples directly, values for each input variables are sampled from a uniform distribution defined by its minimum-maximum interval.

- *Cluster-Based perturbation* - is very similar to Data-based perturbation. However, before sampling the set of random samples, the dataset is divided into a set of clusters according to the observed input variable values and the samples are generated and the perturbation method is performed for each cluster.

Perturbation Based Attribution Methods for Convolutional Neural Networks

As was established, attribution methods can be used on any type of neural network model, even the deep convolutional neural network. In the case of [49], this approach is mostly used to determine if the model is truly identifying the location of the object in the image or just using the surrounding context.

However, because of the fact that the convolutional neural networks are specialized to be used on image data, the basic method perturbing the input had to be augmented. There is much higher number of features in convolutional neural networks, as each pixel in the image is considered to be a feature. In addition, each pixel is highly correlated with its neighbourhood, which is in direct contradiction with the assumption of independent features [49].

Therefore, instead of removing only one feature at a time, multiple pixels are removed at the same time. This is done by introducing a grey, or otherwise coloured, rectangular patch to the image, with its center defined by the observed feature. First, an image is introduced to the network to determine its true class classification. Then, the rectangular grey patch is slided across the input image, each time introducing it again to the network and monitoring the increase or decrease in the probability of the correct class. Each change in the correct output class probability is noted and used in the creation of a heatmap of class probabilities for the correct class with a specific part of the image occluded. By using visualisation of a heatmap of the changes, this approach ensures that the interpretation is as easy as possible

[49]. The created heatmap with its input image with an example of the grey rectangular patch is shown in figure 3.11.

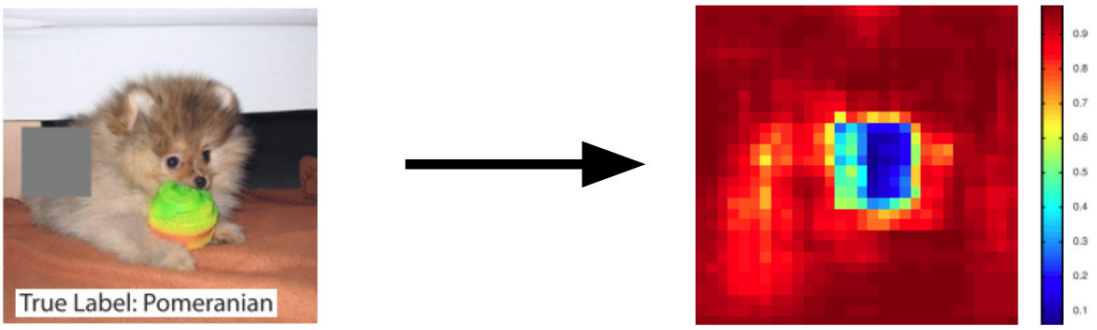


Figure 3.11: *Visualisation of the perturbation based attribution method for the convolutional neural network. A grey rectangular patch is slided across the input image, producing a heatmap of correct class probabilities as a function of the position of the grey square. Each pixel in the heatmap represents the class probability in the case, where the grey rectangular patch is centered on its location. From the visualisation we can determine that when the dog's face is obscured, the probability for the correct class drops significantly [49].*

Similar approach as this was proposed by Zintgraf *et al.* [53]. Similarly to the previous method, a patches of image pixels were also removed. However, instead of using a grey patch for the removal of pixel, they decided to use something else. The goal of their method is to compute a probability $p(c|x_{\setminus i})$, which denotes the probability of class c when all the features are present except x_i. This probability can then be compared with the probability of class with all the features, $p(c|x)$ and a difference can be calculated and used as a feature importance. There are three possible ways of removing the feature x_i. First is to set it to unknown, which only a few classifiers allow. The second is to retrain the whole classifier with the specific feature missing, which is infeasible. The third approach is to marginalise the feature by modeling a probability $p(x_i|x_{\setminus i})$. This modeling is, however, infeasible with large number of features and thus an independence of variable x_i with all others is assumed [53]. Therefore, the approximation of probability of correct class with missing feature can be denoted as:

$$p(c|x_{\setminus i}) \approx \sum_{x_i} p(x_i)p(c|x_{\setminus i}, x_i) \tag{3.14}$$

Nonetheless, the approximation of $p(x_i|x_{\setminus i})$ is crude, mainly when it comes to images, and so they are proposing a fourth approach. They propose a much more accurate approximation based on the observations that pixels are strongly correlated with their neighbourhood and the value of pixel is independent of its position in the image. Therefore for pixel x_i, a patch $\hat{x}_i$ of size lxl that contains x_i is used [53]. The approximation used can then be denoted as:

$$p(x_i)p(c|x_{\setminus i}) \approx p(x_i|\hat{x}_{\setminus i}) \tag{3.15}$$

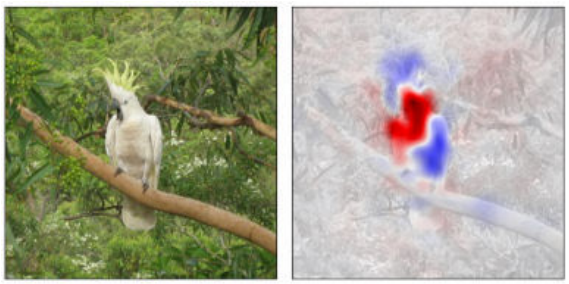


Figure 3.12: *Visualisation of the perturbation based attribution method using a marginal sampling. The visualisation explains why GoogLeNet predicts the correct class of 'cockatoo'. The red parts of the heatmap are supportive of the decision, while blue parts are prohibiting the decision. We can see that the face of the cockatoo is supporting the decision, while the body is prohibiting it. This can be due to the fact that the other highest scoring class is 'white wolf' [53].*

The patch removal is implement in a sliding window fashion, with the patches overlapping. For this reason, the relevance of pixel is obtained by taking an average of all of its relevances obtained from all the patches it was in. Additionally, for sampling, the empirical distribution is always used and a patch is replaced using samples taken directly from other images at the same location. The idea of using a multivariate approach, e.g. removing patches and not single pixels, is based on the fact that neural network should be robust to such a small change. This stems from the observation that there is high correlation between neighbourhood pixels [53]. The results of this method are illustrated in figure 3.12.

Chapter 4

Analysis Summary and Findings

Interpretability is an integral part of machine learning research. Its importance lies in the simple fact that without the ability to explain the reasoning behind the decisions of our models there is no real possibility for them to be used in practice. Without explanations, we cannot trust that our model will behave correctly. And without trust, machine learning models cannot be used in the fields where the cost of an error is huge, like medicine.

Neural networks are one of the most used machine learning models today. They are much more accurate than other models when used correctly. The accuracy of neural networks is due to the fact that their representation of learned knowledge is highly abstract. Additionally, their architecture allows them to be extended really easily by stacking layers on each other. This way, they can represent complex interactions present in the data as close as possible and thus model highly non-linear functions. In addition, there is a tendency to construct more and more complex neural network models for more tricky tasks, like for example convolutional neural networks.

However, this is also the reason why they are regarded as one of the hardest models to interpret and explain. There is extensive research that deals with the problem of explaining the decisions of neural networks and many approaches were already developed. They include things like looking at the weights in the network, visualising weights or activations or any other characteristic of neural network and many others.

Attribution methods for neural networks are the ones that are researched and used the most. These methods are one of the most accurate, as they work directly with the trained network when determining the most important input features for the individual predictions. By assigning a simple number to features, they allow for simple, but effective, visualisation of the neural network.

The visualisation of important features is much better than a simple table output. Humans can interpret images much better than a long list of text. Therefore by visualising which features are important, rather than just listing them, the understanding of what neural network

is doing is easier. This is much more obvious when it comes to images, where each pixel represents one feature.

Almost every attribution method is intended for use on images, while other data types are mostly ignored. It is understandable, that it is this way, as a large number of neural networks are used on images, as no other machine learning models can deal with them easily, due to the complexity of representation of the data. However, there are other data types with similar complexity of representations. One typical example is a text represented as a vector model to take advantage of the order of words and not just the presence of words. Therefore, a method that can easily deal with text is also required.

Each type of attribution methods has some kind of positives, but also negatives. Methods that are using gradients to determine the feature importance are dependent on the architecture of the neural network. For example, some methods cannot deal with rectified linear units, while others work only with them. On the other hand, the methods that use perturbation are independent of the architecture but suffer from massive slowdown when there is large number of features present. For example, when dealing with images, explanation of one decision can take several hours. They are, however, more precise than gradient methods, as the gradient methods work as a kind of approximation. The problem in both types is the inability to deal with the dependence between the different features. Therefore, their results are skewed.

Chapter 5

Thesis Goals and Hypotheses

The goal of our work is to develop a modified perturbation-based attribution method that can reduce the drawbacks of the simple attribution methods. We primarily focus on taking into consideration the interactions that are present in data in the explanation generation process and thus allowing the effective use of this method even on data where the independence of features is not satisfied. In addition, we focus on presenting the created explanation in a form that is interpretable for other people and can give them insight into the decision process of the neural network. The main goals of this thesis can be summarized in the following hypotheses we have set:

1. Inclusion of inherent interactions, present in data, in perturbation-based attribution method can help in producing more precise attributions.

2. Relevance scores, and their visualisation, produced using a modified-perturbation based method on textual data can provide insight into the decision process of the trained model, allowing humans to better understand its behaviour.

To confirm the first hypothesis, we use a statistical evaluation, which consists of the comparison of our method to other attribution methods without the inclusion of interactions. For the second hypothesis, we propose a user experiment, as we want to confirm the user understanding of the explanations and their ability to provide insight into the decision process. More in-depth description of our methods for confirming these hypotheses is presented in section 6.2.

Chapter 6

Determining Relevance of Correlated Features

Our proposed method is focused on improving the understanding of neural network models with any kind of architecture, by explaining the decisions of these models. It is a perturbation-based attribution method that during its explanation generation process, takes into consideration the inherent interactions between features. The reasoning for this method is the fact, as was determined in previous sections (3.4.3), the perturbation-based methods produce more precise attributions when the assumption of independence between different features holds. This assumption, however, does not hold in many cases and therefore there is a need for a method that can overcome this shortcoming, in order to produce better explanations even where there are interactions in the data. As with all the attribution methods, this method assigns a relevance score, or attribution, to each input feature in a specific decision.

The basic process of our method is presented in figure 6.1 and it consists of the following steps:

1. Generate a baseline for comparison.

2. Identify correlations between input features.

3. Perturb the examined feature.

4. Calculate final relevance score for the examined feature.

5. Present the generated explanation.

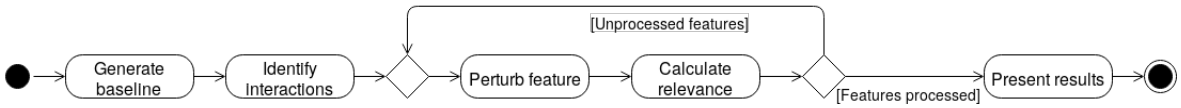


Figure 6.1: *Basic overview of the proposed method. First, a baseline is created. Afterwards, the interactions are identified. Using these interactions each feature is perturbed and its relevance score is calculated. When all the features are processed, the generated results are presented.*

6.1 Proposed Method Details

While the majority of attribution methods were built atop the image data, we have decided to concentrate on textual data. The reasoning for this is the fact, that by using textual data, we can best showcase the possibilities of our new approach. When considering individual words as separate features, the interactions between individual features in textual data become very distinct. The sequence of words is one of the most obvious ones, where the words ‘new‘ and ‘york‘ mean something entirely different when separate and when together. Further interactions are slightly hidden as they are present in the use of synonym words, where two words are considered separate features while representing the same kind of information.

Nonetheless, the decision to concentrate on textual data brings with it some complications that need to be resolved. One of them is the sheer number of features, which causes significant problems for the use of perturbation-based methods. When each individual word is considered one feature, the number of features can be in thousands or tens of thousands. Even though it is not so bad as with the image data, where producing attribution for one image can take even one day, it can still cause significant slowdown. Another problem is the representation of the data. Typical models require the data to be in static one-dimensional vector form. Therefore a bag of words representation is often used, which creates a sparse vector with size equal to the size of dictionary of words used, with each number representing the frequency of individual words. However, this representation ignores the main interaction of words, their sequence, and therefore cause further problems if used.

Fortunately, both of these problems can be counteracted using word embeddings. Word embedding representation produces a two-dimensional representation of text, where the words are mapped to vector of real numbers, thus keeping the order of words and reducing the total number of features for specific decision. Only the words present in the specific observation are used and not all the words from dictionary of all words. This representation is almost identical to the image pixel data representation with one difference. Where pixels are represented as three different channels, red, green and blue, when it comes to words, only one channel is used. Therefore neural networks can still be used for this representation. Based on these facts, we decided to utilize the word embedding representation in our method. Note that in this new representation, individual word, and thus the whole word embedding vector for the specific

word, is still considered as one feature.

Although the aforementioned proposed method generates only local interpretations of the model, e.g. explaining individual decisions and not the model as a whole, it can be further extended to produce approximations of global interpretations of models. The individual feature relevance scores can be aggregated together using, for an example, a simple average of all the relevance scores for specific features across all decisions. These aggregated relevance scores can then be used as a means of explaining which features the model finds most important as a whole.

Nonetheless, in our case, where the focus is on textual data and each word is considered one feature, this approach provides only a crude approximation. Due to the sheer amount of features, this approach could cause s shift in preference. Instead of preferring words with high relevance, which are infrequent, it could prefer the words that appear often, even with low relevance, instead. Although, should the nature of the data change, the case would not apply anymore.

The process of generating an explanation using our proposed method specifically for textual data with word embedding representation is described more in-depth in following sections. The graphical representation of this process is shown in figure 6.2.

6.1.1 Generate Baseline

In this step, a baseline from the selected observation is created. This baseline serves as an important starting point, as in later steps, the effect of the perturbation will be gauged using a comparison with this baseline. The easiest possibility for creating the baseline is to use the basic output of the neural network after the chosen observation is passed to it. The output of the neural network is therefore marked as $y_{baseline}$.

6.1.2 Identify Interactions

The identification of correlated, or interacting, features is performed for each input feature in the selected observation. There are multiple possibilities of how to generate this set of correlated features, each with slightly different output and different properties.

In the case of textual data, we have identified the following 4 possibilities. First possibility is to exploit the character of input data and determine the correlation using an **order** of words. In this case, a number of previous and following words is used. This simple approach guarantees us that the size of the set of correlated features is consistent between features, but also between multiple observations, which causes less problems in further steps. On the other hand, the relevance of some of the features may be overestimated.

As we are dealing with texts with differing sizes, setting the size to be static may not be

the best idea. Imagine having 2 texts, one with 10 words and the other with 50 words. We decide to take 2 previous and 2 following words for the set of correlated features. This way, the first text would have 50% of its features perturbed and the second only 10%. It is clear that the result of the perturbation in the first case would be much more significant than in the other case. To better deal with this, we identify a second possibility of using a **percentual order** of words. Instead of taking static number of words, we change this number in accordance to the word size of the chosen observation. This way, we get much more consistent relevance scores between different observations. However, we must take into consideration this change in the size of the set of correlated features in further steps. Additionally, the problem of overestimation of some of the features is still present.

Another possibility is to exploit the vector representation of textual data we use. In this case, we use a **cosine** similarity between the individual vector representations of words to determine which are most similar. Each word, whose cosine similarity is above a defined threshold, is therefore inserted into the set of correlated features. This approach should produce better results in some cases, as the relevance scores are not overestimated. However, it creates an inconsistency in the size of the correlated features set. Additionally, it can cause underestimation of the relevance scores instead, as the similarity between stop words should be much higher and therefore more features are perturbed when dealing with them.

Last, but not least, there exists the possibility to use a combination of the **cosine** similarity approach with either one of the **order** or **percentual order** approaches. This approach should effectively deal with the problem of overestimation and underestimation of relevance scores for the features.

6.1.3 Perturb Features

The perturbation is performed for all the input features present in the observation. Instead of perturbing only the examined feature, the perturbation is performed on the whole set of correlated features that correspond to this feature.

As a means of perturbation, there exist multiple possibilities. First possibility is setting the value of the perturbed feature to its mean, or average, value. This approach, however, does not make sense, especially when dealing with words, as the average value is something that would make no sense in regard of the data set.

The better possibility is removing the feature altogether. As this is not really possible without retraining the model in each step, which would be counterproductive, setting the value of the feature to indicate an absence of signal is the best solution. In the case of textual data this means setting the embedding vector to a zero vector.

6.1.4 Calculate Relevance Score

After the observation is perturbed using the currently examined feature, we need to generate a new output. This output should be generated the same way as the baseline, in order to be able to compare them. In the most basic case, the observation is again passed to the neural network and its output is saved and marked as y^k_{mod}, where $k = 1, ..., K$, with K indicating the word size of the observation and k representing the index of currently examined word.

A relevance score for the examined feature, r_k, is then calculated using equation $r_k = f(y_{baseline}, y^k_{mod})$. This equation calculates the difference between baseline output and the output from perturbed observation. The function $f()$ can be chosen as an arbitrary metric. It should take into account the fact that the perturbation is performed on multiple features at once, which can cause unwanted artefacts in the results. Consider a case, where an unimportant feature is preceded and followed by an important word. Without any normalization, it would get much higher relevance score as it should. Additionally, the size of the correlated features set could cause artefacts when it is not static. If more words are perturbed, the change in the output is expected to be more significant even though only unimportant words are perturbed. Therefore the size of the correlated features set should be used as an additional normalization.

The final relevance score, r_k, represents the importance of the examined word, with index of k, for the chosen observation. It is expected that if the examined word was really important for the decision process, its perturbation should have significant impact on the output. Therefore higher positive number of the relevance score indicates higher importance for the specific decision, e.g. it was one of the features that was used in the decision making process. The relevance score can also be negative. This indicates that it was one of the words that oppose the decision made, again with lower number indicating higher level of opposition. If the score is zero, or close to zero, it has no quantifiable effect on the decision.

6.1.5 Present Results

The explanation of the decision of the neural network model is presented using the relevance scores calculated in previous step. By sorting the relevance scores in descending order, we can determine the words that were the most important for the decision, as they would be on top. When normalized to a range $[-1, 1]$, we can present the relative importance of the words to each other. This way, we can determine if the decision process was mostly influenced only by a single word, or if multiple words were significant.

As the created relevance scores are numbers that are bounded by a minimal and maximal value, their visualisation can also be created. They can be used for a heatmap visualisation across words, where the background colour of the word is determined by its relevance. We assign the green colour to positive relevance and red colour to negative relevance, with the

deepness of the colour indicating the significance of the word. One such visualisation is presented in figure 6.2.

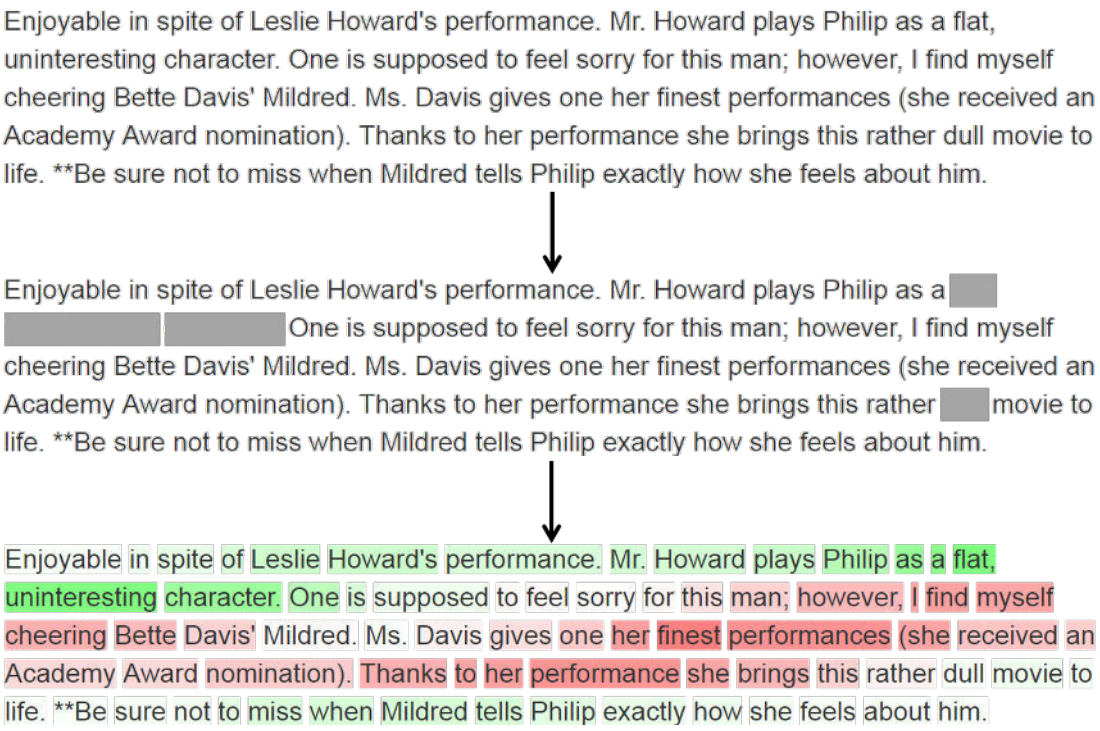


Figure 6.2: *Process overview of our proposed method when used on a text observation. The observation is from a sentiment detection task and is assigned a negative sentiment. For each word, its correlated features set is identified. In this case, the examined word is 'uninteresting'. The words 'flat' and 'character' are in the correlated set due to their proximity to the word and the word 'dull' due to its similarity. These words are perturbed by setting them to zero vector (represented by a grey rectangle). The calculated relevance scores are then used for a heatmap visualisation across words. The darker the green colour around word, the more relevant it is for the current prediction, e.g. negative sentiment. The darker the red colour around word, the more it is in opposition of the decision, e.g. points to positive sentiment. Words with white, or almost white, colour are irrelevant for the decision.*

6.2 Hypothesis Evaluation Approaches

Attribution methods are often hard to evaluate empirically [2, 44]. The main difficulty results from the fact that the explanations are generated using an already pretrained model. There is no special guarantee that the model is error free and therefore distinguishing between what is the error of the trained model and what is the error of the explanation method used for the model is hard to make. Common practice is to assume that the pretrained model is perfect, or error free, and therefore each error present in the explanation is caused by the method used [2, 6, 53, 39, 8].

In addition, evaluating only the correctness of the generated explanations is not enough. As the explanations are used by humans, they should be presented in such a way, that they are interpretable themselves and provide an insight into the decision process of the used model. The insightfulness of the presentation should also be evaluated. The result of the attribution method is a list of relevance scores for the individual input features. Such output is not really interpretable for humans and thus cannot provide any insights. Therefore it should be presented in an interpretable way, preferably in a form of visualisation.

For the evaluation of the attribution method, following 3 properties should be examined:

- **Correctness** of the used model. The assumption that the pretrained model is error free does not make much sense. Without the confirmation that it is really correct, the explanations cannot be judged correctly.
- **Faithfulness** of the generated attributions. This property indicates if the generated explanations really mirror the behaviour of the model. It is the main thing that is evaluated in each attribution method.
- **Insightfulness** of the generated attributions. This property indicates if the generated explanations are understandable and enhance the user understanding of the model. Even though the explanations can be correct, without this property they are useless.

To correctly evaluate our proposed method, we have decided to examine all three of those properties. To do this, we have identified the following 3 ideas for experiments:

- Denoising
- Comparison with attribution based methods
- Using user marked results for evaluation

6.2.1 Denoising

To determine if the neural network model has learned relevant feature representations, and therefore if it is really correct, we decided to use denoising. For the task of denoising a special kind of neural network model is used, a denoising autoencoder [20, 46]. The autoencoder gets an input, forwards it through the network and is expected to output the same representation as the one on the input. When extended to be able to denoise, its training is modified. The input is first modified by introducing some kind of noise to it and the encoder is trained to be able to remove it. Therefore, the autoencoder is used as a kind of dimensionality reduction, where the most important features are extracted and more robust representation of data is learned [20, 46]. We can think of denoising autoencoder as a kind of unsupervised feature selector.

As we are doing something similar, determining which features are relevant for a specific decisions, we can use the denoising autoencoder as a way of evaluation. First, a simple denoising autoencoder is trained with either fully-connected layers or convolutional and maxpooling layers, with their number (in the encoding part) equal to the number of layers of our trained model. Subsequently, a decoding part of an autoencoder, using the architecture of our trained model, is trained, while keeping the already trained parts of the architecture static. An error rate for both of these models is calculated and compared with each other. If the error rate from our trained model is not greater than the one from the simple model, we can state that our trained model has learned the best representation of input data it could and can be used for evaluation of the attribution methods. Otherwise, some kind of modification needs to be performed upon our trained model. Note, that this decision can be made only when the error rate of the simple denoising autoencoder we trained is sufficiently small. If this is not the case, a more complex architecture must be used.

6.2.2 Comparison with Attribution Based Methods

The comparison with other attribution based methods can be viewed as a natural idea, as the method we are proposing produces explanations in similar fashion to other attribution methods, by calculating relevance scores, while taking into consideration interactions in the data. These relevance scores should be comparable, with some preprocessing. In addition, most of the attribution methods can identify features supporting, but also those that oppose, the specific decision. Therefore, it makes sense to compare it to other methods, in order to examine the faithfulness of our proposed method.

Nonetheless, there is also slight problem with this approach. When comparing between our proposed method and other attribution method, we are comparing between two sets of numbers, the produced relevance scores for all input attributes. Therefore, we can produce an aggregated difference between them, to evaluate, if the generated explanations are different. However, if large difference occurs, we cannot tell which method produced better explanations, as we have no golden standard against which to compare. Therefore, we select a bit different approach in this comparison.

First, using the numerical, but also statistical comparison, we determine if our proposed method produces different explanations than other methods. As we are assuming that our method generates better explanations, the difference should be present. If there is no difference, there is no point in continuing the evaluation and we can determine that the produced explanations are not better.

After observing a difference in the explanations, we can determine the correctness of the explanations. For this, we first select the top k most important words from our proposed method, and all other attribution methods against which we want to make a comparison. Then,

we remove those words from the observation, similarly as in our method, by setting them to zero vector. After the removal, we calculate the percentual difference in output of the neural network. This can most easily be performed in a classification problem, where the output probability is bounded. We expect that the method that identified the most important words will attain highest difference in the output. As the point of the attribution method is not only in finding most important words that support the decision, but also those that oppose it, the same is performed with the bottom k words. Both results for single method are aggregated, as the method should be able to do both things very well.

A big advantage of the attribution methods is the fact that they are easily visualised and therefore we do not need to rely only on quantitative comparison, but an expert evaluation can be performed on them. This property is exploited in the user evaluation experiment.

6.2.3 Evaluate through User Marked Results

The last idea for evaluating the proposed method is to use the user marked words to evaluate, if the proposed method generates explanations that correspond to the expected behaviour of the model. We examine if the explanations provide the necessary explanations for the decision process and if they correspond to the human expectations. As humans, we have a pretty good idea about which words are important for promoting the decision, which are in opposition of the decision and which are irrelevant. For example, when presented with sentiment analysis, we can easily determine that the words *not good* together promote a negative sentiment, while just word *good* impede the negative sentiment.This is the very thing we want to take advantage of. This experiment consists of two parts called *user marking* and *user trust*. To guarantee best results, the data for this evaluation approach are from a sentiment detection problem. Both parts of the experiment are used for the examination of the faithfulness and insightfulness property of the method. This experiment only makes sense, and generates good results, when the correctness of the model is guaranteed.

User Marking

In this part, users are choosing words that they think are important. They are presented with a specific observation from the text corpus. The task for them is to choose the words they think are the ones that promote the positive sentiment the most, but also the ones that they think correspond to the negative sentiment. The number of words for either category is not limited. The words selected by users are then used to create a golden standard against which will we then compare our proposed method, along with other attribution methods.

Since users are not asked for any kind of ranking of the words, first, a set of the most relevant words for either category, the one promoting and the one opposing the sentiment,

needs to be chosen. This is performed by defining a threshold value individually for each observation. One possibility is to just take the 10 (or 90) percentile of relevance scores. Afterwards a percentual overlap of these sets is calculated, while taking into consideration the size of the set provided by the user and its proportionality to the overall number of words. When encountering a 100% overlap, if the user has chosen only one important word then it should have different weight than when the number of chosen words is for example 5. Another possibility is to use the Jaccard index for evaluating the overlap. The overlap is calculated for each attribution method we want to use in comparison as well.

When comparing the overlap, multiple approaches are chosen. First, the numerical values are observed, as multiple situations can occur. Both the overlap for our method and for other attribution methods is similar, however very small. In this case, the neural network is not performing as the user expects and therefore there is a problem with the representation the neural network has learned. On the other hand, when both overlaps are high enough, we can declare that the neural network is behaving like is expected. Another case is when the overlaps are significantly different. If the one produced by our method is much higher, then we have encountered a situation where our method has generated better explanation for the behaviour. Or when it is much smaller, we can declare that there is some problem with our approach. If the resulting overlaps are sufficiently high, we can perform a statistical evaluation to determine, if overlap with our method and the other is different with statistical significance.

If the difference is statistically in favor of our proposed method, we can declare that it performs better on correlated data. In addition, in the first part, we can declare the same thing if the occurrence of the case where the overlap of our method is better is most frequent and the case where it is worse is infrequent. The case where both overlaps are very small should be minimal.

User Trust

In the second part of the experiment, we are evaluating if the produced explanations are understandable for humans and if they provide a meaningful insight into the decision process of the trained model.

This evaluation is performed by presenting explanations to the user and asking them some questions. In order to correctly determine this, the process is performed multiple times, each time with differently trained model. Some of the models presented to the users will be good, with high accuracy, but others will be error-prone, with incorrect training process. To get the best results, the users will be presented with all the models, but in different order.

Each presentation for different model will be performed in 2 steps. In the first step, user will be presented with a number of text samples, along with the information about the predicted outcome and the correct outcome. The given task will be to inspect the samples

and answer few questions. The questions will be about the level of trust in the model and about the way they think the model decides. After answering these questions, the user will be presented with explanations of decisions for the previous text samples and asked the same questions again.

When evaluating the results, we will be comparing the level of trust of users in the correct and incorrect model with our expected results. We expect that before being presented the explanations, the correct and incorrect models should attain approximately the same level of trust. However, after being presented with the explanations, the trust in the incorrect model should plummet significantly, with the trust in the correct model staying at approximately the same level. The questions about the way the users think the model behaves will mostly be used as a means of control questions, to determine if the trust was selected randomly, and to inspect if some trends are identified in the text samples and their explanations. We mainly expect that the explanations should help in identifying the bad models from the good models, but in indirect way to avoid any bias.

Chapter 7

Correlated Perturbations on Text Data Set

In this chapter we present the implementation details for the method presented in chapter 6. We present these implementation details in regards of a classifying individual textual observations into multiple classes.

In order to be a bit implementation independent, the textual data is not immediately represented using the embedding vector representation. Instead, we expect the input to be in a form of vector of numbers. These numbers indicate the index of the individual words in a dictionary created from all the words present in the corpus. The index 0 is specially reserved for words without any embedding vector representation. The translation of these words into embedding representation is then handled by the neural network. Additional reason for this representation is the fact that keeping the vector embedding representation beforehand is much more memory inefficient and is not needed for our method to work.

As the word sizes of individual observations are variable, we deal with this problem as well. We define a maximum number of words that can appear in a single observations. The observations having the word size larger than this are cropped. The observations with lesser word size are padded using the aforementioned zero index words.

The implementation of our method can take any number of observations and calculates the attributions in bulk, which makes it more efficient. For the implementation of neural networks we use the *Keras* framework with *Tensorflow* backend. The representation which we use for the data is the *numpy* array.

Generate Baseline

The generation of the baseline for our method is done simply by passing the unmodified observation to the neural network. As we are dealing with a classification problem, this outputs a probability for each class present.

As having the baseline consist of all the probabilities would be too much implementation dependent and would cause problems further down the line, we reduce this number to 2 values. First value is the probability for the class, which the neural network model assigns to the observation, e.g. the one with the highest probability (even if it is incorrect prediction in regards of the data set). The second part is represented as a sum of probabilities of all the other classes. This way, the baseline for classification problem is always the same and we can introduce weighting in the calculation step, with which we can change the importance of these two parts of the baseline.

However, in the following implementation, we do not use both values. The neural networks we are using all have a softmax activation on the last layer. Thanks to this, the probabilities always sum up to 1 and keeping both values is not necessary and therefore is not used.

This then leaves us with only one value as a baseline: the highest probability. The index of the class is also kept, to allow for selection of the same class in later steps.

Identify Interactions

In generation of correlated features set using either the **order** or **percentual order** approach, we implement only symmetrical steps, as they make most sense. What this means is, the same number of words is removed before and after the examined word. As the number of words that are removed is always the same in the case of **order** approach, we use an indexing approach in identifying the correlations. This is performed by simply calculating how much the index should be moved in preceding direction and the following direction. For the **percentual order** approach, we first calculate the number of words that corresponds to the percentual count specified for each observation that is passed to our method. Using this number, we again calculate the step required to move in either direction. As we only implement symmetrical steps, the number is halved and floored to produce the final change.

The generation of correlated features set for the **cosine** similarity approach is a bit different. As it is independent of the observations, and its calculation is computationally expensive, it is precomputed, or its results are loaded from a file. To find the words correlated with any word, we take the embedding vector representation of that word and calculate its cosine similarity with all the other words. The words, whose similarity is larger than the threshold, are then selected to be included in the set. For the representation of this mapping, we use a python dictionary, with the current word as index.

Perturb Features

The perturbation of features is performed on a copy of the data, as it is performed multiple times and the original data must be kept intact. However, this copy is taken only from data, where the index corresponding to the currently examined word is non-zero, to guarantee

higher efficiency. The perturbation is performed by setting the number on a corresponding indices in the numpy array to the special zero index. The selection of indices is dependent on the approach used.

When using the simple **order** approach, a simple numpy array splicing is used. First the left and right index is determined by subtracting or adding the predetermined step. There is a possibility of selecting indices that are out of bounds. To deal with this, for the left index we take the maximum of the index 0 and itself. For the right index we take the minimum of itself and the maximum index. Both the left index and right index is saved for use in further steps. Using these indices, the observations are updated in a bulk, using the range splicing and broadcasting on numpy array.

When using the **percentual order** approach, the same principle as in **order** approach is used. However, it is performed for each observation individually, as the left and right index is different for each observation. Therefore it is calculated and saved for each observation individually. This also renders the broadcasting unusable.

When using the **cosine** similarity approach, we must first find indices of the words we want to remove. The set of correlated features in this case consists only of the words index representations. Therefore the indices of words in this set are found using numpy logical expression and saved for further use. The perturbation is again done individually on each word.

Calculate Relevance Score

When calculating the relevance score, we first pass the perturbed observation to the neural network. From the output, we select the probability of the class using the index that was determined when creating the baseline.

Afterwards, we compare this output with the baseline. For the comparison, we use a simple subtraction of the new probability from the probability determined by baseline. To take into account the fact that we are perturbing multiple features at once, we employ multiple normalizations. First, the difference is divided by a size of the correlated features set. Afterwards, the calculated difference is added to the attribution score of each feature in the set. A count, of how many times a difference was added this way, is kept for each feature. After processing all the features, the relevance scores are divided by their corresponding count. This guarantees that the effect of overestimation of relevance scores for the unimportant words, which have important words around them, is reduced, while keeping the relevance scores of important words unaffected.

Present Results

When presenting the resulting explanations without using any kind of visualisation, no further action needs to be taken. The explanations are presented as a list of pairs, where the first part of the pair represents the word, in string representation, and the second part its relevance score.

We can also present only a small number of important words. In that case, the results are sorted in descending order using the absolute value of the relevance scores and then presented in the same form as before.

To produce a heatmap visualisation across words, multiple steps are performed. First, a minimum and maximum relevance score is identified. Using these two values, the relevance scores for the specific observation are scaled to range [-1, 1]. The scaling is performed in such a way, as to not change the sign of any of the relevance scores. Therefore, it is performed on positive and negative scores separately. Using these scaled values, a colour for each word can be calculated. The colour is represented using the RGB colour spectrum. For positive relevance scores, the halved value of this score is subtracted from the *red* and *blue* spectrum. For the negative relevance scores, the halved value of this score is added to the *green* and *blue* spectrum. This way we can specify colour for each word, with which we can create either HTML output or image output using *imgkit* library.

Chapter 8

Experiments for the Proposed Perturbation Method

For the evaluation of our proposed method we perform multiple experiments. We use 2 different data sets with a slightly differing preprocessing for each of them. These data sets, along with the preprocessing used, are described in section 8.1. Furthermore, we use slightly differing architectures of neural network for the different data sets. For most of the experiments, we use a default sepcification of our method. Both the architecture and the specifications of our proposed method are described in section 8.2.

The specifics of different experiments we perform are described further in this chapter. The performed experiments are similar to those we mentioned in design chapter:

- Denoising
- Comparison with attribution methods
- User experiment

8.1 Data sets

For the evaluation experiments, we used 2 different data sets, each with slightly different specifications. We use the 20 newsgroup data set[1] and the IMDB movie reviews data set[2].

20 News Group

This data set consists of texts of different lengths, with each text dealing with one of 20 specified topics. These topics are sometimes very closely related, talking about different

[1] http://qwone.com/~jason/20Newsgroups/
[2] http://ai.stanford.edu/~amaas/data/sentiment/

computer problems, but some are unrelated, for example discussing religion and computer graphics. This is the main reason why we chose this data set, as the classification task in this case is a bit more complex, as the model has to learn to distinguish between closely related, but also wildly unrelated, categories. For the experiments, we use the version preordered by date, with already split data into train and test. This version contains approximately 18800 different texts.

A simple preprocessing is performed on this data set. All the words are transformed to lowercase. We remove all the stop words from all texts. Additionally, we also remove supporting characters, like apostrophes, brackets and numbers. Finally, we remove multiple white spaces.

After preprocessing, the text is tokenized and transformed to the word embedding vector representation. The tokenization is performed by simply splitting the text when encountering a white space. The maximum number of words allowed for the individual text samples is 1000 and the rest is padded with empty words. For the word embedding representation, we use a pretrained GloVe embeddings[3]. We use the version with 6B tokens that was trained on the Wikipedia 2014 and Gigaword 5. We use a vector length of 100. The transformation to embedding vector representation is done as a first step of neural network.

IMDB

This data set consists of texts of different lengths, with each text belonging under either a positive or a negative sentiment. It is a much simpler data set, as there are only 2 classes. This data set is already split into train a test set, with each set containing exactly 25000 texts.

We perform a more complex preprocessing on this data set. We first expand all the contractions in the text samples. Afterwards, we transform all characters to their lowercase format. We remove all the HTML tags and substitute all the occurences of '&' symbol to word 'and'. We also substitute emails and numbers to a symbolic representation. Afterwards, we remove all non-alfanumerical characters and multiple white spaces.

After preprocessing, we tokenize the text and transform it to word embedding vector representation. For tokenization we use a simple splitting by white space. The maximum number of words allowed for individual text samples is 1000 and the rest is padded with empty words. For the word embedding representation, we use a pretrained GloVe embeddings[3]. We use the version with 840B tokens that was created from a 'Common crawl'. We use a vector length of 300. The transformation to embedding vector representation is done as a first step of neural network.

[3] `https://nlp.stanford.edu/projects/glove/`

8.2 Method Specifications

The architecture of neural networks we use for both data sets is inspired by architecture presented in [24], and its visualisation is presented in appendix in figure A.1. A word embedding layer is used as a first layer. For this layer, we use weights created from the pretrained embeddings we use and set these weights to not be trainable. This layer is followed by three different convolutional layers, with filter widths of 3, 4 and 5 respectively and the filter size of 512. Each layer is followed by a maxpooling layer. The outputs of these maxpooling layers are then concatenated together and flattened. A dropout layer with rate of 0.5 is used next. Afterwards, a fully connected layer with softmax activation is used that operates as an output of class probabilities and therefore its size is equal to number of classes, e.g. 20 or 2. The convolutional layers each use a ReLu activation. As a loss function, we use categorical crossentropy. For both data sets, we use RMSprop optimizer with learning rate of 0.0001. The neural network for 20 news group data set is trained for 100 epochs and achieves 75% accuracy on test data. The neural network for IMDB movie reviews data set in addition contains a dropout layer with 0.5 rate between the embedding and first convolutional layer. It is trained for 40 epochs and achieves 89% accuracy on test data. The training process is presented in figure 8.1.

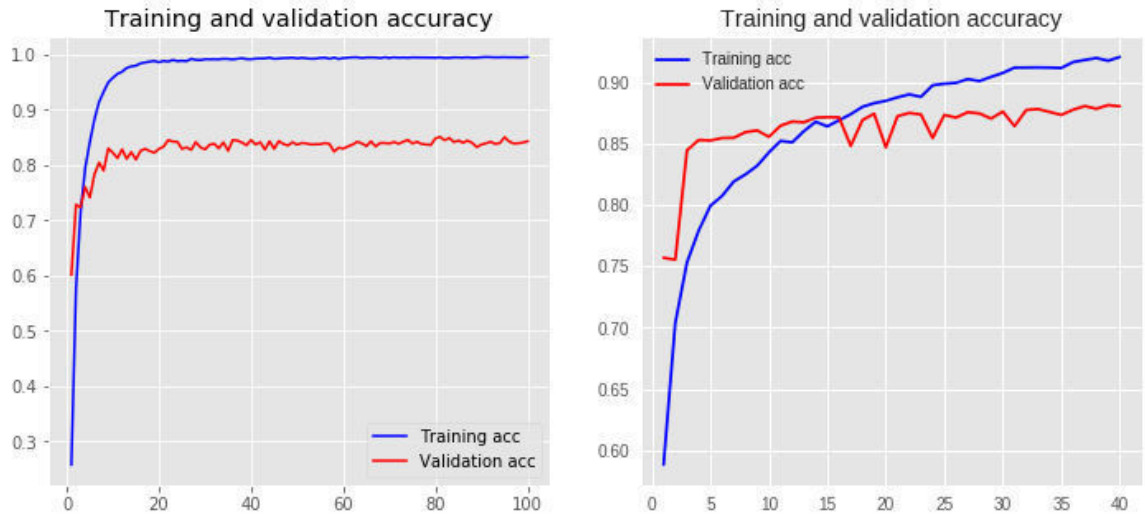


Figure 8.1: *The training process of the neural networks we use. On the left is the neural network used for the 20 news group data set. On the right is the neural network used for IMDB movie reviews data set.*

For most of the experiments, a simple default version of our proposed method is used. It is implemented exactly as described in section 7. We use a combination of percentual order and cosine similarity for the selection of words for the correlated features set. For percentual order, we remove 5% of words and for the cosine similarity we take words with similarity higher than 70%. Any additional changes in specification are described in individual experiments.

8.3 Experiment: Denoising

In the denoising experiment, we use different architectures of neural network to train a denoising autoencoder. The task of this encoder is to reconstruct the noisy data it gets on the input and output them. The adding of noise is performed directly on the embedding vector representations. Therefore, the input, as well as the output, to the neural network is a two-dimensional vector of word embeddings. The numbers in word embeddings are already bounded in approximate range of [-4, 4] and therefore no normalization is performed on the data, so we can use weights from our pretrained network without any preprocessing.

We evaluate two very distinct cases by using two different kinds of noise: *random gausian* noise and *patch noise*. For the random gausian noise, we randomly select number from the gausian distribution with median of 0 and standard deviation of 1.5. This number is generated and added to each element of the vector. The patch noise is slightly different. Instead of adding noise to every element of the vector, we select only a rectangular patch of values in the vector. As the embedding vectors can contain a large number of zero values, due to the fact that the texts can have differing lengths and therefore are padded to have the same size, this selection is performed only on the non-zero subset of the values. The size of the patch is chosen randomly, but its size is limited, as to not affect all the values. This is mainly due to the fact that we are not adding noise to these values, but removing them altogether by setting them to 0. If all the values were removed, there is no chance for the autoencoder to recreate the data.

We use three different architectures of the denoising autoencoder, which are then compared to each other. The first architecture used is a neural network that uses only fully connected layers. For this architecture the input is directly connected to a hidden layer with 512 neurons. This layer also serves as a latent representation of the features. Therefore, it is directly connected to the output, using a linear activation. The second architecture is a convolutional network, which uses the same architecture in the encoder part as the neural network we are using, with some modifications. It does not use the embedding layer, as the input is already in a form of word embeddings. The input is followed by triple convolutional layers, with 512 filters of width 3, 4 and 5 respectively, each followed by maxpooling and concatenated together. This is followed by dropout with a rate of 0.5. However, after dropout, we use a fully connected layer of size 512, which serves as a latent features representation. For the decoder part, the fully connected layer is followed by one upsampling layer and one transposed convolutional layer. The transposed convolutional layer is followed by a simple convolutional layer, with linear activation, filter width of 1 and their number of 1. The output of this layer are the denoised word embeddings. Last architecture is the same as the second one, with weights in the encoder part not initialized randomly, but instead set to weights from our pretrainend network and set to be non-trainable.

Each of these networks is trained using stochastic gradient descent with Nesterov momentum and learning rate of 0.001. The number of epochs used is one, as this number is sufficient and increase in this number results in overfitting. We also use custom loss functions, more specifically the mean absolute error. In addition, we define custom mean squared error and cosine similarity. Each of these loss functions is modified to ignore the parts of the data, where padding is used. Without the use of these modified functions, all the autoencoders have learned to output same number on each position, as the padding was much more prominent than the actual word embedding numbers.

Results

When evaluating the different denoising autoencoder architectures and comparing them together, we are using the modified loss functions. We use all three of them, the mean squared error (MSE), mean absolute error (MAE) and the cosine similarity (COS). The results of the different architectures for the two defined tasks are presented in table 8.1.

Noise	**Random**			**Patch**		
Model	**MSE**	**MAE**	**COS**	**MSE**	**MAE**	**COS**
Dense	0.028	0.056	0.00	0.024	0.050	0.01
CNN	0.017	0.046	0.57	0.017	0.046	0.58
Pretrained	0.018	0.047	0.54	0.023	0.049	0.73

Table 8.1: *The table presents the results of denoising on multiple models. On the left is the case when random gaussian noise was added to each element of the vector and on the right the case where patches of the vector were removed. We can see that all the models have very similar mean squared and mean absolute errors. These errors are very small, considering the range is [-4,4]. The comparison between models can be done most precisely according the cosine similarity. In that case, the dense model fails spectacularly, with other two models having similar results in random noise and our model having much better result in patch noise. We can declare that our model is good enough, as it achieves lower error on these tasks, with the error not being so high.*

From the results in the table, we see that the MSE and MAE is fairly similar, and sufficiently small, for all models and therefore, for comparison, we use the cosine similarity. Here we see that the model with fully connected layers fails spectacularly on both cases. The other two models perform well on both cases, with our pretrained model achieving higher cosine similarity in case of patch noise. Therefore, we can declare that the model we use is good enough and can be used for evaluation of our proposed method.

8.4 Experiment: Comparison with Attribution Methods

In this experiment, we compare the extracted attribution values with results from other attribution methods. We compare with two different methods, the Layer-wise Relevance

Propagation (LRP) [8, 6] and Local Interpretable Model-agnostic Explanations (LIME) [35]. The LRP method we use is slightly modified. This version uses the ε rule for the backpropagation of relevance from output to the input. This rule guarantees better numerical stability of the LRP method, by using a modified chain-rule which can deal with connections with zero weight [2]. For the LIME method, we use its text explainer. We also do not use the Bag-of-Words representation in the LIME method, but the one that takes the position of words into consideration.

For comparison, we first calculate the relevance scores for multiple observations by passing them to the individual methods for explanations. In the case of 20 news group data set, we use a comparison with only the LRP method for the whole data set. In the case of IMDB data set, we compare with LRP and LIME both, but only on a subset of 500 observation from test set, due to the computational complexity of the LIME method. The comparison with LRP is also performed on the whole IMDB set. As the LIME method returns the relevance scores ordered by their size, an ordering must be performed. Luckily the returned attributions specify the indices of the words in the text and therefore we can use them for ordering. Our method and the LRP return the relevance score ordered according to their occurrence and therefore no ordering is performed.

The calculated scores are then normalized. Each of the used attribution methods return the relevance scores with a slightly different meaning. Most notably, the LRP method does not adhere to any predefined interval, as the relevance scores are only relative to each other. Therefore we normalize the relevance scores to range [-1, 1] for each method, for better comparison. To guarantee that no sign in the relevance scores is changed, we performed this normalization on positive and negative relevance scores independently.

The comparison is performed using numerical, but also statistical methods. When comparing the results numerically, we calculate a mean of sum of the squares of differences between values corresponding for the same word in each method. When comparing the results statistically, we first perform the Shapiro-Wilk normality test, to determine if the values are from normal distribution. After that, we use a paired t-test, or its nonparametric equivalent the Wilcox sign test, to determine if the outputs of the different attribution methods differ. We use the 0.95 confidence level with the alternative hypothesis being non-equivalence.

For the second part, we determine the correctness of the explanations by removing the most important words and observing the change in output. For each observation we remove the words that have their relevance score higher than the 90 percentile and calculate a percentual change in the output probability. What this means is, if the output probability before is 50% and after it is 25%, the change would be 50% and not 25. The same is performed for the words that are below the 10 percentile, while comparing to the inverse probability. We also check whether the removal of words has the exact effect as we expect it to have. This due to

the fact that in some cases the removal of important words caused the probability to increase, instead of decrease and when we do not take this into consideration, the percentual changes are wildly incorrect. The final equations have the following form:

$$positive = \begin{cases} \frac{probability - new\_probability}{probability}, & \text{if } probability \geq new\_probability \\ -\frac{new\_probability - probability}{1 - probability}, & \text{otherwise} \end{cases}$$

$$negative = \begin{cases} \frac{new\_probability - probability}{1 - probability}, & \text{if } probability \leq new\_probability \\ -\frac{probability - new\_probability}{probability}, & \text{otherwise} \end{cases}$$

From these two values, we calculate a mean and compare it between different attribution methods, with higher mean indicating better explanations. To examine the behaviour of different attribution methods, we use additional percentiles, instead of just the 10 and 90 ones. This part is performed only on the IMDB data set, as the comparison of all of the methods is neccessary.

Results

The calculated numerical differences between different attribution methods and for different data sets are presented in table 8.2. In addition, the p-values for the Wilcox sign test, as all the

Data set	**IMDB Part**			**IMDB Full**	**20 news**
Method	**LIME**	**LRP**	**Method**	**Method**	**Method**
LIME	-	0.090	0.107	-	-
LRP	-	-	0.062	0.063	0.191

Table 8.2: *The resulting mean of sum of squares for different data sets and methods. We calculate the results for three different data sets: a subset of IMDB movie reviews data set, the whole IMDB data set and 20 news group data set. If the value in any cell is not present, it either was not calculated, as is case with LIME on full IMDB and 20 news, or is already present in other cells. We can see, that on IMDB data set, the results are fairly similar, with our method being closer to LRP. On the other hand, when using more complex data set, the results are more different.*

results fail the Shapiro test, are presented in table 8.3. We can see that when using the simpler data set, the IMDB movie reviews, the difference in relevance scores is small, while still being statistically significant on the 0.95 confidence level. We can also see that our proposed method is closer to the LRP method than to LIME method. The difference of the relevance scores between our method and LRP on subset and the whole IMDB data set is fairly minimal. The results show that on average, when the typical observation has around 200 words, the relevance scores shift completely from maximal value to minimal value in around 6 words between our method and LRP and around 10 words in comparison with LIME. On the other

hand, when using more complex data set, the difference is more obvious, with the change in words being around 20 words.

Data set	**IMDB Part**			**IMDB Full**	**20 news**
Method	**LIME**	**LRP**	**Method**	**Method**	**Method**
LIME	-	3.54e-10	2.33e-95	-	-
LRP	-	-	2.41e-107	0.0	5.09e-12

Table 8.3: *The resulting p-values from the Wilcox sign test for different methods and data sets. We use the 0.95 confidence level with the alternative hypothesis being non-equivalence. We calculate the p-values for three different data sets: a subset of IMDB movie reviews data set, the whole IMDB data set and 20 news group data set. If the value in any cell is not present, it either was not calculated, as is the case with LIME on full IMDB and 20 news, or is already present in other cells. We can see that all the different methods produce different relevance scores.*

The results of the second part are presented in figure 8.2. We use different percentiles values, ranging from 1 to 25. In lower values of the percentile, we see that our method performs similarly to the LIME method, with LRP being far better. However, when increasing the percentile value, the change produced using our method rises rapidly, with it overtaking the other two around the percentile value of 6, while the differences between LRP and LIME are becoming smaller. This shows us that our method can not easily identify the most important words, but can identify more important features that can be hidden by the interactions in the data.

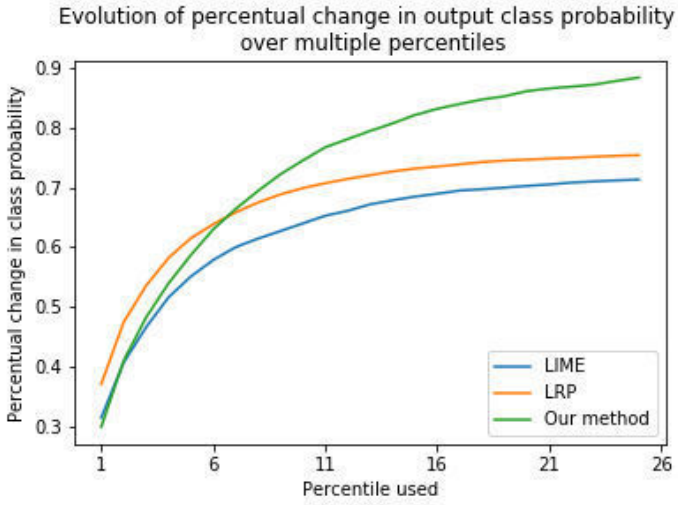


Figure 8.2: *The evolution of change in output probability for the main class for different methods over different percentiles. We can see that when it comes to small percentiles, our method behaves similarly to the LIME method. However, by increasing the percentile, the result increases more rapidly than with other methods.*

8.5 Experiment: User Experiment

In this experiment we create a golden standard against which we will compare different attribution methods, along with our proposed method. Additionally, we evaluate if our generated explanations are understandable and give insight into the decision process of our neural network. This experiment is implemented in a form of web application[4].

The evaluation is performed only on the subset of the IMDB movie reviews data set. As there are 25000 texts in the test data set, it would not be feasible to ask users to go through all of them. In addition, having the text observations marked only by one user can introduce bias into our data. For this reason, we randomly select a subset from the test data. This way, we selected 10 sets, each with 3 observations. We also select 2 additional sets of 5 text observations each that our used for the second part of the user experiment, one labeled good and the other bad.

In the first part of this experiment, we ask users to select word with important sentiment. To guarantee that we have some data for each of the sets we generated, we rotate the set that is presented each time anyone starts the experiment. In this part, the 3 text observations are presented one at a time. As we are dealing with a sentiment detection problem, we ask the users to select words with positive sentiment, but also with negative sentiment. The choosing of words is realized simply by clicking them.

For the second part, we have two trained models. One of these models is the one we are using for all the experiments. The second model is intentionally trained incorrectly. For the architecture, we drop the dropout layers and add another 2 convolutional layers, with filters widths of 2 and 5, followed by maxpools, that are added to the concatenations. In addition, the size of filters is increased to 1024. It is trained only on a subset of 1000 training samples, with validation set of size 750, which are selected to be shorter than 75 words. It is trained for 200 epochs, to guarantee it is perfectly trained on the data, and achieves 72% accuracy on the test set. The training process of this model is presented in figure 8.3.

In this part we examine if the explanations generated by our method are meaningful. Users are first presented with 5 text observation from either the good set of texts or the bad set of texts, along with the correct class for the observation and the decision made by the network. We ask the users to judge their trust in the model on a scale of 1 to 5, with 1 indicating no trust in the model and 5 indicating the absolute trust. In addition, they are asked to explain their answer. After answering these questions, explanations are presented for the text observations and the same questions are asked again, with inquiry in the second question to explain the change in the answer. This is performed for each of the 2 sets.

We use the answers to compare our proposed method to other attribution methods. From

[4] `http://experiment-perturbation.herokuapp.com/`

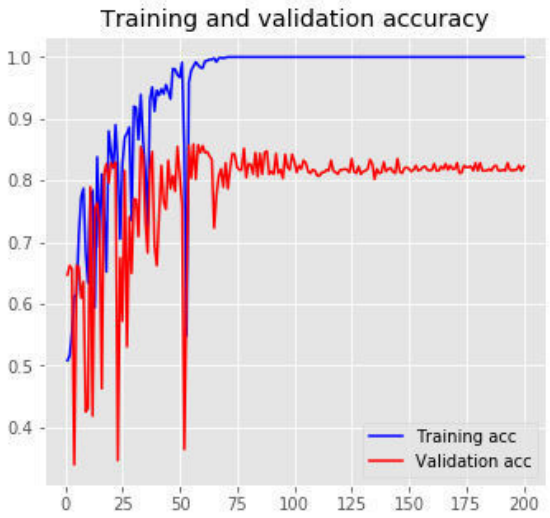


Figure 8.3: *The training process of the incorrect neural network model we use.*

the answers from the first part, we extract the indices for words chosen by each user. We also extract the most important words for positive and also negative sentiment by selecting the words that have their relevance score higher than 95 percentile, or lower than 5 percentile. We calculate a percentual overlap between the indices from users and the indices generated from the individual attribution methods. We calculate the percentual overlap using following equation:

$$overlap = \frac{|A \cap B|}{min(|A|, |B|)}$$

The A and B are the sets of words indices. Using a mean of this overlap, we can determine which attribution method generates explanations that most correspond to the expected behaviour on the data set. As our proposed method has multiple parameters we can tweak, we first use the overlaps to determine the best setup for our network and then compare the results from this setup with other attribution methods. As judging just by comparing numerical values can be misleading, we also perform statistical comparison. We first perform a Shapiro-Wilk normality test on the produced overlaps. Afterwards, we perform either a paired t-test, or its non-parametric version, the Wilcox sign test, between the different attribution methods. We use the 0.95 confidence level with the alternative hypothesis being non-equivalence. If the null hypothesis is dismissed, we use the sign of the statistic for determining if the results of the first method passed to the test are greater, indicated by plus sign, or lesser, indicated by a minus sign.

Using the results from the second part, we evaluate if our proposed method generates insightful explanations. We first process the answers given by users and modify their level of trust accordingly. For example, if the explanation why the answer was chosen is not present or points to the fact that it was in fact chosen randomly, we can choose to remove these answers

from user altogether. However, these are all the changes we perform. After that, we calculate an average level of trust in the good, and the bad, model before and after being presented with explanations. We also count the number of times the level of trust changed in either direction. We expect the level of trust before being presented with explanation to be similar in both models. However, after being presented with explanations, we expect the trust in the bad model to change at least by one degree, with the trust in good model staying approximately the same. We also expect that the number of increases in bad model should be insignificant, with the number of decrease fairly large. In the case of good model, we expect minimal number of changes in either direction.

Results

The total number of participants in this user experiment is 23.

The results for determination of the best configuration are presented in figure 8.4. We can see that the best setup for order is the simple order with step of 2, e.g. removing 2 previous and next words, with the percentual order being surprisingly atrocious. For the cosine similarity, the best threshold is 65%. We use this setup to compare with other attribution methods.

The evolution of overlap, and jaccard index, over different values of percentile is presented in figure 8.5. We can see that with higher number of words selected from different methods, the overlap with participants rises in our method. This tells us that the other two attribution methods are good for identifying few most important features. On the other hand, our method assigns the relevance scores more equally and consistently across the different important features. This is mainly due to the fact that in domains with correlated data, the importance is higher for multiple features, however it is usually hidden.

The results of percentual overlap, and jaccard index, for few selected values of the percentile are presented in table 8.4. The p-values from statistical comparisons for these selected percentile values are presented in table 8.5. The statistical comparison is performed using the Wilcox sign test, as the produced results fail the test for normality. This further confirms our findings in previous step that when dealing with few words, our method is worse than the other two attribution methods, but with increasing number of features used, its accuracy rises more rapidly. This is mostly evident in the evolution with jaccard index, where with increasing size of the features used, the score rises for our method but falls for the other attribution methods. The statistical tests also confirms this. In lower percentiles, the tests points to the fact that the results of other methods are better. At middle percentiles the results are considered the same and in higher numbers the tests swing in favour of our method.

As for the second part, the average trust before and after the explanations for the good and for the bad model is presented in table 8.6. The counts of changes are presented in table 8.7. The results, more or less, correspond to our expectations, with the trust before

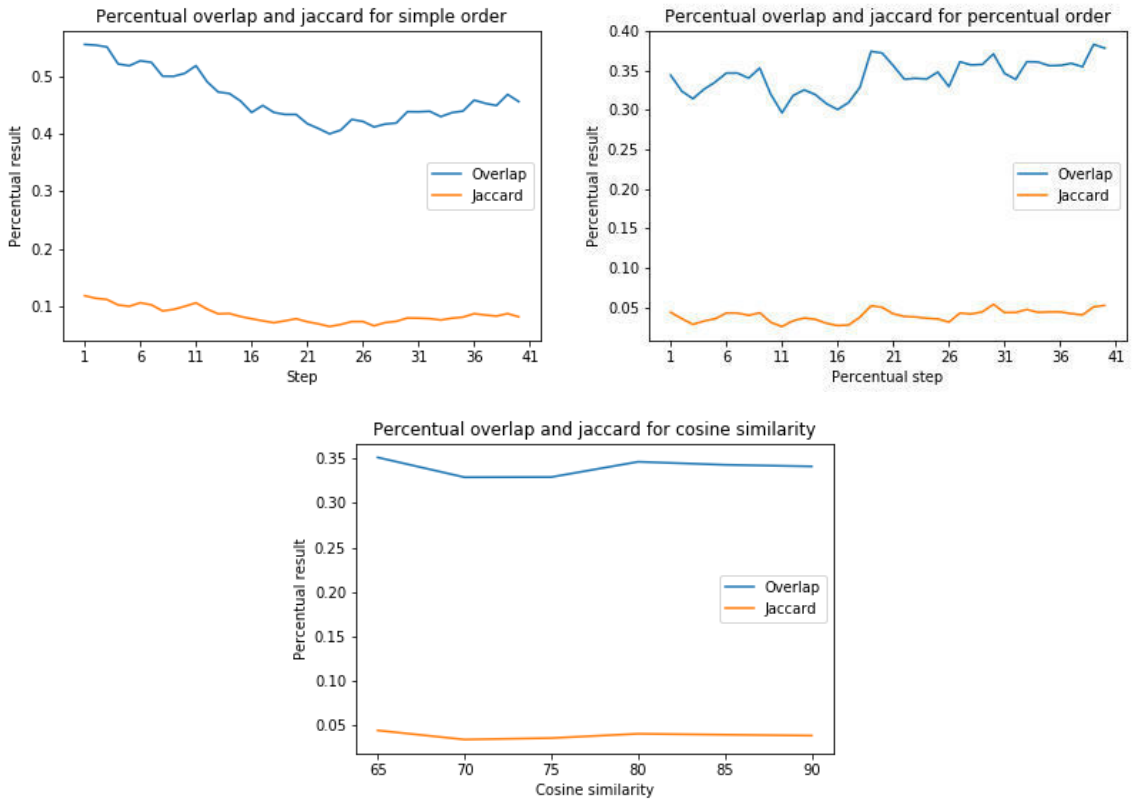


Figure 8.4: *In the picture on the left, we present the evolution of comparison results with participant answer when changing the simple order step. We change this step from 1 to 40 and the best result is obtained around step 2. On the right, the same evolution, but for percentual step is shown. We change this percentual step from 1% to 40% of the words and the maximum is obtained around 20%. The picture on bottom shows the evolution for different cosine similarities, from similarities ranging from 65% threshold to 90%, with best score at 65% threshold.*

explanations being similar and then dropping for the bad model after explanations. The trust in the good model also drops slightly after explanations, which points to some problems in the visualisation of the results, or in the explanation generation. From the text answers, a lot of participants identified that some of the highlighted words do not have any sentimental meaning, which points to the fact that the artefacts generated by perturbing multiple words at once are not dealt with as efficiently as they could be. The changes in count point to the fact that the explanations are somewhat helpful for determining the bad model, however they are not very suitable for reinforcement of trust in the model.

8.6 Results Summary and Discussion

The results from the denoising experiment point to the fact that we used sufficiently good model for evaluation of our proposed method. As the mean squared error and mean absolute error metrics were same, they were not really informative, only in determining that the error

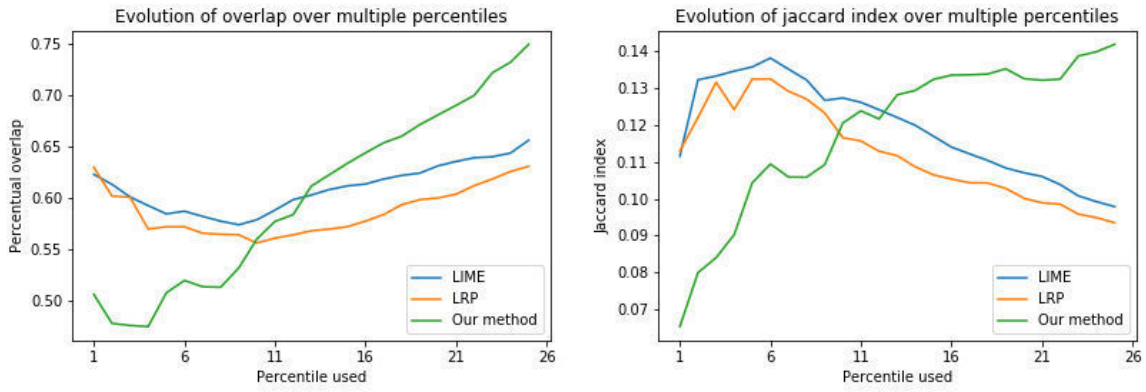


Figure 8.5: *The visualisation of comparison of different methods. We change the value of percentile of features used and compare the results for each method. On the left, we can see the evolution of percentual overlap and on the right, the evolution of jaccard index. We can see that our method performs poorly when the percentiles are lower, but increases considerably in higher percentiles. This, however, is not due to the size of the set of features, as also the jaccard index increases considerably.*

Method	*Overlap*					*Jaccard*				
Percentile	**5**	**10**	**15**	**20**	**25**	**5**	**10**	**15**	**20**	**25**
LIME	0.585	0.579	0.612	0.632	0.656	0.136	0.127	0.117	0.107	0.098
LRP	0.572	0.556	0.572	0.600	0.631	0.132	0.117	0.106	0.100	0.093
Our Method	0.508	0.560	0.634	0.681	0.750	0.104	0.121	0.132	0.133	0.142

Table 8.4: *Evolution of percentual overlap and jaccard index for different selected values of percentile. We can see the trend in our method, that with increasing the number of features we take into consideration, the results are better. This, however, is not due to having bigger size of features from our method, as also the jaccard index rises, while it drops for other methods.*

of the model is sufficiently small. For comparison we opted for the cosine similarity instead. As the pretrained network achieved similar score to fully trained CNN in the case of random noise, and better score in the case of random patch noise, which is a more general case, we can conclude that the latent features learned by this network are meaningful. Therefore, we use this network in further experiments.

When comparing our proposed method with other attribution methods, using a difference in the relevance scores and the change of output probability in case where the important words are removed, gives us a preliminary indication of how our method behaves. We can see that all of the three different methods we are using are behaving differently, with one part indicating that the LRP is closer to our method and the other that the LIME is closer. However, the difference in the outputs is still statistically significantly different. We can conclude that the difference is more evident in the more complex data set. In the case of removal of the words, we can see that our method performs poorly when only a small number of the most important features is selected, but improves rapidly when increasing this number.

Percentile	**5**		**10**		**15**		**20**		**25**	
Method	*p-value*	*s*	*p-value*	*s*	*p-value*	*s*	*p-value*	*s*	*p-value*	*s*
LIME-Method	0.0214	+	0.6388	+	0.3057	-	0.0448	-	0.0003	-
LRP-Method	0.0353	+	0.7245	-	0.0088	-	0.0014	-	0.0000	-

Table 8.5: *Evolution of p-values of Wilcox sign test for different selected values of percentile. The null hypothesis is set to be that the results are same, alternative to non-equivalence, with 0.95 confidence level. If that is not the case, the better results are determined by the sign of the statistic, presented in the column next to the p-value, with '+' indicating that alternative for greater passes and '-' that the alternative for less passes. We can see that our method has worse results in percentiles lower than 10, but better results in percentiles higher than 15, when compared with LIME, and higher than 10 when compared to LRP.*

	before	**after**
good	3.74	3.57
bad	3.67	2.87

Table 8.6: *The trust of the participants in the model, from scale of 1 to 5, with 5 indicating absolute trust and 1 no trust. Before being presented with explanations, the trust is similar for both model with users trusting the model, but with some reservations. After the explanations are shown, the trust in bad model drops to the level where users do not trust the model, only in some cases, while trust in good model stays approximately the same.*

There are two possibilities for this. We can either conclude that the relevance scores assigned by our method are more balanced across the important words, which would indicate that it can identify the interactions better. The other possibility is that, as the results are not normalized using size of removed set, the increase can be caused by more sizeable feature sets. This other possibility is not very likely, as the use of percentiles should handle it. This is also confirmed by other experiments we performed.

When finding the best setup for our method, using the data from user experiment, we have determined interesting properties for our network. We found out that the method is fairly robust to the number of removed features. Instead of just factoring in number of removed features, it puts emphasis on important words. This is evident from the fact that the percentual order configuration is a worse choice than the simple order one. In addition, it is evident from the fact that the higher number of words that are removed this way does not increase the accuracy, but reduces it instead.

In the case of comparison of different attribution methods with our proposed one, using the user marked data, we have determined that our method is better at finding the important features that are overlooked due to correlations. We determined this from the results where, when using small number of features for comparison, our method behaved considerably worse than other method. However, when increasing this number, it rapidly starts to behave

	positive	neutral	negative
good	5	10	8
bad	1	9	13

Table 8.7: *The counts of change in trust of participants after being presented with explanations. We can see that the changes in good model are similar between each other. The positive changes for bad model are almost zero. Therefore we can conclude that the explanations provide some insight.*

considerably better. This is also the case when using the jaccard index for comparison. We can therefore conclude that the increase is not simply from the fact that the number of words is higher, as the jaccard index would decrease in this case, as is the case with other methods. This shows that our proposed method is not good for finding the absolutely most important words, but on the other hand it assigns the relevance scores to important correlated features more evenly. Someone can ask if this is not just due to the artefacts produced by overestimation or underestimation of some of the features due to the removal of multiple words. However, if this was the case, the unimportant words would be chosen more often in later percentiles and therefore the increase in accuracy would not be present in the results.

The data from the experiment were also used to determine if the explanations presented by us are understandable and provide any insight. The results, more or less, correspond to our expectations. Both the good model and the bad model have the same level of trust before explanations. After being presented with the explanations, the trust of participants dropped in the case of bad model, while it stayed approximately the same in case of good model. This tells us that the explanations provide an insight into the decision process. On the other hand, the results show that the explanations do not reinforce trust in the model. Although the level of trust in good model does not drop very much, the explanations caused the level of trust to decrease in slightly more cases than they caused an increase and remained neutral in approximately the same number of cases. A large number of participants mentioned that some of the important words determined by the method were not in fact important, but had no sentiment value instead, which in turn decreased their trust. This can be a result of a presence of artfacts that were still not properly dealt with, as most of these words occurred around sentimentally important words.

Chapter 9

Conclusion

The goal of this thesis is to create a perturbation-based attribution method for explaining individual decision of neural network that can take into consideration the inherent interactions in data. In addition, we aim to produce and present the explanations in a way that is understandable for humans and gives them insight into the decision process of the model. The method is designed in regards of textual data that uses the word embedding vector representation. For the evaluation of the performance of our proposed method, we design multiple experiments to examine the properties of our network.

Using the results from our experiments, we can conclude that our method is better at finding important features that are overlooked due to the correlation than other attribution methods. When using only small number of features, our method behaves considerably worse than other methods. However, with increasing the number of features used, it rapidly overtakes the other attribution methods. This shows that our method does not prefer only a small number of features with high relevance scores, but assigns the scores to correlated features more evenly instead. It is not due to the higher number of features used, or the presence of artefacts where the importance of unimportant features is overestimated, as this increase is evident even in case where the size of features used is taken more strictly into consideration.

We can also conclude that the way we present our results is understandable for humans and gives them some insight, although some problems are still present. When given a task where the humans indirectly determine if the model is good or bad, they are able to find the bad and good model consistently when presented with explanations from our method. However, the explanations do not reinforce trust in the models. On multiple occasion, a problem with sentimentally neutral words being selected as sentimental is detected. In all cases, the problem is caused by the presence of the already mentioned artefacts from removal of multiple words, as these words are in close proximity to words with strong sentiment. Therefore, there is still room for improvement in the handling of such artefacts in the method

and the presentation of the results.

Overall, we can judge our proposed method to be a success, but there is still a room for improvement. There are still some artefacts present, from the removal of multiple words, that can cause the unimportant words being selected as important. To deal with this, a more complex normalization and a slight change in process can be used. We can take, for example, a weighted average between relevance scores from method with interactions included and the one that do not include the interactions. In addition, the behaviour of the method was examined only on the convolutional neural networks, but in the case of textual data it would be interesting to see how it performs on recurrent networks that are far better at handling of sequences.

Resumé

Modely strojového učenia sa používajú v mnohých disciplínach. Avšak, na to aby sa dali efektívne používať, potrebujeme nejaký spôsob ako vysvetľovať rozhodnutia takýchto modelov. Ak to nedokážeme, tak ľudia nebudú dôverovať našim modelom a teda ich nebudú používať. Ďalšou možnosťou je využiť vysvetľovanie na to, aby sme si overili, že model sa naozaj správa tak ako predpokladáme, a že sme nespravili chybu v trénovacom procese, čo je veľmi typickým problémom [12]. Najtypickejším modelom, ktorý potrebuje vysvetlenia, sú neurónové siete, ktoré sú vďaka pokrokom v hlbokom učení veľmi úspešné, ale na druhú stranu veľmi ťažké na vysvetlenie [17]. Typický prístup vysvetľovania rozhodnutí je použitie atribučných metód, ktoré pridelia skóre relevancie každému atribútu. Hlavným cieľom našej práce je vytvoriť metódu založenú na narušení vstupu, ktorá vie do úvahy zobrať interakcie.

Neurónové siete

Neurónové siete patria do skupiny modelov optimalizovaných gradientom [17, 4]. Základom neurónových sietí je Rosenblattov perceptrón, ktorý rozširuje model od McCullocha a Pittsa. Percentrón obsahuje viacero vstupov, každý s vlastnou váhou, pričom jedným zo vstupov je aj vychýlenie (*bias*). Na vytvorenie výstupu sú všetky vstupy prenásobené ich váhami, sčítané a prehnané cez aktivačnú funkciu [17, 4].

Samostatný perceptrón môže byť použitý ako jednoduchý lineárny klasifikátor. Keďže však dokáže vyriešiť lineárne separovateľné problémy, používa sa väčšie množstvo perceptrónov, nazývaných aj neuróny, v jednej vrstve. Takisto sa používa väčšie množstvo vrstiev, či už iba s necyklickými, ale aj cyklickými prepojeniami. Dnes sa používajú hlboké modely s použitím veľkého množstva vrstiev, pričom ukladajú naučené informácie vo váhach, vďaka čomu vedia reprezentovať vysoko nelineárne funkcie [17, 4].

Typickým modelom sú jednoduché dopredné siete, volané aj viacvrstový perceptrón. Informácia v takomto modeli je šírená vždy iba jedným smerom. Tieto modely používajú plne prepojené vrstvy, v ktorých je každý neurón prepojený so všetkými neurónmi z predošlej a nasledujúcej vrstvy. Pri trénovaní sa používa gradientová optimalizácia, za použitia chybových funkcií, ktorá mení hodnoty váh. Zmena hodnôt váh prebieha za pomoci spätnej propagácie

chyby z výslednej vrstvy smerom na vrstvu vstupnú [17, 4].

Ďalším typickým modelom sú konvolučné neurónové siete. Tieto sa používajú hlavne na dáta s komplexnejšou štruktúrou, ako sú obrázky. Tieto modely sa veľmi nelíšia od jedoduchej doprednej siete. Základným rozdielom je použitie iných typov vrstiev, ktoré nie sú prepojené so všetkými neurónmi, ale iba s malým regiónom, čím sa znižuje množstvo potrebných parametrov. Typickými vrstvami sú konvolučné, ktoré pomocou posuvného okna rátajú konvolúcie nad rôznymi neurónmi. Na zníženie množstva parametrov používajú zdielanie parametrov. Bežne sa tiež používa zhlukovacia (pooling) vrstva, ktorá zmenšuje počet neurónov tým, že zoberie iba jednu hodnotu z väčšieho množstva, a taktiež plne prepojená vrstva.

Interpretovateľnosť neurónových sietí

Interpretovateľnosť je dôležitá na vybudovanie dôvery v systém a overenie si, že sa správa tak ako predpokladáme. Toto je dôležité hlavne v oblastiach, kde malá chyba môže mať katastrofálne dôsledky [12]. Ďalšia potreba pre vysvetľovanie modelov vychádza aj z pohľadu zákona, keďže iba nedávno bol prijatý zákon v Európskej únii, ktorý okrem iného priniesol aj právo vysvetliť rozhodnutie ľuďom, ktorých toto rozhodnutie ovplyvnilo [18].

Problémom pri interpretovateľnosti je tiež to, že slová *interpretácia* a *vysvetlenie*, ktoré znamenajú rozličné veci, sú chápané, že znamenajú to isté a teda nie je jasne pomenované o čo sa v práci ľudia pokúšajú [15]. V našej práci pod týmito pojmami myslíme **generovanie vizuálnych reprezentácii relevancie komponentov modelov pre ich výstup, ktoré sú pochopiteľné pre ľudí**.

Neurónové siete sú vnímané ako čierne skrinky, ktoré nie je možné vysvetliť. Hlavným dôvodom je to, že dokážu robiť vysoko nelineárne transformácie nad dátami, pričom naučené znalosti ukladajú vo forme váh naprieč veľkým množstvom vrstiev. Na vysvetľovanie neurónových sietí existujú 4 hlavné prístupy [15]:

- Prístupy **nezávislé od modelu**, ktoré zjednodušujú model a vytvárajú vysvetlenia rozhodnutí pomocou zástupného modelu.
- Prístupy na **automatickú extrakciu pravidiel**, ktoré vytvárajú jednoduché pravidlá z rozhodovacieho procesu
- **Vizualizačné** prístupy, ktoré zobrazujú proces, alebo naučené znalosti, vo forme vizualizácií.
- **Atribučné** prístupy, ktoré vysvetľujú na základe dôležitosti vstupných atribútov pre daný výstup.

Prístupy nezávislé od modelu

Tieto modely trénujú jednoduchšie, viac vysvetliteľné modely, na základe vstupno-výstupného mapovania zložejšieho modelu. Následne vysvetlenia tohoto náhradného modelu použijú na vysvetlenie modelu komplexného.

Typickým reprezentantom je prístup od Ribeira [35], lokálne interpretovateľné, od modelu nezávislé vysvetlenia (*LIME - Local Interpretable Model-agnostic Explanations*). V tomto prístupe prezentujú malú časť najdôležitejších atribútov ako vysvetlenia, pričom berú tie s pozitívnym, ale aj tie s negatívnym dôsledkom. V tomto prístupe robia lokálne aproximácie komplexného modelu tak, že v okolí daného pozorovania vytvoria nové pozorovania, ktoré preváhujú na základe vzdialenostnej metriky. Tieto pozorovania vysvetľujú správanie modelu v okolí daného pozorovania. Tie sa následne použijú na natrénovanie jednoduchého lineárneho modelu. Je ukázané, že tento prístup funguje aj na vysvetlenie rozhodnutí komplexných neurónových sietí, ako je napríklad Google Inception model [35].

Automatická extrakcia pravidiel

Extrahované pravidlá z komplexného modelu sú väčšinou vo forme **IF-THEN**, **M-of-N**, alebo vo forme rozhodovacích stromov. Existujú 3 rôzne prístupy: dekompozičné, ktoré rozdeľujú neurónové siete na individuálne neuróny, alebo vrstvy a z nich priamo extrahujú pravidlá; pedagogické, kde sa na základe vstupno-výstupného mapovania vytvárajú pravidlá; alebo eklektické, ktoré sú kombináciou predchádzajúcich [19, 7].

Na použitie na hlboké neurónové siete je prístup s názvom *DeepRED* [52] najvhodnejší. V tomto prístupe sa vytvára strom, ktorý čo najpresnejšie mapuje rozhodovací proces siete. Aj napriek tomu, že sa používajú rôzne prístupy na zmenšenie veľkosti, takto vzniknutý strom je veľmi veľký, kvôli čomu nie je o nič viac vysvetlitelný.

Vizualizácia

Pri vizualizácii existujú 2 možné prístupy, ktoré sa dajú zvoliť. Buď sa vizualizujú zaujímavé časti použité pri predikcii, alebo aktivácie a trénovací proces siete.

Pre vizualizáciu zaujímavých častí sa používa gradientová metóda aktivácií tried, *Grad-CAM* [41]. Táto metóda využíva informáciu o gradiente na poslednej konvolučnej vrstve na porozumenie dôležitosti atribútov a ich následnú vizualizáciu.

Pri vizualizácii trénovacieho procesu sa používa metóda opísaná v [34]. Tu sa hodnoty jednotlivých neurónov transformujú do 2-dimenzionálnej podoby pomocou multidimenzionálneho škálovania (*MDS*) a vizualizuje sa ich poloha pred a po tréningu. Taktiež sa vizualizuje posun týchto neurónov v priestore a ich aktivácia.

Atribučné metódy

Atribučné metódy fungujú takým spôsobom, že každému vstupnému atribútu z daného pozorovania pridelia skóre, ktoré predstavuje dôležitosť daného atribútu pre konkrétne rozhodnutie. Výhodou takéhoto prístupu je to, že výsledky je veľmi jednoduché vizualizovať [2]. Existujú 2 základné typy atribučných metód: gradientové a prístupy založené na narušení.

Gradientové prístupy

Gradientové prístupy využívajú gradient na aproximáciu atribúcií v jednom prechode. Ich výhodou je, že sú veľmi rýchle, avšak sú závislé od architektúry.

Prvou, a teda aj najjednoduchšou gradientovou metódou je analýza citlivosti. Táto metóda využíva jednoduché pravidlo spätného reťazenia parciálnych derivácií. Problémom tohoto prístupu je však to, že je veľmi ovplyvnený derivačným šumom a dokáže rozpoznávať iba atribúty, ktoré majú pozitívnu dôležitosť [29]. Ďalšou metódou sú integrované gradienty (*Integrated Gradients*) [44], ktoré vznikli na základe preddefinovaných axióm. V tomto prístupe sa najprv vytvorí určitý baseline, ktorý predstavuje absenciu signálu. Následne sa na ceste od toho baselinu ku konkrétnemu vstupu definujú body, v ktorých je rátaný gradient, a ktoré sa na konci integrujú, alebo iba jednoducho zosumujú. Poslednou metódou je šírenie relevancie naprieč vrstvami (*Layer-wise Relevance Propagation - LRP*) [8, 6]. V tomto prístupe sa na výstupnej vrstve definuje určité skóre, ktoré sa následne pomocou váh medzi neurónmi distribuuje na nižšie vrstvy. Problémom je, že tento prístup si vie poradiť iba s *ReLU* a *tanh* aktiváciami a v niektorých prípadoch môže byť numericky nestabilný.

Prístupy založené na narušení

Pri týchto prístupoch sa pozoruje to ako sa mení výstup v prípade, keď je vstup narušený. Tento prístup je nezávislý od architektúry, počíta dôležitosti priamo a teda je presnejší, pričom však musí platiť predpoklad, že vstupné atribúty sú nezávislé, inak by narušenie nemalo zmysel [47, 2].

Základnou myšlienkou je teda zmena vstupu, či už priradenie priemernej hodnoty, alebo prechod všetkými možnými hodnotami, a následné pozorovanie ako sa mení výstup. Rátanie zmeny vo výstupe sa dá mnohými vzorcami, ako je obyčajný rozdiel, gradient, poprípade aj normalizovaný rozdiel a iné. Tieto zmeny môžu prebiehať buď **lokálne** a teda menením iba hodnoty jedného atribútu, kde poznáme prístupy ako *vynechaj jeden kovariant (Leave-one-covariate-out - LOCO)* [28]. Druhou možnosťou sú globálne prístupy, kde sa menia viaceré atribúty, či už s použitím celej dátovej sady, alebo iba časti [23, 9].

Existujú aj prístupy priamo vytvorené pre konvolučné siete nad obrázkami. Či už ide o prístup, kde sa odstraňujú časti obrázka za použitia šedého štvoruholníka a sleduje sa pokles

predpovedanej triedy, ktorý sa následne vizualizuje vo forme heatmapy [49]. V druhom prístupe, [53], pristupujú rovnako, odstraňujú pixely v okolí, tým, že ich položia rovnými nule, pričom výsledok prirátajú ku každému odstránenému pixelu a výsledné hodnoty znormalizujú. Tiež používajú prístup, kde namiesto odstránenia použijú vzorkovanie na vybratie hodnoty pre daný pixel z hodnôt pixelov na rovnakých pozíciach v iných obrázkoch [53].

Ciele práce a hypotézy

Cieľom tejto práce je vyvinúť atribučnú metódu založenú na narušovaní vstupu, ktorá vie do úvahy zobrať interakcie v dátach. Taktiež sa zaoberáme prezentovaním výsledkov tejto metódy v podobe pochopiteľnej pre ľudí. Tieto ciele sa dajú sumarizovať v týchto 2 hypotézach:

1. Zahrnutie interakcií, ktoré sa nachádzajú v dátach, do atribučnej metódy založenej na narušovaní vie pomôcť k vytvoreniu presnejších atribúcií.

2. Skóre relevancie, a ich vizualizácia, vytvorené použitím modifikovanej metódy založenej na narušení použitej na textové dáta dokáže poskytnúť náhľad do rozhodovacieho procesu naučeného modelu, čím dopomôže ľuďom lepšie pochopiť jeho správanie.

Popis navrhnutej metódy

Naša navrhnutá metóda by mala vedieť zobrať do úvahy interakcie medzi dátami počas vytvárania rozhodnutí. Funguje na princípe atribučných metód založených na narušení vstupu, pričom jej výstupom je skóre dôležitosti pre každý vstupný atribút.

Pri vytváraní našej metódy sa zameriavame na textové dáta, keďže sú typickým príkladom dát s interakciami. Napríklad slová ‘new‘ a ‘york‘ znamenajú niečo iné keď sú použité spolu a oddelene. Textové dáta reprezentujeme vektorovo vo forme vnorených slov (*word embeddings*). Toto nám zaručuje, že sa zachová poradie slov, a interakcia medzi slovami v tejto forme, a taktiež, že nepríde k rapídnemu spomaleniu metódy. O zmenu do tohoto formátu sa však stará až prvá vrstva neurónovej siete, aby sme boli nezávislí od architektúry, pričom slová repezentujeme ich indexmi vrámci slovníku slov. Taktiež, keďže texty môžu mať rôznu dĺžku, si definujeme pevnú maximálnu veľkosť pre texty, pričom všetky kratšie sa vyplnia nulovými slovami.

Vrámci našej metódy sa zameriavame na problém klasifikácie a rozpoznania sentimentu, ktorý sa dá považovať za binárnu klasifikáciu. Naša metóda sa skladá z 5 krokov: *vytvorenie baselinu*, *identifikovanie interakcií*, *narušenie vstupu*, *výpočet finálnej dôležitosti* a *odprezentovanie výsledkov*.

Vytvorenie baselinu

Na vytvorenie baselinu sa jednoducho preženie pozorovanie cez neurónovú sieť. Takto dostaneme výstupné pravdepodobnosti pre každú triedu, pričom keďže používame softmax na výslednej vrstve, tak nás zaujíma iba najpravdepodobnejšia trieda.

Identifikovanie interakcií

Pri identifikovaní interakcií nad textovými dátami vieme zvoliť jeden zo 4 prístupov. Ako prvé vieme využiť charakter dát a teda zobrať iba určitý počet predchádzajúcich a nasledujúcich slov. Podľa toho či zvolíme statickú velkosť, alebo veľkosť závislú od počtu slov v texte, ide buď o prístup s **poradím**, alebo **percentuálnym poradím**. V oboch prípadoch sa identifikujú slová iba symetricky.

Ďalší prístup je vybrať tie slová, ktoré sú si viac podobné ako nejaká hranica, za použitia kosínovej podobnosti nad vektorovou reprezentáciou.

Poslednou možnosťou je spojenie prístupu s poradím spolu s prístupom s podobnosťou, pričom tento prístup by mal byť najefektívnejší, lebo rieši problémy s precenením, poprípade podcenením, niektorých slov.

Narušenie vstupu

Narušenie sa vykonáva postupne pre každý atribút. Ako funkciu narušenia v našom prípade používame jednoduché nahradenie slova špeciálnym nulovým slovom repzentovaným nulovým vektorom.

Výpočet finálnej dôležitosti

Pri kalkulácií výslednej dôležitosti daného atribútu sa upravené pozorovanie preženie neurónovou sieťou a porovná oproti výsledku z baselinu. Funkcia, ktorou počítame výsledný rozdiel, berie do úvahy aj počty odstraňovaných atribútov a na základe toho vykonáva normalizáciu.

V našom prípade používame jednoduchý rozdiel, ktorý je predelený počtom slov, ktoré boli odstránené. Aby sme vyriešili problémy s precenením niektorých slov, výsledok tohoto rozdielu je prirátaný ku každému slovu, ktoré bolo odstránené. Na konci je každé slovo ešte predelené počtom narušení, v ktorých sa dané slovo nachádzalo.

Prezentovanie výsledkov

Na prezentovanie výsledkov bez vizualizácie netreba vykonať žiadnu ďalšiu akciu, iba namapovať späť indexy slov na slová v textovej reprezentácii a vrátiť ich spolu s ich dôležitosťou.

Taktiež vieme vrátiť iba tie najdôležitejšie slová tak, že ich usporiadame s použitím absolútnej hodnoty a vrátime iba prvých k.

Keď chceme prezentovať výsledky vo forme vizualizácie tak sa najprv jednotlivé dôležitosti naškálujú do intervalu [-1, 1]. Tieto hodnoty sa následne odčítajú od príslušných zložiek RGB farby tak, aby zelená farba vyjadrovala slová podporujúce rozhodnutie a červená slová, ktoré odporujú danému rozhodnutiu.

Overenie metódy

Atribučné metódy je ťažké overovať empiricky, keďže je ťažké oddeliť chyby modelu od chýb metódy [2, 44]. Pri overovaní našej metódy sa zameriavame na 3 hlavné veci: či použitý model je správny; či vygenerované vysvetlenia dôverne popisujú správanie modelu; a či dávajú človeku pridanú hodnotu. Na takéto overenie využívame 3 experimenty: odšumovanie, porovnanie s inými atribučnými metódami a využitie dát označkovaných ľuďmi.

Počas overenia používame 2 dátové sady, *20 news group*, ktorá je komplexnejšia, keďže ide o klasifikáciu do 20 tried a filmové recenzie od IMDB, kde sa jedná o určenie pozitívneho a negatívneho sentimentu. Obe tieto sady sú predspracované, pričom na vektorovú reprezentáciu používame predtrénované *GloVe* vnorenia. Nad oboma sadami sme natrénovali neurónovú sieť s rovnakou architektúrou, pričom nad prvou dosahuje úspešnosť 75% a na druhej 89%. Používame základné nastavenie metódy, s percentuálnym poradím o veľkosti 5% a kosínovou podobnosťou cez 70%.

Odšumovanie

Odšumovaním sa snažíme overiť, či sa nami použitá neurónová sieť naučila dobré reprezentácie. Používame 3 rôzne architektúry autoenkóderov na odšumovanie. Jedna je zložená iba z plne prepojených vrstiev. Ďalšie 2 sú konvolučné siete s rovnakou architektúrou, pričom pri jednej z nich používame ako kódovač našu predtrénovanú sieť.

Používame 2 typy šumu a porovnávame schopnosť jednotlivých sietí si s ním poradiť. Jeden, pri ktorom náhodne pričítavame šum z normálneho rozdelenia. Druhý, pri ktorom odstraňujeme hodnoty nachádzajúce sa v náhodne vybratom štvoruholníku. Na porovnanie používame strednú kvadratickú chybu, strednú absolútnu chybu a kosínovú podobnosť.

V oboch prípadoch a pri všetkých sieťach nám vyšla podobne malá kvadratická aj absolútna chyba, v hodnote 0,02 a 0,05. Kosínová podobnosť pre plne prepojenú architektúru vyšla 0, pre konvolučnú sieť 0,57 v oboch prípadoch a pre sieť s predtrénovanou časťou 0,54 na náhodnom šume a 0,73 na štvorcovom šume. Z toho vidíme, že neurónová sieť, ktorú používame, sa správa dostatočne dobre a teda ju môžeme použiť aj v ďalších experimentoch.

Porovnanie s inými atribučnými metódami

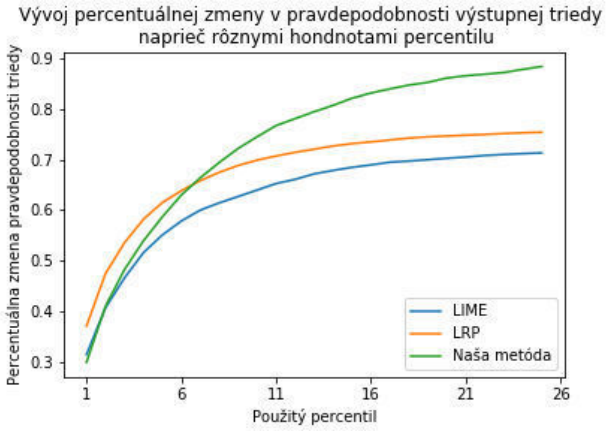
Porovnávame sa voči 2 metódam, LRP, kde používame verziu rozšírenú o ε pravidlo, ktoré zaručuje lepšiu numerickú stabilitu, a LIME, kde používame verziu s reprezentáciou, ktorá udržuje poradie slov.

Metódy najprv porovnávame numericky a štatisticky, aby sme si overili, že neprodukujú totožné vysvetlenia. Kvôli výpočtovej zložitosti metódy LIME sa s touto metódou porovnávame iba na podčasti o veľkosti 500 z IMDB dátovej sady. Keďže aj všetky metódy vracajú dôležitosti s rozličným významom, ich výsledky normalizujeme na interval [-1, 1].

Pri numerickom overení cez sumu štvorcov nám vyšlo, že existujú nejaké rozdiely medzi jednotlivými metódami, aj keď na IMDB dátovej sade nie sú až také značné, ako pri 20 news group dátovej sade. Pri overovaní cez Wilcoxonov znamienkový test nám vo všetkých prípadoch vyšlo, že so štatistickou signifikanciou na hladine významnosti 0,95 vieme zamietnuť, že výsledky metód sú totožné.

Keďže nemáme žiadny zlatý štandard voči ktorému sa môžeme porovnať a tak usúdiť, ktorá metóda je lepšia, rozhodli sme sa použiť prístup odstraňovania slov a pozorovania ako sa mení výsledok modelu. Dôležité slová vyberáme použitím percentilu, ktorého hodnoty meníme od 1 po 25. Výsledný rozdiel rátame percentuálne oproti pôvodnému výsledku. Toto spravíme aj pre slová čo potláčajú rozhodnutie a zoberieme priemer takýchto výsledkov.

Výsledky z takéhoto porovnania sú na obrázku 9.1. Môžeme vidieť, že naša metóda sa so zvyšujúcou hodnotou percentilu zlepšuje výraznejšie ako tie zvyšné, čo značí, že vie lepšie nachádzať korelované slová.



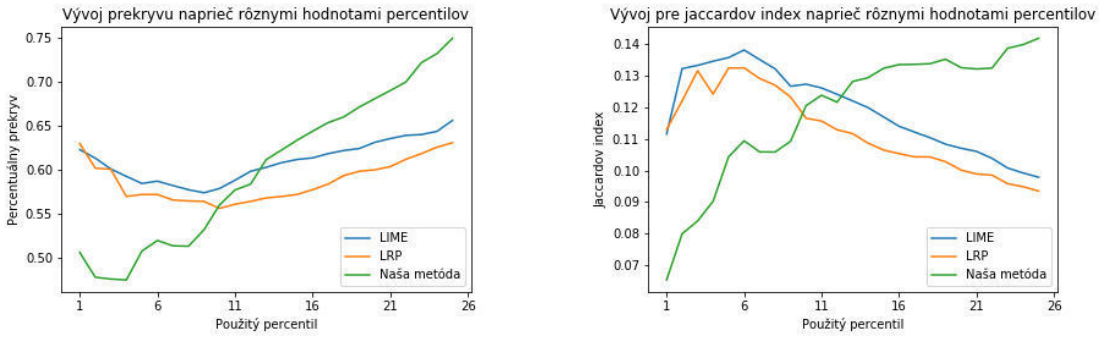
Obr. 9.1: *Vývoj zmeny v pravdepodobnosti hlavnej triedy naprieč rôznymi hodnotami percentilu pre rôzne atribučné metódy. Môžeme vidieť, že pri malých hodnotách percentilu sa naša metóda správa podobne ako metóda LIME. Avšak, keď zvyšujeme hodnotu percentilu, výsledok našej metódy stúpa rapídne rýchlejšie oproti ostatným metódam.*

Využitie dát označkovaných používateľmi

Tento experiment slúži hlavne na vytvorenie zlatého štandardu, voči ktorému sa vieme porovnávať, a na overenie, či je naše prezentovanie výsledkov pochopiteľné a či dáva pridanú hodnotu ľuďom. Experiment má 2 hlavné časti. V tomto experimente bolo 23 účastníkov.

V prvej časti sú používateľovi prezentované 3 texty z jednej z 10 možných sád textov z IMDB dátovej sady. Úlohou je vybrať tie slová, ktoré majú pozitívny a negatívny sentiment.

Výsledky z tejto časti používame na porovnanie medzi našou metódou a metódami LRP a LIME. Z jednotlivých metód vyberieme najdôležitejšie slová použitím percentilu, pričom jeho hodnotu meníme od 1 do 25. Takto dostaneme množiny slov, ktoré medzi sebou porovnáme použitím percentuálneho prekryvu a jaccardovho indexu. Porovnanie prebieha použitím strednej hodnoty z hodnôt týchto metrík. Výsledky z tohoto testu prezentujeme na obrázku 9.2.



Obr. 9.2: *Vizualizácia porovnania rôznych atribučných metód, pri ktorom meníme hodnotu percentilu počtu použitých čŕt a porovnávame výsledky so zlatým štandardom od používateľov. Naľavo môžeme vidieť vývoj percentuálneho prekryvu a napravo vývoj pre hodnoty jaccardovho indexu. Môžeme vidieť, že naša metóda sa správa zle pri nízkych hodnotách percentilu, avšak pri zvýšení jeho hodnoty sa značne zlepšuje. To nie je iba dôsledok väčšieho počtu použitých čŕt našou metódou, keďže rastie aj hodnota pre jaccardov index.*

V druhej časti tohoto experimentu sú participantom prezentované postupne 2 sady po 5 textov, pričom okrem textu je prezentovaná aj predpovedaná trieda a správna trieda. Úlohou je určiť svoju dôveru v daný model na stupnici od 1 do 5, kde 1 je najmenšia a 5 najväčšia dôvera, a taktiež odpovedať na otázku ohľadom dôvodu tejto dôvery. Následne sa zobrazia vysvetlenia pre jednotlivé rozhodnutia, pričom pri jednej textovej sade sú vysvetlenia zo zlého modelu, a znova sa odpovedá na rovnaké otázky.

Výsledky z tejto časti sa spracujú a porovnajú s očakávanými výsledkami. Očakáva sa, že dôvera v oba modely pred vysvetleniami by mala byť podobná, avšak po zobrazení vysvetlení by mala rapídne klesnúť pre zlý model, pričom pri dobrom modeli by mala ostať približne rovnaká. Tiež sa pozeráme na počet zmien dôvery na každú stranu. Výsledky z tejto časti viac menej korešpondujú s očakávaniami, kde dôvera pred vysvetleniami je na úrovni 3,74 pre

dobrý a 3,67 pre zlý model, pričom po vysvetleniach to je 3,57 a 2,87. Pri zlom modeli je taktiež počet zvýšení dôvery iba 1 pričom zníženie a žiadna zmena sú na približne rovnakej úrovni. Pri dobrom modeli sú jednotlivé zmeny všetky na približne rovnakej úrovni.

Diskusia výsledkov

Z výsledkov odšumovania vidíme, že náš model, je dostatočne dobrý na použitie aj pri iných experimentoch. Z experimentov s porovnaním vidíme, že naša metóda nedokáže dobre identifikovať tie najlepšie slová, avšak dôležitosť rozdeľuje viac rovnomerne medzi slová s interakciami a teda pri použití väčšieho počtu slov funguje lepšie ako ostatné. Nie je to dôsledkom iba toho, že by percentil vybral viac slov, keďže toto isté je možné pozorovať aj pri použití jaccardovho indexu, ktorý tvrdo trestá veľkosti množín slov.

Z dát z experimentu s ľuďmi tiež vieme povedať, že naše vysvetlenia sú pochopiteľné a dávajú pridanú hodnotu. S použitím vysvetlení je možné odhaliť zlý model. Avšak, aj keď dôvera v dobrý model neklesla až tak výrazne, počet jej znížení je vyšší ako počet zvýšení. Taktiež veľké množstvo účastníkov spomenulo, že vo výsledkoch sa nachádzali označené niektoré slová bez sentimentu, čo zapríčinilo pokles ich dôvery. Vo všetkých prípadoch sa tieto slová nachádzali v blízkosti slov so silným sentimentom, čo značí, že riešenie artefaktov z preceňovania slov nie je ešte vyriešené tak dobre ako by malo.

Záver

Cieľom tejto práce je vytvoriť atribučnú metódu založenú na narušení vstupu, ktorá vie pri svojej činnosti zobrať do úvahy interakcie v dátach, pričom výsledky tejto metódy chceme prezentovať v pochopiteľnej podobe s pridanou hodnotou pre ľudí.

Z výsledkov z experimentov môžeme usúdiť, že naša navrhnutá metóda vie lepšie nájsť dôležité atribúty, ktoré môžu byť prehliadnuté kvôli interakciám v dátach. Toto je evidentné z toho, že pri použití malého počtu atribútov sa správa horšie ako zvyšné metódy, ale pri vyšších počtoch atribútov ich veľmi rýchlo predbehne. Taktiež vieme usúdiť, že vysvetlenia sú pochopiteľné a dávajú pridanú hodnotu, aj keď sú s nimi ešte menšie problémy. Použitím vysvetlení je možné odhaliť zle sa správajúci model, avšak v mnohých prípadoch sa stáva, že aj nedôležité slová sú označené za dôležité.

Celkovo vieme zhodnotiť, že sme splnili naše ciele, aj keď ešte existuje priestor na zlepšenie v oblasti vysporiadania sa s artefaktami, kde je preceňovaná dôležitosť slov. Tento problém by sa dal vyriešiť použitím váhovania medzi metódami bez a s interakciami. Správanie metódy tiež bolo overené iba nad konvolučnými sieťami a bolo by zaujímavé pozrieť sa na správanie pri rekurentných sieťach, ktoré sa lepšie vedia vysporiadať s textom.

Bibliography

[1] Jamal Alsakran, Ali Rodan, Nouh Alhindawi, and Hossam Faris. Visualization analysis of feed forward neural network input contribution. *Scientific research and essays*, 9(14):645–651, 2014.

[2] Marco Ancona, Enea Ceolini, Cengiz Oztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *6th International Conference on Learning Representations (ICLR 2018)*, 2018.

[3] Fei-Fei Li Andrej Karpathy and Justin Johnson. Convolutional neural networks (cnns / convnets), 2017.

[4] Fei-Fei Li Andrej Karpathy and Justin Johnson. Neural networks, 2017.

[5] Ehsan Ardjmand, David F Millie, Iman Ghalehkhondabi, William A Young II, and Gary R Weckman. A state-based sensitivity analysis for distinguishing the global importance of predictor variables in artificial neural networks. *Advances in Artificial Neural Systems*, 2016, 2016.

[6] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. " what is relevant in a text document?": An interpretable machine learning approach. *PloS one*, 12(8):e0181142, 2017.

[7] M Gethsiyal Augasta and T Kathirvalavakumar. Rule extraction from neural networks—a comparative study. In *Pattern Recognition, Informatics and Medical Engineering (PRIME), 2012 International Conference on*, pages 404–408. IEEE, 2012.

[8] 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):e0130140, 2015.

[9] Paulo Cortez and Mark J Embrechts. Using sensitivity analysis and visualization techniques to open black box data mining models. *Information Sciences*, 225:1–17, 2013.

[10] Mark W Craven and Jude W Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning Proceedings 1994*, pages 37–45. Elsevier, 1994.

[11] Mark W Craven and Jude W Shavlik. *Extracting comprehensible models from trained neural networks*. PhD thesis, University of Wisconsin, Madison, 1996.

[12] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, page arXiv:1702.08608, Feb 2017.

[13] LiMin Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(8):1114–1124, 1994.

[14] G. David Garson. Interpreting neural-network connection weights. *AI Expert*, 6(4):46–51, April 1991.

[15] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An approach to evaluating interpretability of machine learning. *arXiv preprint arXiv:1806.00069*, 2018.

[16] ATC Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995.

[17] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[18] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a" right to explanation". *arXiv preprint arXiv:1606.08813*, 2016.

[19] Tameru Hailesilassie. Rule extraction algorithm for deep neural networks: A review. *arXiv preprint arXiv:1610.05267*, 2016.

[20] Kai Han, Yunhe Wang, Chao Zhang, Chao Li, and Chao Xu. Autoencoder inspired unsupervised feature selection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2941–2945. IEEE, 2018.

[21] S. Hashem. Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 419–424 vol.1, Jun 1992.

[22] Eduardo R Hruschka and Nelson FF Ebecken. Extracting rules from multilayer perceptrons in classification problems: A clustering-based approach. *Neurocomputing*, 70(1-3):384–397, 2006.

[23] Bertrand Iooss and Paul Lemaître. A review on global sensitivity analysis methods. In *Uncertainty management in simulation-optimization of complex systems*, pages 101–122. Springer, 2015.

[24] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] Daniel T Larose and Chantal D Larose. *Discovering knowledge in data: an introduction to data mining*. John Wiley & Sons, 2014.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, (just-accepted), 2017.

[29] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 2017.

[30] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[31] Julian D Olden and Donald A Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1-2):135–150, 2002.

[32] Stacy L Özesmi and Uygar Özesmi. An artificial neural network approach to spatial habitat modelling with interspecific interaction. *Ecological modelling*, 116(1):15–31, 1999.

[33] Hean-Lee Poh, Jingtao Yao, and Teo Jašic. Neural networks for the analysis and forecasting of advertising and promotion impact. *International Journal of Intelligent Systems in Accounting, Finance & Management*, 7(4):253–268, 1998.

[34] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):101–110, 2017.

[35] 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. ACM, 2016.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[37] Andrea Saltelli and Paola Annoni. How to avoid a perfunctory sensitivity analysis. *Environmental Modelling & Software*, 25(12):1508–1517, 2010.

[38] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, and Marco Ratto. *Sensitivity analysis in practice: a guide to assessing scientific models*. John Wiley & Sons, 2004.

[39] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 28(11):2660–2673, 2017.

[40] Warren S. Sarle. How to measure importance of inputs?, 2000.

[41] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *See https://arxiv. org/abs/1610.02391 v3*, 7(8), 2016.

[42] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[44] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[46] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems*, pages 341–349, 2012.

[47] JT Yao. Sensitivity analysis for data mining. In *Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American*, pages 272–277. IEEE, 2003.

[48] Daniel S. Yeung, Ian Cloete, Daming Shi, and Wing W.Y. Ng. *Sensitivity Analysis for Neural Networks*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[49] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[50] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2921–2929. IEEE, 2016.

[51] Caroline Ziemkiewicz and Robert Kosara. Preconceptions and individual differences in understanding visual metaphors. In *Computer Graphics Forum*, volume 28, pages 911–918. Wiley Online Library, 2009.

[52] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. Deepred–rule extraction from deep neural networks. In *International Conference on Discovery Science*, pages 457–473. Springer, 2016.

[53] Luisa M. Zintgraf, Taco Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *5th International Conference on Learning Representations (ICLR 2017)*, 2017.

Appendix A

Technical Documentation

For the implementation of the proposed method, and all the experiments performed, we use a *python* programming language, version 3.6.8.

In order to be able to setup the development environment easily, for ease of use and for our results to be reproducible, we use multiple different docker images and containers as our environments. The *Dockerfiles* are also included in code directories. If the use of docker images and containers is not possible, all the required packages that are needed for setting up of the development environment are defined using a pip virtual environment using *Pipfile* and *Pipfile.lock* files in the code directories. If neither of those is available, the requirements are provided in the following section.

Requirements

For training and using the neural network, we use a *Keras* library (version 2.2.4) with the *Tensorflow* backend (version 1.13.1). For preprocessing of data, we use a *NLTK* library (version 3.4) and *contractions* library (version 0.0.17). For effective handling of data, we use *Numpy* (version 1.16.2) and *Pandas* (version 0.24.1). We also use the *scikit-learn* library (version 0.20.3). For visualisation purposes, we use *matplotlib* (version 3.0.3), *seaborn* (version 0.9.0) and *imgkit* (version 1.0.1) with *Pillow* (version 5.4.1). These last two libarries require a *wkhtlmtopdf* installed.

For the comparison with other attribution methods experiments, the libraries containing these methods are required. We use the *DeepExplain* library[1] for LRP method and the *LIME* library[2] (version 0.1.1.32).

The basis of the user experiment is realized as a web application and therefore requires very different libraries. We use a *Flask* library (version 1.0.2) with extensions of *Flask-wtf*

[1] `https://github.com/marcoancona/DeepExplain`
[2] `https://github.com/marcotcr/lime`

(version 0.14.2), *Flask-bootstrap* (version 3.3.7.1), *Flask-cli* (version 0.4.0), *Flask-socketio* (version 3.3.2), *Flask-script* (version 2.0.6) and *Flask-cors* (version 3.0.7). We also use the *click* library (version 7.0), *eventlet* (version 0.24.1) and *python-dotenv* (version 0.10.1). As a database, we use *postgresql10* and therefore also use the following libraries: *psycopg2* (version 2.7.7), *sqlalchemy* (version 1.3.1), *Flask-sqlalchemy* (verion 2.3.2) and *Flask-migrate* (version 2.4.0).

Specifications

In this section, we present specifications for our method and our experiments.

Proposed Method

As a development environment for our proposed method, we are using a docker image[3]. Although there are two tags for this docker image, we mostly use the one with tag *simplified*. In addition to all the requirements for the method and most of the experiments, it also contains a *jupyter notebook* library (version 1.0.0) for ease of use. To use this docker image, a volume must be mounted to access files. It should be mounted at the */host* path in the container. In addition, the jupyter notebook must be executed from the container. The ports that need to be mapped are 8888 and 6006.

The implementation of our method is realized using a class *CorrelatedPerturbation* from the *attributions* module. When initializing the class, a pretrained model that will be used and a dictionary for mapping of words in text form to index form must be passed. In addition, there is a possibility of either passing an *embedding_matrix*, representing the embedding vector for all the words as they are defined the the word index, or the *config_path* variable representing a location with configuration files. These are used internally and will be described later.

Only one main method is not used internally and therefore only it should be called, the *calculate_word_importance* method. It defines following parameters:

- Input parameters:
  - **data** - not optional, represents the observations for which we want to calculate importances. Passed as a two-dimensional numpy array with rows representing observations with words represented as numbers from the *word_index* used in initialization.
  - **correlation_type** - not optional, one of the following: *order*, *cosine*, *combined*. Determines how the correlated features set is calculated.

[3] `https://hub.docker.com/r/branop/deep_ml/`

  - **mode** - not optional, one of the following: *default*, *percentual*. Further specifies what type of order is used in case of *correlation_type* being either *combined* or *order*.
  - **return_best** - optional, defaults to *False*. Determines if all feature attributions are returned or only the best ones.
  - **num_of_words** - optional, defaults to 10, used only when *return_best* is *True*. Determines number of feature attributions returned.
  - **step** - must be present in the case of default order or combined correlation type, must be positive. Determines how many preceding/following words are removed.
  - **percentual_step** - must be present in the case or percentual order of combined correlation type and be in range [0, 1]. Determines the percentual count of removed words.
  - **threshold** - must be present in case of cosine or combined correlation type and be in range [0, 1]. Determines threshold for cosine similarity where the words are considered similar.

- Output parameters:
  - **final_importance** - attributions for all the observations in passed data.
  - **parent** - the probability of the resulting class.
  - **full** - probabilities for all the classes.

Using its input parameters, this method determines with which specifications to call other internal methods for calculation of attributions. Also checks the requirements for passed parameters.

Following is the description of internal methods used:

- **multiple_correlated_perturbation_importance** - performs all the calculations using the passed internal *setup*, *perturb* and *update* methods. Also maps the number representation of words to their text representation using the *word_index* dictionary. As large part of the calculation is same for all the approaches for determining correlation set, this function is passed functions as parameters that are used in places that differ.

- **map_importance_to_words** - performs the mapping of number representation of words into their text representation. In addition, this method handles the clipping of attributions in cases, where only limited number of top ones are required.

- **setup** methods - methods starting with *__setup_perturbation* followed by a name of perturbation used. These methods are used for creating the correlation set. They

assign values to internal variables that are then used in further computation. In case of use of similarity, this method also either calculates the correlation set using the *embedding_matrix* variable passed at initialization of class, or loads the precomputed set from a location determined by the *config_path* variable passed at initialization.

- **perturb** methods - methods starting with *__perturb* followed by a name of perturbation used. These methods are used for perturbing the copy of data.
- **update** methods - methods starting with *__update* followed by a name of perturbation used. These methods are used for calculating the relevance scores and updating the arrays used for normalization.

Visualisations

The *attributions* module also contains functions for visualising the results. There are three possibilities of how the results can be visualised. First is using the *attributions_to_colors* which takes attributions as input parameters and optional parameters which specify if we want the colour in RGB format or hexadecimal format. This method scales the attributions to range [-1, 1], while using the maximum and minimum values, and then calculates the RGB colour by subtracting the resulting attributions from the specific RGB elements. When calculating the hexadecimal representation, this RGB colour is passed to *rgb_to_hex* method, which transforms it to hexadecimal string representation of this colour.

Another possibility is to get the visualisation in form of HTML by using the *attributions_to_html* method. It takes the attributions as input and uses the *attributions_to_colors* method to calculate the hexadecimal representation for each attribution. Using these colours, it inserts each word into a *span* element, with the background colour specified as the output colours and return a string representation of a list of these elements.

Last, but not least, to get visualisation in form of image, the *attributions_as_image* method can be used. It takes the attributions as input, generates HTML representation by calling the *attributions_to_html* method and then uses the *imgkit* library to create an image. This image is also opened using the *Pillow* library and returned.

Serialization

The *serialize* module contains the functionality for serialization of objects. For saving of the architecture and weights of the trained neural network, it contains the *save_model_bpker* method, which takes the model, file name and path to the file as its parameters. For loading of trained neural network, along with its weights, it contains *load_model_bpker* method. This method takes the file name and path as input parameters. It also takes optional parameter

called *load_train*, which is used to load weights from training checkpoints, which have '.train' in their name. This method returns the loaded object

For serialization of other object, two methods that use the *pickle* library are present here. For saving, the *pickle_save* method can be used, which takes the object, file name and path to file parameters. For loading, the *pickle_load* method can be used, which takes file name and path to the file as parameters and returns the loaded object.

Other Attributions Methods

For the development environment for these attribution methods we also use the default docker image.

For the realization of implementation of the other two attribution methods we use, we define following class, *Explain*, which is part of the *attributions* module. For the initialization, it takes 2 non-optional parameters, the model we want to use and the word index, the dictionary mapping of word text representations to their number representation. In addition, it also takes an optional parameter, called *texts* that is required only in the case of *LIME* method to create a tokenizer. Using this parameter, it precomputes the tokenizer required for this method.

There are two main methods in this class:

- **LRP** - this method is used to calculate the LRP attributions. It uses two internal methods, the *calculate_attributions*, which calculates the attributions using the *DeepExplain* library, and the *map_importance_to_word* method, which maps the number representation of words in attributions to text representation. This method takes already preprocessed data as input.

- **LIME** - this method is used to calculate the LIME attributions. It uses the internal method, the *pipeline_for_text*, which uses the tokenizer and the provided model to transforms the observations in string representation into class probabilities. This method takes data in string representation as input. The attributions returned are already sorted in descending order, with the words represented by their occurence in the specified text.

User Experiment

For the development environment of the user experiment we use 2 different docker images that are composed together using a docker compose file. This docker compose file creates a docker image for the web application that contains the *flask* application and composes it with a *postgresql 10* image.

Following is the structure of the application:

- **/main.py** - the main script of the application that handles various tasks using the *Manager* from *flask_script* library. This script handles the running of the application using the *run_app* method. It also handles all the operations with database: *migrate_app* migrates the database, *deploy* upgrades the database to the latest migration, *recreate_db* removes all files from the database and recreates it and *seed_db* inserts all the data into the database, from the *data* directory.
- **/config.py** - contains all the configuration for the application.
- **/data** - directory containing all the data that we use in the application in csv format.
- **/migrations** - directory containing all the migrations in the application.
- **/app** - directory containing all the code for the application. In the application we use the *Blueprints* approach, e.g. the application is split into different modules, with each modules containing the model, controller and its corresponding views in its own separate directory. We use only one module called *experiment*.
- **/app/static** - contains all static files, the css and javascript in the application.
- **/app/templates** - contains all the HTML for different modules. It contains one base HTML file, which is then extended in the *experiment* module.
- **/app/modules/base** - contains the base model, which is then extended in other modules. This base model specifies the *ID* column, which serves as a primary key, and the *date_created* and *date_modified* columns.
- **/app/modules/experiment** - the main module used in our application. It contains the specification of model and routes used. This module is described bellow.
- **/app/modules/experiment/model.py** - specifies the schema we use in the application, with all the tables, columns and relations.

We identify two possible paths in our experiment: the main one and the subsidiary one. The main path can be initiated at the root path of the application. The subsidiary one can either be initiated at the root, or at the end of the main path of the application. We define following routes in the application:

- **index** - the root of the application. Contains basic info about the experiment and links to the other paths in the application.
- **experiment** - the main path of the application. This route handles the round robin choosing of texts that are shown. Accepts only ‘GET‘ request and renders the page containing the whole experiment.

- **identify** - the subsidiary path of the application. This route only accepts ‘GET‘ request and the optional *user_id* parameter, which serves as an identification for user continuing from the main path of the application. Renders HTML page that leads to the form where the users identify themselves.

- **selecting** - the subsidiary path of the application. This routes saves the identifier selected by the user. If the optional *user_id* parameter is passed, it first searches for the user with the given id, if not, it searches if user with the specified identifier exists and if not then creates new user. Afterwards, selects the texts that have no annotation from the selected user and renders them. Accepts only ‘GET‘ requests.

- **save_chosen** - the main path of the application. This route is used for saving the annotation of texts by users after the first part of the experiment is done. The expected parameters are *count*, which specifies how many texts were annotated. Then it parses *count* lists of word ids corresponding to positive and negative sentiment and saves them as rows in *ChosenWord* table. This routes only accepts ‘POST‘ request in AJAX form and returns a JSON response or error.

- **save_answers** - the main path of the application. This route is used for saving the answers from users after the second part of the experiment is done. The expected parameters are *answers_count*, which specifies how many answers were given. Then it parses *anwers_count* objects containing the values required for the creation of row in *Answer* table, which are then saved. This routes only accepts ‘POST‘ request in AJAX form and returns a JSON response or error.

- **save** - subsidiary path of the application. This route is used for saving the annotations made by participants in the optional part of the experiment, which closely corresponds to the first part of the experiment, except it is performed on all other texts. The expected parameters are *identification*, which is a primary key for users, *id*, which is the identification of the row from *Text* table and a list of ids of words with positive and negative sentiment. This routes only accepts ‘POST‘ request in AJAX form and returns a JSON response or error.

- **download** - this route only accepts ‘GET‘ requests. It extracts all the data uploaded by participants, saves them as a pickle object and sends this pickled object to the requester.

This application serves only for the presentation of data and for participants of the experiment, it does not generate any attributions or visualisations on its own. All the required data, for example the texts and colours for the words in those texts, are saved for it in the database.

Used Architecture of Neural Network

The architecture of the neural network we use in most of our experiments is in following figure: A.1.

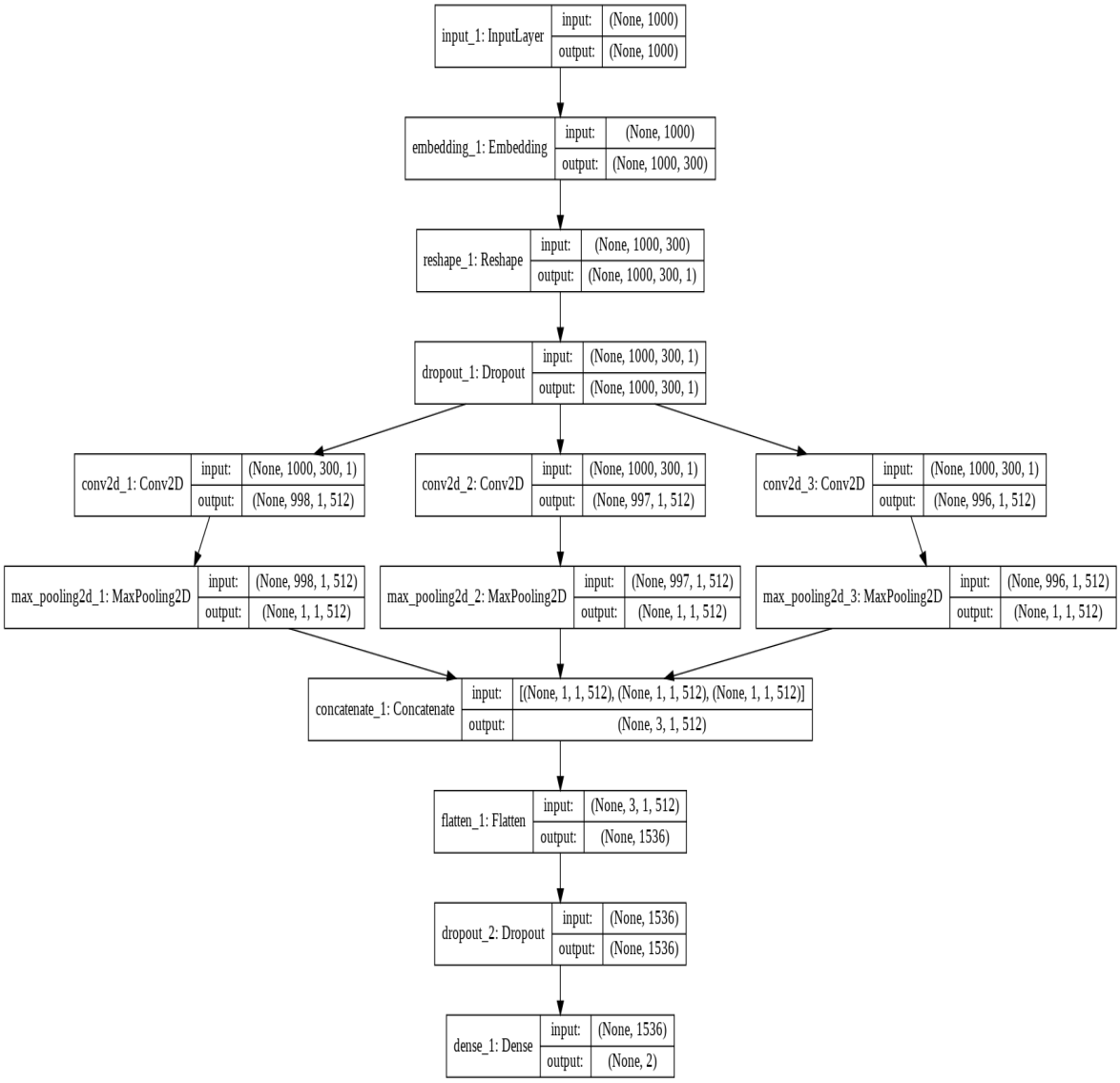


Figure A.1: *The architecture of the neural networks that we use for both data sets. Note that this architecture is taken from the neural network used for IMDB movie reviews data set and therefore the embedding layers outputs embeddings of size 300. For the 20 news group data set, this size is reduced to 100.*

Appendix B

Plan of Work and its Fulfilment

Following is the plan of work for the winter semester 2018/2019:

- **Week 1-2** - research more possibilities for determining the dependency between attributes, in addition to correlation coefficients. Research the possibilities of Kullback–Leibler divergence and the Kolmogorov–Smirnov test. Try out the methods on a simple dataset with dependent attributes to see how it performs.
- **Week 3-6** - implement a basic prototype of the proposed sensitivity analysis method. Use a simple dataset with tabular data and only a few dependent attributes. Train a simple neural network model for a classification task on this dataset outputting a probability of the given class. Use the prototype of the proposed method on this model to determine importance of different attributes. Compare the results with other sensitivity analysis methods described in 3.4.3.
- **Week 7-9** - improve the implemented prototype of the method. Use other methods for determining the interaction between attributes. Try other methods for removal of observed attribute and for calculating the difference between normal output and output after removing the attribute. Test the performance of the method on other input data, e.g. textual data or image data.
- **Week 9-12** - determine the best way of verifying the proposed method. Create a framework for verifying the proposed method by comparing the neural network model method with other methods for determining the feature importances, for example by taking importances from decision trees. Verify the method against other simple methods of sensitivity analysis but also methods from other approaches

The fulfillment of the previous plan is hard to judge. Along with the first point in the plan, a further research into the problem domain was performed and a change in the method has been made. Instead of focusing on any data in general, we have decided it would be best to

focus only on one domain. For this reason we have decided to specify our method for neural networks dealing with text represented as word embeddings. For this reason, the methods mentioned in the first point are not needed, even though they were tried out. Additionally, this research has yielded a finding, that there is a discrepancy in the naming of the methods, where the name ‘sensitivity analysis‘ has a lot of different meanings, as it is a rather old nomenclature. To be precise, what was previously meant as a ‘sensitivity analysis‘ is now a set of methods called attribution methods, where sensitivity analysis is one of the methods. For the consistency sake, I have decided to leave the previous plan as it was before, however a renaming should be done as described previously.

Another thing that was changed, was the shift of the development of the evaluation technique to a much sooner date. Instead of designing ways of evaluating our method at the end of semester, we have decided it would be best to start with it around week 4. Before that, a simple prototype of our method was implemented, so we could evaluate something. Therefore the improvement of the created prototype was left as a work for another semester.

All in all, I can say that the plan that was set for this semester was fulfilled, with a few changes to it, as was described previously.

Following is the plan of work for the spring semester 2018/2019:

- **February** - make improvements to the prototype developed in previous semester. Try out the prototype, but also the improved method on another data set, for example on the IMDB ratings data set for sentiment detection.
- **March** - evaluate the proposed method using the evaluation approaches presented in the design chapter. Prepare data for the user experiment, execute it and then evaluate the results from it.
- **April** - further refinements of the proposed method and its evaluation as needed. Perform another round of evaluations as needed.

To summarize, this semester will be about refining the proposed method and determining, if it is behaving as we expect, comparing it to other solutions and determining if it really performs better, therefore evaluating it.

All in all, the plan for this semester was fulfilled, with few other tasks added to it. The IMDB movie reviews data set was prepared before the end of February. The user experiment, and data for it, was prepared, and executed at the end of March. The results were evaluated around middle of April, as that was when we got enough participants. In March and April, further refinements were performed on the method. In April, further evaluation of the improved method was also performed. During the semester, additional tasks were performed that were not mentioned, such as preparing the submission for the IIT-SRC conference.

.

Appendix C

IIT-SRC Submission

This appendix contains the paper, that was submitted to the IIT-SRC 2019 conference. The submission was made on 27th of March 2019.

Explaining Individual Neural Network Decisions Using Correlated Perturbations

Branislav PECHER*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
branop95@gmail.com

Abstract. Deep neural networks are widely regarded as black-box models lacking transparency, which impedes their adoption in many areas. It is valuable to be certain that our model is behaving as it should. Therefore methods allowing the explanations of decisions are becoming popular. We propose a new method for explaining individual neural network decisions using a perturbation-based attribution approach, that can take into consideration the inherent interactions present in data. We evaluate this method using novel approaches designed to deal with the difficulties present in evaluation of attribution methods. The proposed method is designed and evaluated using text data set with word embedding representations.

1 Introduction

Deep neural networks are one of the most powerful machine learning models. They are rapidly growing in popularity and are improving the possibilities of use of machine learning models in many different areas like medicine or banking. Even though their promise for many tasks is high in theory, their adoption in practice is impeded due to their black-box nature.

Interpretability matters [3]. When developing a deep neural network model, the main focus is improving the accuracy of the model for a given task. However, looking only at the accuracy of the model can be misleading. As the set of used data and the training process are created by humans, it can easily happen that some problems with the data set or evaluation functions are unintentionally overlooked, or even generated [3].

Take for an example a situation where a model has learned to encode information in high frequency signal in images, due to its loss function, making it susceptible to adversarial attacks [2]. This shows, that we are still prone to overlooking problems with the model setup, which leads to improper model that tends to make errors. Such errors are unacceptable in high cost domains. Only with interpretability can the models be audited. Having an explanation for the decision process helps us understand it better and fix it, when the decision does not correspond to our expectations.

Although generating explanations is important when it comes to creation of the model, it is much more important when it is deployed in practice. When the model can explain its decision process, a domain expert can determine if it is behaving correctly and therefore build his trust in model. By enhancing the trust in the model, its use is being promoted.

In this paper we propose a new method for explaining individual decisions, that can take into consideration the inherent interactions present in the data. The explanations are in a form of importance of features for the specific decision. We are focusing on the text data sets using word embedding representations.

The paper is organized as follows. In section 2 we briefly present papers related to our approach. In section 3 we describe our proposed approach, while section 4 deals with our early experiments for this method and planned future tests for a more in depth evaluation. The section 5 concludes this paper with our findings and observations.

2 Related work

When improving the interpretability of deep neural network model by explaining its individual decisions made, there are only two possible approaches - *surrogate* or *attribution*.

In *surrogate* approaches, a surrogate model is trained. In order for this approach to be feasible, the

* Master study programme in field: Intelligent Software Systems
Supervisor: Dr. Jakub Ševcech, Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

IIT.SRC 2019, Bratislava, April 17, 2019, pp. 1–8.

surrogate model chosen is one that is much simpler and more interpretable than the neural network. One such approach is proposed by Zilke et al. [7], where they use a decompositional approach progressively on each layer to decompose a deep neural network into a decision tree. Although a pruning is performed on the generated tree, it is still quite large. This results in the explanation power of the generated tree.

Ribeiro et al. [4] propose approach called *Locally Interpretable Model-agnostic Explanations (LIME)*. Their main focus is on enhancing the trust in the model, thus making the explanations as interpretable for humans as possible. They are explaining individual decisions by exploring the neighbourhood of the observation in input space and generating weighted new observations from it. These observations are then used in a training process of a surrogate model, oft a decision tree, which is then used to explain the decision. It was shown, that it performs quite well even on deep networks. However, the generation and training process needs to be done individually for each decision, which results in a slowdown of the explanation.

The *attribution* approach assigns a score to each input feature in the model. This score, called attribution or relevance score, represents the contribution of the input feature to the decision made, which can be used as a means of explanation.

One of the attribution approaches is *Layer-wise Relevance Propagation (LRP)* proposed by Bach et al. [1]. In this approach, an arbitrary score is defined on the output layer. Then, this score is decomposed, using a gradient of the function and a Taylor decomposition, onto the lower layer using a "message sending" approach, that closely corresponds to the weighted connections in the network. Sundararajan et al. [6] proposed approach based on axioms for explanation methods. First, they define a baseline input, representing a lack of signal in the data. Afterwards, a straight-line path is defined between the baseline and a specific observation. On fixed point on this path, a gradient is computed. The generated gradients are then integrated, or summed, together and used as an attribution. These two approaches are gradient-based as they are using a gradient for approximating the attribution, which makes them architecture dependent.

Another type are the perturbation-based methods. These methods compute the attributions directly by introducing a perturbation to the input and observing how it affects the output. Robnik et al. [5] propose a basic perturbation-based approach used for numerical attributes. At each feature, they explore each possible weighted value, or interval of values, of the feature and observe how it changes the information gain from the unperturbed input. Their approach was built upon by Zintgraf et al. [8] by extending it for use on deep neural networks used for images using a sliding window instead. Although independent of architecture, these approaches experience a drastic in presence of large number of features. In addition, they cannot easily deal with interactions present in the data.

We think the perturbation based approaches have the most promise, as they are calculating the attributions directly from the behaviour of the network. Therefore we focus on reducing the problem caused by the interaction in the data and to some extent with the speed of generating an explanation.

3 Perturbation-based attribution method with interactions

We propose perturbation-based attribution method that can take into consideration the inherent interactions in the data, thus generating more accurate explanations. We generate an attribution score for each attribute, by introducing a perturbation to it, that signify the relevance of the given attribute to a given decision. By doing this for every attribute present and showing which are the most relevant ones, we can provide explanations for individual neural network decision. We identify both the positively and also negatively important features, eg. the ones that suppress the decision. As we are focusing on text data sets, where the interactions between features are apparent, we assume that the input is using a vector-based representation, more specifically word embeddings. This guarantees that we can easily determine the interaction between features and there is no drastic slowdown due to the number of features. We consider a whole vector of word embeddings to be one feature.

When generating an explanation of a decision, we follow a simple process. First we generate a baseline for comparison by passing the observation to a pretrained model. The baseline consist of 2 output probabilities, the probability of the correct class and the sum of probabilities for all other classes.

After obtaining the baseline, we calculate the attribution score for each feature one at a time. First, a set of correlated features is identified. We can choose from one of the 3 possible approaches, each with its own positives and negatives. First possibility is to use a number of previous and following words, thanks to the character of our data. This way we have consistent size of the set, however the significance of some features may be overestimated. Another possibility is to exploit the vector representation and find features that are most similar, using a thresholded cosine similarity. This produces better results in some cases, but causes further slowdown and inconsistency of size of the set. Last possibility is to use the combination of previous two approaches. This approach reduces the problems of previous ones, however can cause problems when

the set is too large.

The examined feature, along with the set of its correlated features, is perturbed. As a means of perturbation, we can either set the feature vector to zero vector, or replace it with sampling of features at the same positions from other observations. The observation perturbed this way is passed to the model, the output is recorded and compared to the baseline. For the comparison we use a weighted difference of the two recorded values.

When calculating a final attribution of a feature, we must take into account the fact, that we are perturbing multiple features at once, which can cause unwanted artifacts in the results. Consider an unimportant feature that is preceded and followed by an important word. It would get much higher relevance as it should normally. Therefore the output difference is added to the attribution of each feature in the set. Additionally, the attribution for each feature is normalized by the size of its correlated features set. This guarantees, that the effect described earlier is reduced in unimportant words, while the important ones are not affected. The final normalized attribution is used as a relevance explanation.

4 Evaluation scenarios

For the evaluation of the proposed method, we have come up with 3 experiments: *keyword comparison*, *attribution method comparison* and *user feedback*. A basic version of each one of these was already performed, with the last one planned for a more in depth evaluation.

4.1 Experimental setup

We use two different data sets. For the comparison experiments we use a 20 newsgroup data set[1], the version cleaned, ordered by date and using all 20 classes. While this data set is vital for the comparison experiments, due to its complexity, for the user experiment we needed a much simpler one. We have decided to use the IMDB moview review sentiment analysis data set[2], with 2 classes.

Both dataset are preprocessed, tokenized and transformed to word embedding representation using pretrained GloVe embeddings[3]. For 20 newsgroup we use the set with 6B tokens and vector length of 100 and for the IMDB we use the set with 840B tokens with length of 300.

The set of correlated features in our method is generated by selecting 1 previous and following word in the 20 newsgroup data set and 2 previous and following words for the IMDB one. As perturbation, we set the embedding vectors to zero vector. The difference is calculated only using the probabilities of correct class. The pretrained neural network contains 3 concatenated layers with 512 filters of width 3, 4 and 5 respectively, each using ReLU activation, followed by a maxpooling layer. Afterwards a dropout with rate of 0.5 is used, followed by a dense layer with softmax activation. The model is trained using a categorical cross-entropy loss, RMSprop optimizer and learning rate of 0.0001 and achieves 70% accuracy on 20 newsgroup and 90% accuracy on IMDB data set.

4.2 Keyword comparison

Text keywords can be considered the distinguishing factor between texts in the corpus. Therefore we assume, that these are the words that the model determines as the the most important ones in its learning process.

To confirm our assumption, we first calculate the attribution for all the features in the observation using the proposed method. The words that appear multiple times are merged together, summing their relevance. The resulting list of words is sorted by their relevance and first 10 words are extracted.

Afterwards, the set of 10 keywords is extracted using methods for keyword extraction. We use two such methods: *Rapid Automatic Keyword Extraction (RAKE)* and *TFIDF*. The RAKE method is used without any changes, only taking the top words. When calculating the TFIDF, no threshold on the occurrence of words is used. The score is calculated in two steps. First two scores are calculated: one by passing only the texts with correct class and second by passing all the other texts. In a second step, we calculate difference between these two, which is used to extract top 10 words. Having three sets of words from different methods, an overlap between each one is calculated.

4.3 Attribution method comparison

The drawback of the keywords is the fact, that only features with positive impact are identified, in addition to the possibility of discrepancy between the meaning of keywords and important words. This experiment is designed as an extension to deal with these drawbacks.

As a baseline for comparison we selected the LRP [1] method, modified for better numerical stabil-

[1] `http://qwone.com/~jason/20Newsgroups/`

[2] `http://ai.stanford.edu/~amaas/data/sentiment/`

[3] `https://nlp.stanford.edu/projects/glove/`

ity using ε rule. As the attributions from this method have a slightly different meaning, we normalized them to <-1, 1> range, before comparing them using a sum of squares (SS).

4.4 User feedback

To effectively compare between different attribution methods, a golden standard (GS) for comparison must be generated. The first part of this experiment is used to do just that. Users are presented with a movie review from the IMDB data set and are asked to select the words that represent the positive and negative sentiment for that review. Each user is presented with multiple such observations. This way we get a set of positive and negative words. The results from different methods can be compared to these sets using a simple overlap or Jaccard index. The results from multiple methods can be then compared using statistical tests.

The explanations generated using our method should also be comprehensible for humans and have some added value for them. This is what we want to evaluate using a second part of the experiment. Here, users are presented with a set of movie reviews, along with the class predicted by a given model and users are asked a few questions. The questions are aimed on determining their trust in the model. After answering the questions, explanations for each decision from before are generated and the questions are asked again. The answers are compared to our expected answers. As we are using good, but also bad models, that appear to be good, in this part, we expect that our explanations should help in identifying the bad models.

4.5 Results

Keyword comparison shows, that on average 25% of words identified by our method match those from TFIDF, 16% from RAKE and only 11% between TFIDF and RAKE. There is still a room for improvement, as no normalization for length of text was used, in addition to slight problem in keyword generation process.

Attribution comparison shows that on average less than 1 word fully flips between the two methods, corresponding to normalized SS result of 0.79. However, if a large difference occurred, there was no way to determine which method is better. For the time being we used keywords comparison to determine this, while planning to use a user feedback in the future instead. The LRP method achieved 24% match with TFIDF, 13% match with RAKE and a 46% match with our method. This shows our method performs better.

The preliminary results from *user experiment* performed on 2 participants look promising. Our method achieved a 50% match with GS, while the LRP method achieved only 45%. The second part also showed that the explanations helped, however more participants are needed for this part and the experiment as a whole.

5 Conclusion and future work

In this paper, we propose a new perturbation-based attribution method that can deal with the inherent interactions in the data, thus generating better explanations. We use this method for explaining individual decisions in a text classification problem with word embedding representation.

We performed few preliminary experiments evaluating the success of our method. The results give us an indication that our method has the potential to perform better than other attribution methods, matching 5% more words with TFIDF and also the golden standard. Although, additional experiments and tweaks to our method are needed.

In the near future, we are planning to perform additional, more in depth, experimentation, which should give us much clearer results. Most importantly, the user experiment with more participants is planned, along with an evaluation, not mentioned in this paper, based on denoising,

References

[1] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 2015, vol. 10, no. 7, p. e0130140.

[2] Chu, C., Zhmoginov, A., Sandler, M.: CycleGAN, a Master of Steganography. *CoRR*, 2017, vol. abs/1712.02950.

[3] Doshi-Velez, F., Kim, B.: Towards A Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, 2017, p. arXiv:1702.08608.

[4] Ribeiro, M.T., Singh, S., Guestrin, C.: 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*, ACM, 2016, pp. 1135–1144.

[5] Robnik-Šikonja, M., Kononenko, I.: Explaining Classifications For Individual Instances. *IEEE Transactions on Knowledge and Data Engineering*, 2008, vol. 20, no. 5, pp. 589–600.

[6] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 3319–3328.

[7] Zilke, J.R., Loza Mencía, E., Janssen, F.: DeepRED – Rule Extraction from Deep Neural Networks. In Calders, T., Ceci, M., Malerba, D., eds.: *Discovery Science*, Cham, Springer International Publishing, 2016, pp. 457–473.

[8] Zintgraf, L.M., Cohen, T., Adel, T., Welling, M.: Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. In: *5th International Conference on Learning Representations (ICLR 2017)*, 2017.

Appendix D

Electronic Medium

Registration number of the thesis in information system: FIIT-182905-72287

Following is the description of electronic part of this thesis, sumbitted with the name *DP_prilohy_digital_xpecher.zip*.

/Code/Dataset/IMDB

- Folder containing the IMDB movie reviews data set, split into train and test set.

/Code/Dataset/20news-bydate

- Folder containing the 20 news group data set, split into train and test set.

/Code/Config/imdb_config.pkl

- Binary file containing pickled loaded and preprocessed texts from IMDB data set.

/Code/Config/news_configuration.pkl

- Binary file containing pickled loaded and preprocessed texts from 20 news group data set.

/Code/Config/UserExperimnet

- Folder containing indices used for user experiment and results from this experiment.

/Code/Config/Comparison

- Folder containing indices used for the simple comparison.

/Code/Modules/attributions.py

- Python file used for calculation of different attributions. Contains the implementation of our proposed method, along with the creation of visualisation. Also provides class for generating attributions from LRP and LIME method.

/Code/Modules/serialize.py

- Python file that provides an implemented serialization methods.

/Code/UserExperiment

- Folder containing the code for the whole web application, and the data for the database, that was used for the user experiment.

/Code/Dataset_Preparation_and_Results_Visualisation.ipynb

- Jupyter notebook containing the loading and preprocessing of the different data sets we use. In addition this file contains the creation of figures for different experiments. It also compares some of the results from the first part of the user experiment, mainly performs the statistical tests.

/Code/Colab_Notebooks

- Folder containing jupyter notebooks that were extracted from the used Google Colab notebooks.

/Code/Colab_Notebooks/20news.ipynb

- Jupyter notebook containing the training of neural network used on 20 news group.

/Code/Colab_Notebooks/IMDB.ipynb

- Jupyter notebook containing the training of neural network used on IMDB movie reviews data set.

/Code/Colab_Notebooks/Denoising.ipynb

- Jupyter notebook used for training and evaluation of neural networks used in the denoising experiment. Also contains results from this experiment.

/Code/Colab_Notebooks/Attributions_and_Simple_Comparison.ipynb

- Jupyter notebook containing the generation of attributions for different experiments. Also contains the code that was used in the simple comparison between different attribution methods.

/Code/Colab_Notebooks/User_Experiment_Comparison.ipynb

- Jupyter notebook containing the code that was used to get results from the both parts of user experiment.

/Code/Model

- Folder containing the neural networks trained for different data sets. The neural network fo IMDB data set is in *IMDB* folder and for the 20 news group is in *20news* folder.

/Document/DP-III

- Master's thesis (this document)