Interpreting Sum-Product Networks via Influence Functions

Interpretierung von Sum-Product Networks mittels Influence Functions
Bachelor-Thesis von Mark Rothermel
Tag der Einreichung:

1. Gutachten: Prof. Dr. Kristian Kersting
2. Gutachten: Xiaoting Shao, M.Sc.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Machine Learning

Interpreting
Sum-Product Networks via Influence Functions
Interpretierung von Sum-Product Networks mittels Influence Functions

Vorgelegte Bachelor-Thesis von Mark Rothermel

1. Gutachten: Prof. Dr. Kristian Kersting
2. Gutachten: Xiaoting Shao, M.Sc.

Tag der Einreichung:

Abstract

Correctness and trustworthiness are crucial properties for the use of artificial intelligence (AI); incorrect AI models are worthless, and models lacking trust are not used. Comprehensibility of models is a prerequisite for trust and necessary for verification. Nevertheless, widely used models such as artificial neural networks (NNs), are semantically opaque, and hence referred to as *black-boxes*. Therefore, it is desirable to make black-box models comprehensible.

The presented thesis lays the foundation of interpreting *sum-product networks (SPNs)* via *influence functions (IFs)* by examining important basic aspects of applying IFs to SPNs. SPNs are probabilistic graphical black-box models which were introduced with the aim of compact and tractable probabilistic modeling and inference. IFs serve here for approximation of the influence of arbitrary train samples on the model parameters. Based on this, the influence on the loss of an arbitrary test sample can be approximated, too. The IF values can, in turn, be used to conclude interpretations about functionality and predictions of the considered SPN.

We will see that different kinds of model parameters may contribute to the influence in different orders of magnitude. As a consequence, influence contributions of some certain parameters may be drowned out by the contribution of other parameters. Furthermore, this work shows that the selection of an adequate loss function is crucial in terms of explanatory expressiveness of IF values. Accordingly, IF values regarding the joint likelihood (given in the root of the SPN) are largely homogeneous and many samples have influence equal to zero, whereas IF values regarding the conditional likelihood are way more heterogeneous and unequal to zero. Moreover, the effect of the Hessian matrix is investigated with the result that situations exist where the Hessian gives a useful contribution, balancing the IF values and providing more realistic IF results. Finally, we document the phenomena of influence noise in regions of generally low influence, meaning that influence of features not relevant for classification may outweigh influence of other, classification-relevant features.

Zusammenfassung

Korrektheit und Vertrauenswürdigkeit sind zwei entscheidende Eigenschaften für den Einsatz künstlicher Intelligenz (KI); inkorrekte Modelle sind nutzlos und KI-Modelle, denen nicht vertraut wird, werden nicht verwendet. Nachvollziehbarkeit der Modelle ist Voraussetzung für Vertrauen und eine Notwendigkeit für die Verifikation derer. Allerdings sind die heutzutage typischerweise eingesetzten Modelle wie künstliche neuronale Netze (NNs) semantisch sehr intransparent und werden daher auch als *Black-Box* bezeichnet. Es ist daher Ziel, Black-Box-Modelle nachvollziehbar zu machen.

Diese Thesis legt den Grundstein für die Erklärung von (Entscheidungen von) *Sum-Product Networks (SPNs)* mittels *Influence Functions (IFs)*, indem die grundlegenden Aspekte der Anwendung von IFs auf SPNs beleuchtet und untersucht werden. SPNs sind probabilistische graphische Black-Box-Modelle, die mit den Zweck der möglichst effizienten Darstellung und Berechnung probabilistischer Sachverhalte eingeführt worden sind. Die IFs werden hier eingesetzt für die Approximation des Einflusses von ausgewählten Trainingsinstanzen auf die Modellparameter. Darauf aufbauend kann auch der angenäherte Einfluss auf den Loss einer beliebig ausgewählten Testinstanz ermittelt werden. In der Folge davon können Rückschlüsse auf die Funktions- und Entscheidungsweise des SPNs gewonnen werden.

Dabei wird festgestellt, dass unterschiedliche Modellparameter verschieden stark zum Einfluss einzelner Instanzen beitragen; sogar so sehr, dass möglicherweise der Beitrag anderer Modellparameter nicht mehr wahrnehmbar ist. Darüber hinaus wird gezeigt, dass die Wahl einer adäquaten Loss-Funktion eine wichtige Rolle dahingehend spielt, wie informativ die resultierenden IF-Werte sind. Demnach sind IF-Werte bezüglich der vom SPN im Wurzelknoten ausgegebenen Verbundwahrscheinlichkeit sehr homogen, oftmals auch Null, wohingegen IF-Werte bezüglich der bedingten Wahrscheinlichkeit deutlich heterogener und ungleich Null sind. Des Weiteren wird der Effekt der Hesse-Matrix, welche Bestandteil der IFs ist, evaluiert, mit der Beobachtung, dass es durchaus Situationen gibt, in denen die Hesse-Matrix einen nützlichen Beitrag zur Balance der IF-Werte liefert. Schlussendlich wird auch das Phänomen dokumentiert, dass in Regionen geringen Einflusses ein Rauschen existiert in dem Sinne, dass Features, die eigentlich keine Rolle für die Klassifikation einer Testinstanz innehaben, einen deutlich höheren Einfluss haben können als andere, für die Klassifikation relevantere Features.

Acknowledgments

Most of all, I want to thank Xiaoting and Kristian for the supervision of this thesis and for the numerous and partly long discussions we had about my experiments.

Special thanks go to Alejandro Molina for the substantial development of `SPFlow`[1], an extensive open-source library for SPN creation and learning. This work would not have been feasible without the help of `SPFlow`. I also thank Koh et al. for making the code basis of influence functions publicly available[2], which contributed significantly to the development of this work.

Furthermore, I am very grateful for the tireless engagement of all the volunteer members of Fachschaft Informatik. Their commitment is a great help for my fellow students and me to have a bright and smooth time of studies as a computer science student at the TU Darmstadt.

I also thank the "Friedrich-Naumann-Stiftung für die Freiheit" for the scholarship, supporting me to be financially trouble-free during my studies.

Thank you!

[1] `https://github.com/SPFlow/SPFlow`

[2] https://github.com/kohpangwei/influence-release

Contents

1 Explainable AI: An Introduction

The impact of artificial intelligence (AI) on the human's daily life rose dramatically in recent years. For example, the dream of autonomous car driving is already reality. Or, AI models reached par-human performance in medical diagnosis of e.g. skin cancer [1], which is the most common human malignancy. Because often human health or life depends on the AI model's decision, avoiding mistakes in such use cases is crucial. However, not only correctness is desirable but also trustworthiness. "If the users do not trust a model or a prediction, they will not use it." [2] Hence, methods are needed to disclose the functionality of models so that users can comprehend them and, in turn, gain trust. Nevertheless, commonly used AI models, especially deep neural networks (NNs), are black-boxes, i.e. the process of decision making is practically incomprehensible to the user. This is because the semantic of NN parameters is generally unclear. But, comprehensibility may be needed for identification of security issues or weaknesses. For instance, the lack of robustness can be exploited by tiny perturbations of input data in order to alter the model's decision outcomes completely, as a study [3] shows. Another danger could be a bad training set where, e.g., the patient ID does highly correlate with the target label. [4] As a consequence, the model tends to learn the wrong input features to be relevant for a certain prediction. Incidents like this one are also called *data leakage*, causing a model easily to be right for the *wrong* reasons. Developing and using explanatory methods help us to discover such misbehavior. As a result, the discipline of *Explainable AI (XAI)* emerged, aiming at making models more understandable. The presented thesis is an attempt to contributing to XAI.

1.1 Motivation of this Work

Many different black-box architectures exist. Poon et al. [5] introduced in 2012 a deep learning architecture known as *sum-product networks (SPNs)*. The purpose of SPNs is most expressive tractable probabilistic modeling, where the models are represented by an alternating order of sum and product layers. SPNs are especially able to model complex mixtures of probability distributions where inference is generally intractable. Here, probabilistic inference via SPNs takes time linear to the SPN's size, hence, joint, marginal as well as conditional probabilities can be computed very easily. These properties make SPNs especially interesting, wherefore they were selected for this thesis.

Also SPNs suffer from the lack of understandability. This bachelor thesis is an attempt of making SPN inference more comprehensible. To this end, *influence functions (IFs)* are used. IFs are classic mathematical instruments, already used in 1980 [6] for detecting influential cases in regression. Later on, Koh et al. [7] presented a method of how IFs can be used in order to understand black-box predictions. The presented thesis is going to transfer the concepts proposed by Koh et al. onto SPNs.

1.2 Related Work: Other Explaining Approaches

Next to IFs, also other explanation techniques were developed. They can be roughly divided into model-agnostic and example-based explainers. To mention some examples: In 2016, Ribeiro et al. [2] invented *LIME (Local Interpretable Model-agnostic Explanations)*, an algorithm for generating prediction explanations for any kind of classifier or regressor "in a faithful way, by approximating it locally with an interpretable model." [2] ML models may also be explained with the help of *input gradients*. [8] The input gradient of a sample z is a vector normal to the model's decision boundary at z, which "serves as a first-order description of the model's behavior" near z. [8] *Partial Dependence Plots (PDPs)* show the relationship between the outcome of an ML model and one or two input features. [9] The relationship can

be a linear, monotonic or more complex dependency. For more XAI techniques and detailed discussions see [10, 11].

To the best of my knowledge, up-to-date, no work was published where attempts were made to interpret SPNs via IFs.

1.3 Outline

After a brief notation definition in Section 1.4, the concept of SPNs will be introduced in Chapter 2, where also several exemplary applications are shown. Chapter 3 follows, introducing the definition of IFs. Then, based on the fundamentals from Chapters 2 and 3, IFs are going to be applied to SPNs in Chapter 4. Some ideas serving as possible future work are summarized in Chapter 5. The thesis concludes the most important observations and inferences in Chapter 6.

1.4 General Definitions and Notation

In order to use symbols consistently throughout the thesis, we make some central definitions at first.

Definition 1.4.1. We define the following symbols:

(i) The set X denotes the set of all *unlabeled samples*.

(ii) The set Y denotes the set of all *labels* or *classes*.

(iii) The set $Z := X \times Y$ denotes the set of all *labeled samples*.

(iv) The set $Z_{\text{train}} := \{(x, y) \in X \times Y : x \text{ is train sample with label } y\}$ denotes a set of selected *train samples* or *observations*. Moreover, we specify $n_{\text{train}} := |Z_{\text{train}}|$ as the number of all train samples.

(v) The set $Z_{\text{test}} := \{(x, y) \in X \times Y : x \text{ is test sample with label } y\}$ denotes a set of selected *test samples*. Moreover, we specify $n_{\text{test}} := |Z_{\text{test}}|$ as the number of all test samples.

2 Sum-Product Networks (SPNs): A Deep and Tractable Architecture for Probabilistic Modeling

Real-world situations often can be modeled by probability distributions. For example, if a patient smokes cigarettes frequently but is young, eats healthy food and does sports often: How likely will he become lung cancer in the next ten years? Or, there is a big city with millions of people, mainly young, but living space is rare. How many babies will be born in 2020 most likely? Since many, many factors play a role, modeling can become arbitrarily complex. These can aggregate to probabilistic distributions which can be represented by graphical models such as Markov random fields or Bayesian networks. Nevertheless, inference in such models is intractable in general. Other classes of graphical models, such as mixture models, where inference is tractable, exist. However, these models are quite limited in the distributions they can represent compactly. [5]

Therefore, Poon et al. [5] introduced in 2012 the *sum-product network (SPN)*. It is a graphical model with alternating sum and product layers. In particular, SPNs were developed with the aim of learning tractable representations even of complex probability distributions.

The goal of this chapter is to give a general and informative introduction to SPNs. To this end, we are going to define the term SPN at first in Section 2.1. Then, in Section 2.2, we examine how inference with an SPN works. After that, a look on two different learning techniques follows in Section 2.3. Furthermore, we summarize the advantages of SPNs compared to other techniques in Section 2.4. Finally, we finish the SPN chapter with two exemplary classification problems and a completion problem where an SPN is instated as a classifier/regressor.

2.1 Architecture and Properties of an SPN

We begin with the central topological definition of an SPN and all its sub-components.

Definition 2.1.1. A *sum-product network (SPN)* S is defined as follows:

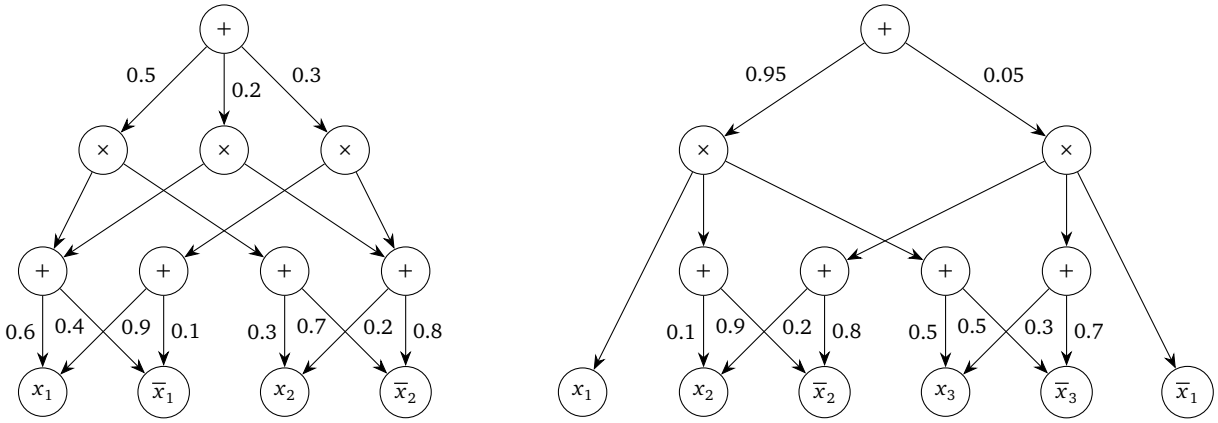
(i) S is a rooted directed acyclic graph $S := (N, E)$, where N and $E \subseteq N^2$ are the set of nodes and edges, respectively. Each node $n \in N$ is either a *sum node*, a *product node* or a *leaf node*. The sets N_{sum}, N_{prod} and N_{leaf} denote the set of sum nodes, product nodes and leaf nodes, respectively. Furthermore, we define r as the (unique) root node of S.

(ii) We call $\text{Ch}(n) := \{m \in N | (n, m) \in E\}$ the set of all children of a node $n \in N$.

(iii) Each edge $e := (n, m) \in E$ emanating from a sum node $n \in N_{\text{sum}}$ has an associated non-negative weight $w_{nm} \in \mathbb{R}^+$.

The nodes of an SPN are arranged hierarchically in layers where nodes of a layer are all of the same type (sum, product or leaf). The root node r forms the topmost layer, whereas an alternating order of layers of sum nodes and layers of product nodes follows. The lowermost layer only consists of leaf nodes.

Leaf nodes directly represent elementary distributions, e.g. continuous Gaussian distributions, Poisson distributions, or just a discrete Bernoulli distribution. Different leaf nodes may represent different distributions, talking about Mixed SPNs (MSPNs). [12]



(a) SPN over random variables X_1 and X_2.

(b) SPN over random variables X_1, X_2 and X_3.

Figure 2.1: Examples of sum-product networks, consisting of four layers: the root layer, a product layer, a sum layer and the leaf layer. The leaf layer consists of indicators, representing Bernoulli distributions. Numbers indicate the weights of the corresponding edges. (cf. [5])

Figure 2.1 depicts two exemplary SPNs. Both have four layers. The leaves (x_1, $\overline{x}_1$ etc.) are indicators, representing Bernoulli distributions over the Boolean random variables X_1, X_2 and X_3. An indicator x_i is 1 iff the corresponding variable X_i is true, 0 otherwise, and vice-versa for $\overline{x}_i$. We refer to $x := (x_1, x_2, ..., x_d, \overline{x}_1, \overline{x}_2, ..., \overline{x}_d)$ as the *state*. We call the state *complete* iff for each variable X_i it holds $x_i + \overline{x}_i = 1$.

For the sake of simplicity, we focus on discrete variables here. The extension to continuous variables is quite straightforward.

Probabilistic inference is made bottom-up, from leaves to the root. Product nodes pass the product of their children's values and sum nodes propagate a linear combination of their children's values. This gives us

Definition 2.1.2. We define the term *value* as follows:

(i) For each sum node $n \in N_{\text{sum}}$ its value $S_n(x)$ regarding a state x is defined as $S_n(x) := \sum_{m \in Ch(n)} w_{nm} S_m(x)$, i.e. the weighted sum of its children's values.

(ii) For each product node $n \in N_{\text{prod}}$ its value $S_n(x)$ regarding a state x is defined as $S_n(x) := \prod_{m \in Ch(n)} S_m(x)$, i.e. the product of its children's values.

(iii) The value of the SPN S regarding an input x is defined as the value $S(x) := S_r(x)$ of its root node r.

So far, we can only compute values for single states. But, we can go further and extend SPN inference to sets of states: A set E of complete states is called *evidence* if the states together represent a partial instantiation of the variable $X = (X_1, ..., X_d)$. We then define the value $S(E)$ of an SPN S under evidence E as $S(e_1, ..., e_d, \overline{e}_1, ..., \overline{e}_d)$ where for each instantiated X_i the values e_i and $\overline{e}_i$ are according the instantiation of X_i, and for all other X_i the values are set to $e_i = \overline{e}_i = 1$.

As an example, consider the variable $X = (X_1, X_2)$ where $X_1 = 1$. Then, for the corresponding evidence E it holds $E = \{(1, 1, 0, 0), (1, 0, 0, 1)\}$. Finally, we can obtain $S(E)$ with $S(E) = S(1, 1, 0, 1)$.

An SPN is worthless if its representation is invalid, representing no useful distribution. Therefore, a meaningful SPN must satisfy certain properties. In order to define these properties, we introduce the term *scope*.

Definition 2.1.3. The *scope* $\text{Sc}(n)$ of a node $n \in N$ is defined as the set of all variables appearing in the leaves of the sub-SPN rooted at n.

Based on the scope, we can define certain SPN probabilities.

Definition 2.1.4. Let S be an SPN.

(i) We call S *decomposable* when each product node $n \in N_{\text{prod}}$ has children $\text{Ch}(n)$ with disjoint scopes, i.e. for all $m, m' \in \text{Ch}(n)$ it holds $\text{Sc}(m) \cap \text{Sc}(m') = \emptyset$.

(ii) We call S *consistent* when each product node $n \in N_{\text{prod}}$ has children $\text{Ch}(n)$ such that no variable of the scope of a child $m \in \text{Ch}(n)$ appears negated in the scope of another child $m' \in \text{Ch}(n)$.

(iii) We call S *complete* when each sum node $n \in N_{\text{sum}}$ has children $\text{Ch}(n)$ with identical scopes, i.e. for all $m, m' \in \text{Ch}(n)$ it holds $\text{Sc}(m) = \text{Sc}(m')$.

(iv) We call S *valid* if and only if for each evidence E it holds $S(E) = \sum_{x \in E} S(x)$.

Poon et al. [5] state the important relation that an SPN is valid if it is complete and consistent.

As declared in Definition 2.1.4, "an SPN is valid if it always correctly computes the probability of evidence." [5] Vice-versa, if an SPN is not valid, there exists evidence where the SPN computes a wrong probability, thus, the SPN represents nonsense. Moreover, the tractability of inference is only guaranteed to be linear in size of the SPN if it is valid. Hence, it is desirable always to construct valid SPNs.

Sum nodes of a valid SPN statistically represent mixtures of probability distributions. Product nodes represent a factorization of independent random variables. Moreover, every SPN S can be expressed as a polynomial of the form $S(z) = \sum_k a_k \prod_l (...)$. [5]

2.2 How Inference with an SPN Works

Computing joint, marginal and conditional probabilities with valid SPNs is very easy and is done in time linear to the model size. Let P be a probability measure and let S denote a valid SPN representing the distribution of P. Furthermore, let $z := (z_1, ..., z_d, \overline{z}_1, ..., \overline{z}_d)$ denote a complete state for Boolean random variables $Z = (Z_1, ..., Z_d)$.

2.2.1 Obtaining the Joint Probability

The value of the root node of S is the joint probability of the variables $Z := (Z_1, ..., Z_d)$. In other words,

$$P(Z_1 = z_1, ..., Z_d = z_d) = S(z_1, ..., z_d, \overline{z}_1, ..., \overline{z}_d). \tag{2.1}$$

2.2.2 Obtaining the Marginal Probability

Consider we want to compute the marginal probability $P(Z_2 = z_2, ..., Z_d = z_d)$. This probability can be written as the sum of joint probabilities such that Z_1 is marginalized out, i.e.

$$P(Z_2 = z_2, ..., Z_d = z_d) = P(Z_1 = 0, Z_2 = z_2, ..., Z_d = z_d) + P(Z_1 = 1, Z_2 = z_2, ..., Z_d = z_d).$$

According to Equation 2.1, these two joint probabilities can also be written as

$$S(0, z_2, ..., z_d, 1, \overline{z}_2, ..., \overline{z}_d) + S(1, z_2, ..., z_d, 0, \overline{z}_2, ..., \overline{z}_d).$$

Since S is valid, for each evidence E it holds $S(E) = \sum_{x \in E} S(x)$. Therefore, the above equation is equivalent to

$$S(1, z_2, ..., z_d, 1, \overline{z}_2, ..., \overline{z}_d).$$

In short, we have

$$P(Z_2 = z_2, ..., Z_d = z_d) = S(1, z_2, ..., z_d, 1, \overline{z}_2, ..., \overline{z}_d). \tag{2.2}$$

In other words, marginalizing out the variable Z_1 can be done by simply setting all corresponding indicators of Z_1 to 1. This is also referred to as *marginal inference*.

2.2.3 Obtaining the Conditional Probability

Assume we want to obtain the conditional probability $P(Y|X)$ of random variables X and Y with complete states x and y, respectively. The mathematical definition of the conditional probability is given as

$$P(Y|X) = \frac{P(X,Y)}{P(X)}.$$

Hence, with Equations 2.1 and 2.2, we get

$$\frac{S(x,y)}{S(x,1)},$$

where 1 is a vector of ones here. So, we can calculate the conditional probability with only two bottom-up evaluations.

2.2.4 Finding the Most Probable Explanation (MPE)

Given a random variable W with state w and unspecified random variables X and Y, what are the "most probable" states x and y such that the probability $P(X = x, Y = y|W = w)$ is maximal? In other words, we want to obtain the state

$$(x,y) = \arg\max_{x,y} P(X = x, Y = y|W = w)$$

which maximizes the probability where a state w prevails. The state (x, y) is also called the *most probable explanation (MPE)* under w.

An approximation of the MPE state can be found with two network evaluations (an upward and a downward pass) where all sum nodes are replaced with max nodes. [5]

MPE is useful for classification problems, where the MPE is equivalent to the predicted label, as well as for regression problems where parts of data (for example the lower half of an image) must be inferred through MPE (given the upper half of that image).

2.3 Learning an SPN

There are two main aspects of learning an SPN: Structure learning and parameter learning.

One idea of learning the structure is starting with a predefined, densely connected but valid SPN. Then, after the parameters are learned, all edges with weights close to zero are pruned, resulting in the final SPN structure. Since only edges emanating from sum nodes do have a weight, discarding them does not affect validity. Thus, the resulting SPN structure is guaranteed to be valid. [5] Another idea was proposed by Gens and Domingos [13] where the SPN structure is learned directly from data.

Given an SPN structure, the parameters can be learned generatively or discriminatively.

- **Generative** SPN learning algorithms try to fit the SPN ideally to the train data such that each class of instances is well approximated with distributions, individually and independently from other classes. To this end, the joint log-likelihood (JLL) per class is taken as the objective to be maximized (locally). Hard Expectation Maximization (EM) or gradient descent can be used to update the parameters, whereas only the first algorithm is suitable for deep SPNs since gradient descent suffers from the gradient diffusion problem (also referred to as vanishing/exploding gradient problem). Poon and Domingos [5] proposed a generative learning algorithm.

- **Discriminative** SPN learning algorithms focus on the segregation of instances from other classes. To this end, the conditional log-likelihood (CLL) may be maximized (locally). The CLL $\log P(y|x)$ can be written as

$$\begin{aligned}\log P(y|x) &= \log \frac{P(y,x)}{P(x)} \\ &= \log P(y,x) - \log P(x) \\ &= \log \sum_h P(y,h,x) - \log \sum_{y,h} P(y,h,x) \\ &\overset{S \text{ valid}}{=} S(y,1,x) - S(1,1,x),\end{aligned}$$

  where h is the state of a hidden variable H and where S returns the logarithm of the joint probability here. As one can see, the CLL can be easily computed with only two network evaluations. Then, (hard) gradient descent, but no (hard) EM, may be used for parameter update. Discriminative learning algorithms were proposed by e.g. Gens and Domingos [14] and by Rashwan et al. [15].

2.4 The Superiority of SPNs Compared to Other Models

Inference in graphical models such as hierarchical mixture models or junction trees is generally intractable. Tractability can only be ensured by low treewidth, but this is a strong restriction. SPNs "overcome this by exploiting context-specific independence and determinism" [13] and allow mixtures over subsets of variables and their reuse. Therefore, SPNs can be exponentially more compact compared to other graphical models. Moreover, SPN evaluation takes linear time with respect to its size. [15] Each, joint, marginal and conditional queries can be answered by no more than two SPN evaluations, hence, in linear time too. This is a substantial advantage since inference in Bayesian and Markov models may grow exponentially with their size. Furthermore, compared to state-of-the-art architectures (of 2012), SPNs can be trained "at least an order of magnitude faster" [5], and an SPN performed temporarily as the most accurate classifier on the STL-10 data set[1]. [14] Besides, SPNs "are more general than both hierarchical mixture models and thin junction trees." [5] Unlike other, widely-used deep architectures such as Convolutional Neural Networks (typically used for image recognition) or Long Short-Term Memory Networks (often used for sequential data like text and speech) where input and output is defined through the structure, input and output of an SPN can be determined during query time. As explained in Section 2.2, for MPE each random variable can be individually selected to serve either as input or as output. In contrast to typical classifiers, the same SPN structure is able to perform different types of tasks at once, such as image classification and image completion, as demonstrated in the following section.

In a nutshell, an SPN is truly a powerful architecture for probabilistic modeling. Therefore, the selection of SPNs for this work as a black-box model is especially interesting.

2.5 Application Examples

In this section, we are going to learn SPNs on several exemplary data sets: on an artificially generated 2D data set and on the MNIST digit data set. Furthermore, we'll have a look at the test set performance and inspect how the MPEs for just a given label looks like. Moreover, the same SPNs are used for an image completion task.

The SPNs are created and trained with `SPFlow`[2], an open-source SPN library for simple SPN learning, inference and manipulation. It uses the generative learning algorithm `LearnSPN` proposed by Gens et al. [13] "`LearnSPN` uses soft EM for instance clustering and maximum likelihood estimates for univariate distributions. Then `LearnSPN` returns a locally optimal SPN, in the sense that no higher-likelihood SPN can be reached from it by a local repartition of variables or instances." [13]

[1] `https://cs.stanford.edu/~acoates/stl10/`
[2] `https://github.com/SPFlow/SPFlow`

2.5.1 2D Classification Problem

Because it can be visualized easily, a 2D classification problem was chosen. Three different data sets with 200 train samples each were generated, visualized in Figure 2.2.

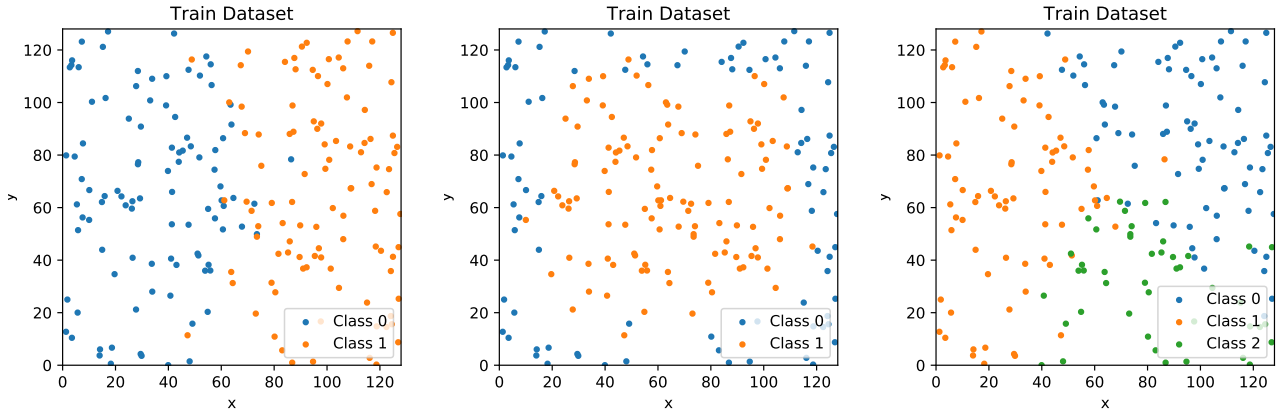


Figure 2.2: Three synthetic 2D data sets: (left) a linearly separable two-class data set, (center) a non-linear two-class data set, and (right) a thee-class data set. Colors indicate classes. Each data set consists of 200 samples and contains a little bit of noise.

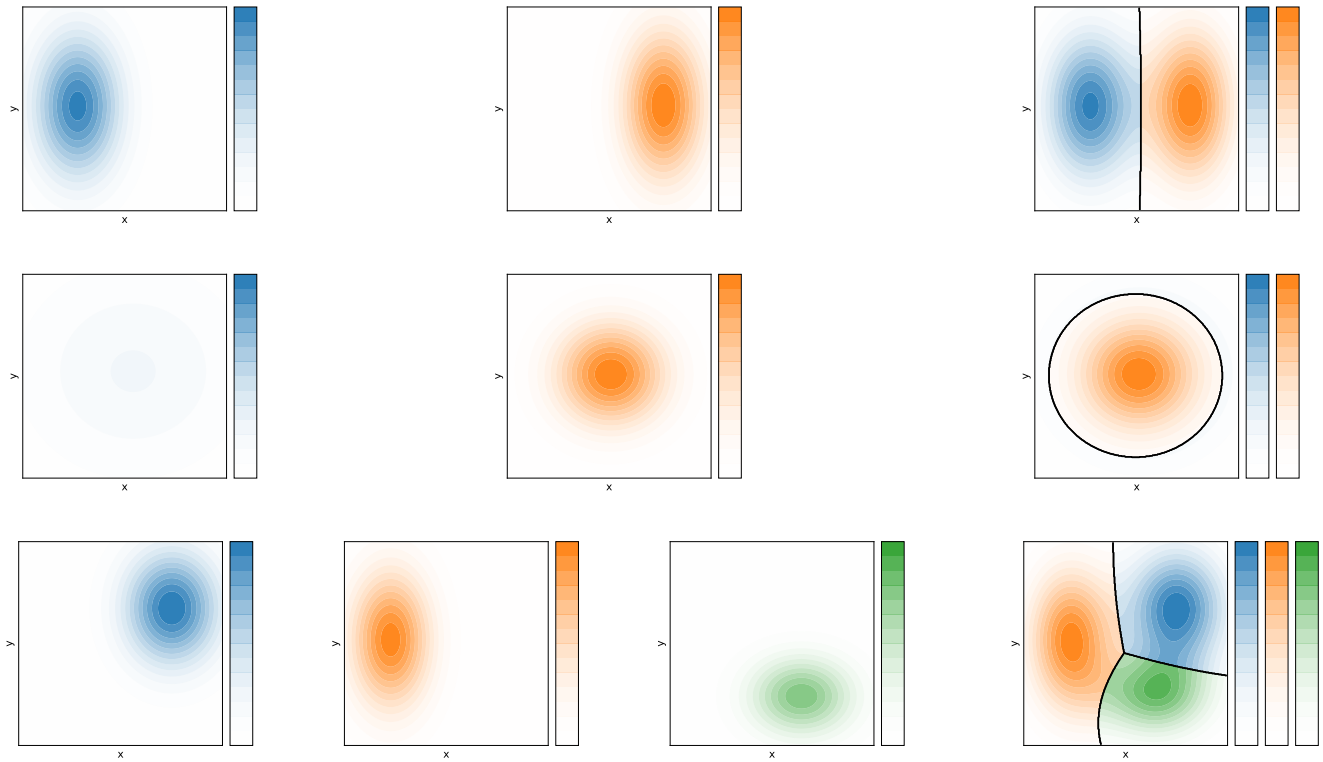


Figure 2.3: The learned distributions for the three data sets. Each row of plots belongs to one learned SPN. First row: linearly separable data set, second row: non-linear data set, third row: three-class data set. The rightmost column depicts the joint probability over all classes (i.e. the sum of all class probabilities). Dark, saturated colors indicate high probability. The black lines show the decision boundary. All other (left handed) plots visualize the individual probability distributions per class.

Our first goal is to learn on each of the three data sets an individual SPN which represents each class with a mixture of two univariate Gaussian distributions, one distribution per coordinate. Each instance has two features (the x and the y coordinate) and a label. Hence, the SPNs we want to train on the data sets take three random variables. Since we want to use Gaussian distributions, we choose Gaussian leaves for the two coordinates. The leaf responsible for labels is a categorical node, solely representing the indicator function $\chi_i(y)$ of the label y defined as

$$\chi_i(y) = \begin{cases} 1, & \text{if } y = i, \\ 0, & \text{otherwise.} \end{cases}$$

As explained in Section 2.3, the generative learning process tries to find for each class individually the parameter values that fit the distributions (represented by the SPN) ideally to the class samples. In our case, mean and variance of each Gaussian distribution are learned such that the JLL of all samples of the corresponding class is maximal, regardless of other classes.

The learned results are shown in Figure 2.3, where the individual distributions per class, the sum of all class probabilities and decision boundaries can be viewed. The linearly separable data set was approximated quite straightforward with two symmetric and equally sized Gaussian distributions resulting in an almost linear decision boundary. In contrast to that, the second data set was learned with two equally centered distributions. These two Gaussians only differ in their variance. In particular, the orange Gaussian has a smaller variance, such that—inside the region of the circular decision boundary—the joint probability of "orange" is higher than of "blue". Eventually, the three-class problem was learned with three completely different distributions.

One should remark that neither the train instances are drawn from Gaussian distributions nor is a representation of bivariate Gaussians allowed. Hence, the lack of covariances does not enable the SPNs to learn diagonally oriented Gaussians because the variances are only oriented in x- and y-direction. Nevertheless, this section does not aim at perfectly trained SPNs but at a visualization of the basic principles.

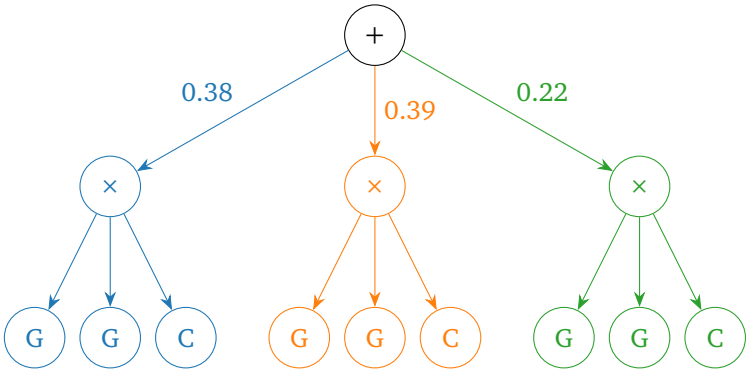


Figure 2.4: Learned SPN structure for the three-class data set. Colors indicate the corresponding class. Leaves with "G" denote Gaussian nodes and leaves with "C" denote categorical nodes. The numbers indicate edge weights.

Let us have a look at the learned SPN structure for the three-class case, depicted in Figure 2.4. The SPN can be divided into three sub-SPNs, each one representing the learned distribution of the corresponding class. Moreover, the weights meet the ratio of the number of corresponding class instances to the total number of instances inside the train data set.

2.5.2 MNIST Digit Classification Problem

The MNIST database[3] is a large open-source database of handwritten digits from the Modified National Institute of Standards and Technology. We are going to train two differently sized SPNs on this data set. Again, we will use only univariate Gaussians as leaves.

One MNIST instance comprises of 28×28 pixels (where each pixel is an integer value ranging from 0 to 255) and of one label (integer from 0 to 9), indicating the true digit represented by the pixels. 10,000 train samples are used for SPN learning. As in the previous example, the generative learning routine is used.

SPN	Train set accuracy	Test set accuracy
Small SPN	50.8%	48.3%
Large SPN	86.0%	76.1%

Table 2.1: Train and test set performance for the small and the large SPN.

A smaller SPN with three layers and approx. 8000 nodes as well as a bigger SPN with 13 layers and approx. 65,000 nodes were learned on the same train set. Their train and test performance can be viewed in Table 2.1. The larger SPN has a better test accuracy compared to the smaller one indicating that the larger SPN was more capable of learning the digits' shape.

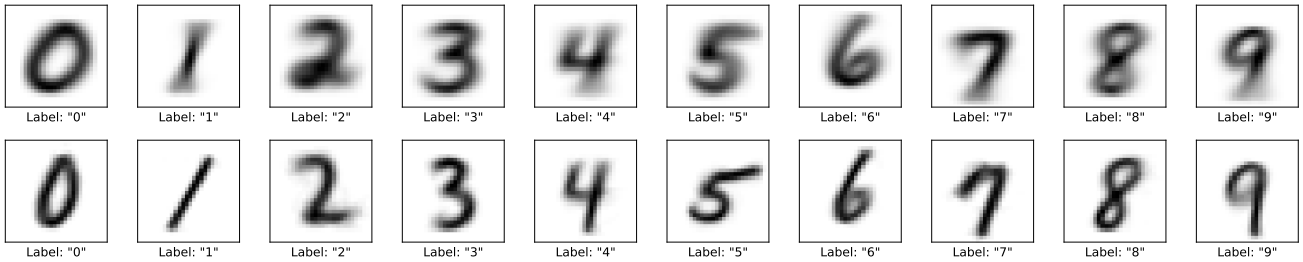


Figure 2.5: Most Probable Explanations (MPE) for all ten given digit labels for both, the small SPN (first row) and the large SPN (second row).

When only the label y is given as input, the SPN can generate digit images x which are "most probable" according to that given label, i.e. the SPN uses MPE to approximate an x maximizing $S(x|y)$. This gives us insight about whether the SPN has learned the shapes of the ten different digits to some extent. Figure 2.5 depicts the MPE for each of the ten labels for both, the small and the large SPN. It can be observed that the MPEs basically meet the true expectation for the corresponding digit label. Generally, the smaller SPN's MPEs are much more blurry than the other ones. Blurriness indicates *bias*. Bias means the inability of a model to fit the train data adequately, for instance, because the structure is not flexible enough like in the case of the small SPN. Therefore, the smaller SPN is more biased than the bigger one.

Looking at the MPE of "1" from the large SPN we can see that the digit is quite clear and almost not blurry. The reason is that, compared to other images, there are fewer variations to write down a "1" by hand, allowing the large SPN to fit the comparably simple distribution of "1" quite well.

Bias can also affect only regions of a digit. As an example, look at the MPE for "9" of the small SPN: The lower half of "9" is more blurry than the upper half, indicating that the SPN is more biased representing the lower half's distribution than the one of the upper half.

[3] `http://yann.lecun.com/exdb/mnist/`

2.5.3 MNIST Completion Problem

Without changing the structure of the larger SPN learned for MNIST classification of the previous section, the same SPN can be reused for image completion. The challenge of the image completion problem is that only a part of an image is given, and the purpose of the model is to complete the other, missing part.

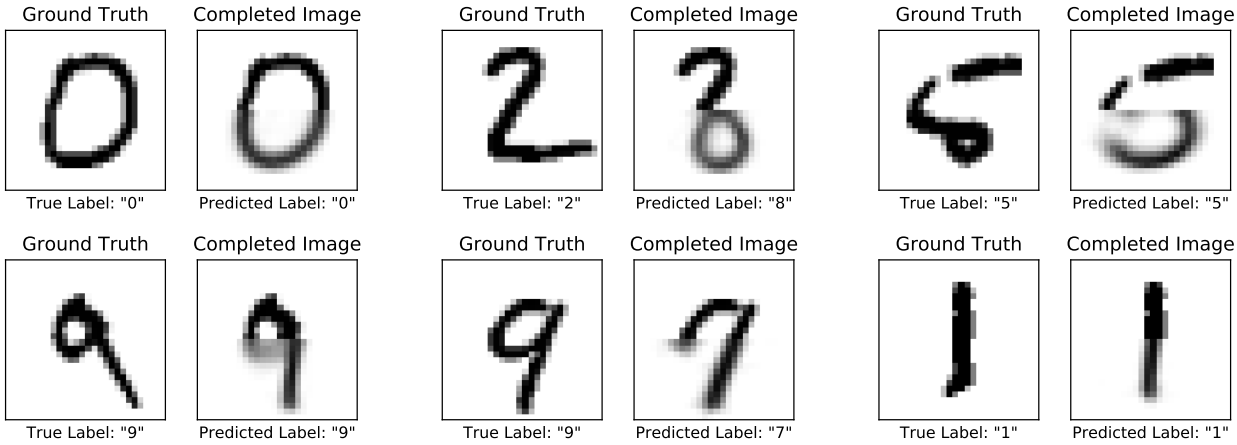


Figure 2.6: Demonstration of the application of an SPN to the image completion problem. Only the upper half and no label was given to the SPN.

Some completion examples processed by the large MNIST SPN can be seen in Figure 2.6. Only the upper half of each image was given and the true label was not known to the SPN. MPE was used to determine the label and to generate the "most probable" lower half of each image. As we can see, there are images completed quite well and images completed wrongly, reflecting the SPN's imperfect test set performance. Again, bias can be observed in the upper center example.

Nevertheless, image completion was done easily by only redefining input and output of the SPN, demonstrating the flexibility of using SPNs.

3 Influence Functions (IFs): Our Tool for Making Black-Box AI More Understandable

Machine learning models often lack understandability. For example, the prediction of a deep neural network (NN) is generally untraceable. Not because it is hard to trace neural signals. Quite the opposite: The signals passed inside of an NN can be followed and computed exactly, from the input layer through hidden neurons, weight multiplications, and activation functions until the signal is released in the output layer. But, the problem is that the meaning of the NN's parameters, such as edge weights or biases, is mainly unclear. As a consequence, it is generally hard to find a semantically comprehensible explanation for an NN prediction without any additional help.

This chapter gives an introduction to *influence functions (IFs)*, a mathematical tool helping to understand black-box models such as NNs. IFs are a classic technique from robust statistics [6]. Koh et al. [7] worked out approaches of using IFs in order to make black-box predictions better understandable. The goal of this chapter is to provide a general introduction into IFs, basing on Koh et al.'s paper. Later on, we will learn important aspects of how IFs may be applied to SPNs in Chapter 4.

First, we are looking at the core idea of measuring the influence in Section 3.1. Second, we do some preparation in advance in Section 3.2 before, third, defining the influence functions in Section 3.3. Finally, Section 3.4 talks about some powerful use cases of IFs and why they may be reasonable to interpret SPNs.

3.1 The Idea of Measuring Influence

The purpose of influence functions is to measure the effect of upweighting/perturbing a single chosen train sample $z \in Z_{\text{train}}$ on the parameters of a model and, in turn, the effect on the loss/score of a specific test sample $z_{\text{test}} \in Z_{\text{test}}$. Without IFs, this effect could be measured by considering the parameters/loss for a model which was re-learned on the modified train data with the upweighted/perturbed sample z. Nevertheless, re-learning a model may be computationally very expensive. However, IFs evade this problem by constructing local second-order Taylor approximations of the model's parameter values regarding the modified train sample z. In other words, IFs approximate the influence of infinitesimally small re-weightings/perturbations of a single train sample on the model parameters. To this end, the loss gradient regarding the model's parameters is computed, which is then used to estimate how parameters would be updated in order to minimize the loss on z_{test}.

3.2 Initial Assumptions and Definitions

Throughout this chapter, consider an unspecified model with parameters θ of a parameter space Θ. Furthermore, let $L : Z \times \Theta \to \mathbb{R}$ denote a loss function for samples out of Z depending on the model parameters θ. Moreover, we will consider the *train loss*

$$\begin{aligned} L_{\text{train}} &: \Theta \to \mathbb{R} \\ \theta &\mapsto L_{\text{train}}(\theta) := \frac{1}{n_{\text{train}}} \sum_{z \in Z_{\text{train}}} L(z, \theta), \end{aligned}$$

which is the mean of the individual losses of all train instances. Define $\hat{\theta} := \arg\min_{\theta \in \Theta} L_{\text{train}}(\theta)$ as the *loss minimizer*.

Koh et al. [7] make two central assumptions about the train loss:

1. The train loss L_{train} is twice-differentiable.
2. The train loss L_{train} is strictly convex in θ.

Property (1) is equivalent to the existence of the Hessian $H_{\hat{\theta}}$ of L_{train}. Property (2) is equivalent to the fact that the Hessian $H_{\hat{\theta}}$ is positive definite. These two assumptions are necessary in order to ensure that the inverse Hessian $H_{\hat{\theta}}^{-1}$ does exist, which is an essential term of the IFs.

3.3 Definition of Influence Functions

Koh et al. [7] listed three different IFs in their paper. We are going to introduce these three IFs, one by one.

3.3.1 Influence on the Model Parameters

The first influence function $I_{\text{up,params}}$ is the function measuring the **influence of a train sample on each parameter of the model**. It is defined as follows.

Definition 3.3.1. Let $z \in Z_{\text{train}}$ be a labeled train sample and let $\hat{\theta}_{\varepsilon,z}$ denote the loss minimizer where z is upweighted by an infinitesimally small $\varepsilon \in \mathbb{R}$. Then, $I_{\text{up,params}}$ is defined as

$$I_{\text{up,params}} : Z_{\text{train}} \to \mathbb{R}^{n_{\text{params}}}$$
$$z \mapsto I_{\text{up,params}}(z) := \left.\frac{\mathrm{d}\hat{\theta}_{\varepsilon,z}}{\mathrm{d}\varepsilon}\right|_{\varepsilon=0} = -H_{\hat{\theta}}^{-1} \cdot \nabla_\theta L(z, \hat{\theta}),$$

where $H_{\hat{\theta}} := \frac{1}{n_{\text{train}}} \sum_{z \in Z_{\text{train}}} \nabla_\theta^2 L(z, \hat{\theta})$ is the Hessian of the loss regarding θ, and where $\nabla_\theta L(z, \hat{\theta})$ is the gradient of the loss regarding θ.

This function returns a vector of values where each value indicates the influence of the train sample on the corresponding model parameter. The higher the influence, the stronger is the parameter changed when the train sample becomes upweighted.

3.3.2 Influence on a Chosen Test Sample

The estimated influence of a train sample on the model's parameters can be used, in turn, to approximate the **influence on the loss of a specific test sample** z_{test}. This is what the second influence function $I_{\text{up,loss}}$ does.

Definition 3.3.2. Let $z \in Z_{\text{train}}$ be a labeled train sample, $z_{\text{test}} \in Z_{\text{test}}$ a labeled test sample, and let $\hat{\theta}_{\varepsilon,z}$ denote the loss minimizer where z is upweighted by an infinitesimally small $\varepsilon \in \mathbb{R}$. Then, $I_{\text{up,loss}}$ is defined as

$$I_{\text{up,loss}} : Z_{\text{train}} \times Z_{\text{test}} \to \mathbb{R}$$
$$(z, z_{\text{test}}) \mapsto I_{\text{up,loss}}(z, z_{\text{test}}) := \left.\frac{\mathrm{d}L(z_{\text{test}}, \hat{\theta}_{\varepsilon,z})}{\mathrm{d}\varepsilon}\right|_{\varepsilon=0}$$
$$= \left(\nabla_\theta L(z_{\text{test}}, \hat{\theta})\right)^T \cdot \underbrace{\left(-H_{\hat{\theta}}^{-1} \cdot \nabla_\theta L(z, \hat{\theta})\right)}_{=I_{\text{up,params}}(z)}.$$

As indicated in the last line of the formula, $I_{\text{up,loss}}$ is the inner product of the influence of z on the parameters and the loss gradient at z_{test} regarding the parameters. In other words, influence of z on the loss of z_{test} is simply the sum of all individual effects the parameters have on the loss of z_{test}, weighted by the influences given by $I_{\text{up,params}}$.

3.3.3 Influence Contribution of Individual Features

Sometimes, it is interesting to know not only the influence of a train sample as a whole but also how the single features of the train sample contribute to the influence individually. To this end, $I_{\text{pert,loss}}$ computes the **influence gradient for every single feature according to the test sample loss**.

Definition 3.3.3. Let $z \in Z_{\text{train}}$ be a labeled train sample, $z_{\text{test}} \in Z_{\text{test}}$ a labeled test sample, and let $\hat{\theta}_{z_\delta,-z}$ denote the loss minimizer where z is replaced by $z_\delta := z + \delta$. Then, $I_{\text{pert,loss}}$ is defined as

$$\begin{aligned} I_{\text{pert,loss}} : Z_{\text{train}} \times Z_{\text{test}} &\to \mathbb{R}^{n_{\text{input}}} \\ (z, z_{\text{test}}) \mapsto I_{\text{pert,loss}}(z, z_{\text{test}}) &:= \nabla_\delta L(z_{\text{test}}, \hat{\theta}_{z_\delta,-z})\big|_{\delta=0} \\ &= -\left(\nabla_\theta L(z_{\text{test}}, \hat{\theta})\right)^T \cdot H_{\hat{\theta}}^{-1} \cdot \nabla_x \nabla_\theta L(z, \hat{\theta}), \end{aligned}$$

where n_{input} is the input dimension (number of input values/features).

The real value $I_{\text{pert,loss}}(z, z_{\text{test}})^T \cdot \delta$ tells us the approximate effect of $z \mapsto z + \delta$ having on the loss of z_{test}.

3.4 Some Use Cases of IFs

IFs can be helpful in several use cases. For example:

- **Identification of malicious train samples**: The influence is a powerful indicator in order to identify malicious train samples. If a train sample z has a high influence on the loss of z_{test} (i.e. $I_{\text{up,loss}}(z, z_{\text{test}}) >> 0$), then z must be a helpful sample. In contrast, a low valued influence (i.e. $I_{\text{up,loss}}(z, z_{\text{test}}) << 0$) means that z has a negative effect on the loss of z_{test}. Hence, z is a malicious train sample. This information can be helpful for data cleaning. But, be aware: A sample z which is malicious for the prediction of a single test sample could, in contrast, be helpful for other test samples. Hence, removing z from the train set (for data cleaning purposes) without checking influence on other test samples is not recommended.
- **Identification of most responsible features**: The IF $I_{\text{pert,loss}}$ returns a gradient vector indicating the influence of each single feature of a train sample. A high influence of a feature means that increasing the value of that feature has a positive effect on the loss of z_{test} and, conversely, a negative influence value means that increasing the feature value has a bad effect. Influences close to zero indicate a low contribution to the loss of z_{test}, i.e. modifying these feature values does not change prediction on z_{test} very much. In contrast to that, features with strong influence are especially responsible for prediction changes. This property will serve us in order to identify responsible features in Chapter 4, seeing whether the model is *right for the right reasons* [8].
- **Train set attacks**: Taking the aforementioned properties of $I_{\text{pert,loss}}$ into consideration, moving z into the opposite direction of $I_{\text{pert,loss}}(z, z_{\text{test}})$ maximally increases loss on z_{test}. Hence, z can be manipulated such that the model makes completely wrong predictions on z_{test}. This is called *train set attack*. Koh et al. used this IF successfully to fool an image classifier, whereas changes in the modified image were not visible to the human eye. [7] Train set attacks serve as an effective approach to test the robustness of a classifier.

4 Interpreting SPNs with Influence Functions

Finally, we are ready for the attempt of laying the fundamentals of interpreting SPNs via IFs. The goal of this chapter is explicitly not to generate watertight and easily understandable explanations for a given SPN. Rather, the goal is to study the most important aspects of how IFs may be applied to SPNs reasonably and what the influence values actually mean in the context of SPNs. A thorough understanding of these aspects is required in order to interpret the IFs' outcomes, so that valuable conclusions about the model can be made.

This chapter contains the core work of the presented thesis. Here, we are going to unite the basics introduced in the previous two chapters. At first, we define two adequate loss functions in Section 4.1. Next, in Section 4.2, the influence contribution of the different SPN parameter types is investigated which is necessary in order to understand the influence outcomes. After that, the effect of the Hessian on the influence is examined in Section 4.3 followed by a comparison of the explanatory power of the two previously defined loss functions in Section 4.4. Then, we document the existence of influence noise in Section 4.5 and close this chapter with a look at some especially helpful or malicious MNIST images in Section 4.6.

In this chapter, the SPNs from Chapter 2 (created and learned with `SPFlow`) are re-used. The code implemented for the conversion of the SPNs into a proper format, doing the influence computations on them and visualizing the IF values can be found in this[1] GitHub repository, which I made publicly available. The code uses `TensorFlow` and the pre-implemented IFs from the repository[2] provided by Koh et al. [7].

For simplicity and comparability, the leaves of the used SPNs in this thesis are only Gaussian and categorical leaves. Choosing other leave types does not affect the main contribution of this work.

4.1 Defining the Loss Function

For influence investigation, we need to define a proper train loss function L_{train} (see also Section 3.2).

Like the SPNs in Section 2.5, the SPNs in this chapter are learned generatively with the `LearnSPN` routine of Gens et al. [13] As stated in Section 2.2, generative learning algorithms maximize the joint log-likelihood. In other words, the objective

$$\log P(z_1, ..., z_{n_{\text{train}}}) \overset{\text{i.i.d.}}{=} \sum_{z \in Z_{\text{train}}} \log P(z) = \sum_{z \in Z_{\text{train}}} S(z)$$

is maximized. Based on this, we define the loss function $L^{\text{JLL}}_{\text{train}}$ as the scaled negation of the above equation.

Definition 4.1.1. We call $L^{\text{JLL}}_{\text{train}}$ the *train loss according to the joint log-likelihood* (JLL) and write

$$L^{\text{JLL}}_{\text{train}}(\theta) := -\frac{1}{n_{\text{train}}} \sum_{z \in Z_{\text{train}}} S(z),$$

where S is the SPN depending on the parameters θ. Based on this, we define $L^{\text{JLL}}(z, \theta) := -S(z)$ as the individual loss of a sample z according to the JLL.

[1] `https://github.com/MaggiR/Interpreting-SPNs`
[2] `https://github.com/kohpangwei/influence-release`

Additionally, we also consider an other loss definition based on the conditional log-likelihood, defined analogically.

Definition 4.1.2. We call $L_{\text{train}}^{\text{CLL}}$ the *train loss according to the conditional log-likelihood* (CLL) and write

$$L_{\text{train}}^{\text{CLL}}(\theta) := -\frac{1}{n_{\text{train}}} \left(\sum_{(x,y) \in Z_{\text{train}}} (S(x,y) - S(x,1)) \right).$$

where S is the SPN depending on the parameters θ. Based on this, we declare $L^{\text{CLL}}(z,\theta) := S(x,1) - S(x,y)$ as the individual loss of a sample $z = (x,y)$ according to the CLL.

Later, we will see that these two loss functions will give us influence values of different quality and explanatory power.

4.2 Influence Contribution of SPN Parameters

The goal of this section is to get familiar with influence values returned by $I_{\text{up,loss}}$ and $I_{\text{pert,loss}}$ starting from a simple and easy comprehensible basis. To this end, the SPN trained on the linearly separable 2D data set from Section 2.5 will serve as the model under influence investigation. Step by step, we are going to have a look at the individual influence contributions of the SPN's parameters in order to understand the overall influence results finally.

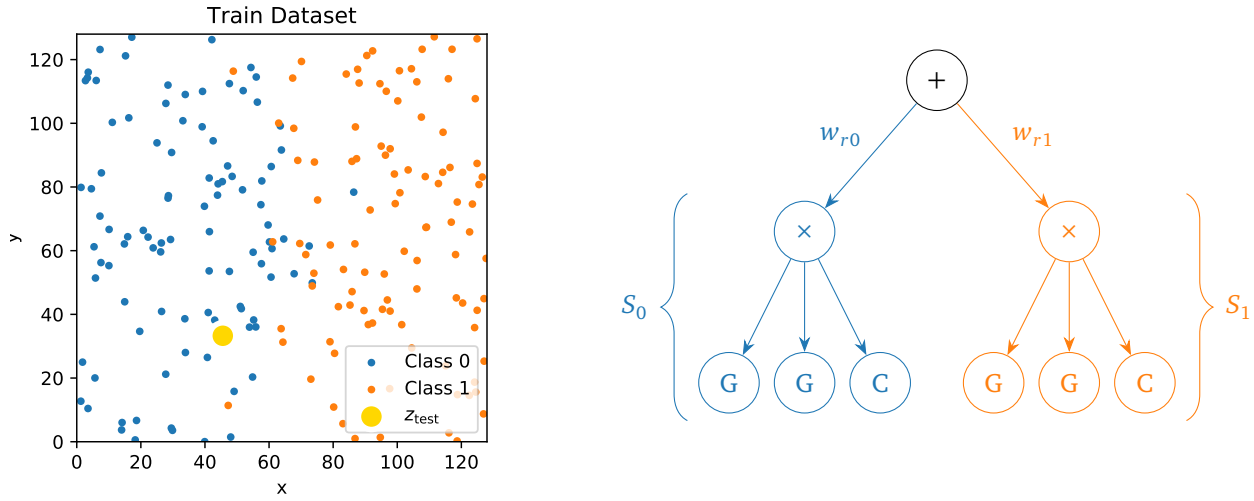


Figure 4.1: Left: the linearly separable, 2D two-class data set with the regarded test sample z_{test}. Right: the SPN S learned on the left data set with weights w_{r0} and w_{r1} and sub-SPNs S_0 and S_1.

Let S be the SPN from Section 2.5 trained on the linearly separable 2D data set. Both, the data set (with z_{test}) and S are visualized in Figure 4.1. The test sample z_{test} has label "0". The SPN consists of two sub-SPNs S_0 and S_1 (weighted with w_{r0} and w_{r1}), individually representing the class distributions of class 0 and class 1, respectively. S has Gaussian and categorical leaves. Hence, it has three different types of parameters: Sum node weights, Gaussian means and Gaussian variances, written as $\theta = (\theta_{\text{weights}}, \theta_{\text{means}}, \theta_{\text{variances}})$. We can make undisturbed interpretations of the influence values by allowing only one type of parameters to be influenced. This can be done easily by replacing θ in the definition of the IFs with θ_{weights}, θ_{means} or $\theta_{\text{variances}}$, respectively.

The procedure will be the following: First, we are going to allow only one parameter type to change. Second, we formulate expectations on the results of $I_{\text{up,loss}}$ and $I_{\text{pert,loss}}$, each with respect to $L_{\text{train}}^{\text{JLL}}$. Third, we generate plots and finally summarize our observations and interpretations.

4.2.1 Influence Contribution of the Weights

Assume that the sum node weight parameters θ_{weights} are the only influenceable SPN parameters, which semantically means that only the relation between the two class distributions (more precisely, the area under the distribution curves) can be changed. For instance, increasing the weight of an edge pointing to a sub-SPN S_i is equivalent to raising the area under the distribution curve represented by S_i which is the same as increasing the likelihood of each sample of class i linearly. Upweighting a train sample z of class i increases the weight w_{ri} because the ratio of class i compared to other classes is increased. As z_{test} is of class "0", train samples also of class "0" are expected to have a positive (good) influence on the loss $L^{\text{JLL}}(z_{\text{test}}, \theta_{\text{weights}})$ of the test sample z_{test}. Conversely, since weights of a sum node always sum up to 1, consequently the other weight must decrease. Therefore, samples of the other class are expected to have a negative (bad) influence. Furthermore, one could expect the position of the samples to have no effect on the influence since θ_{weights} is determined by the number of train samples and not by their location. As a consequence, influences inside of a class are expected to be homogeneous and the feature influences are expected be zero.

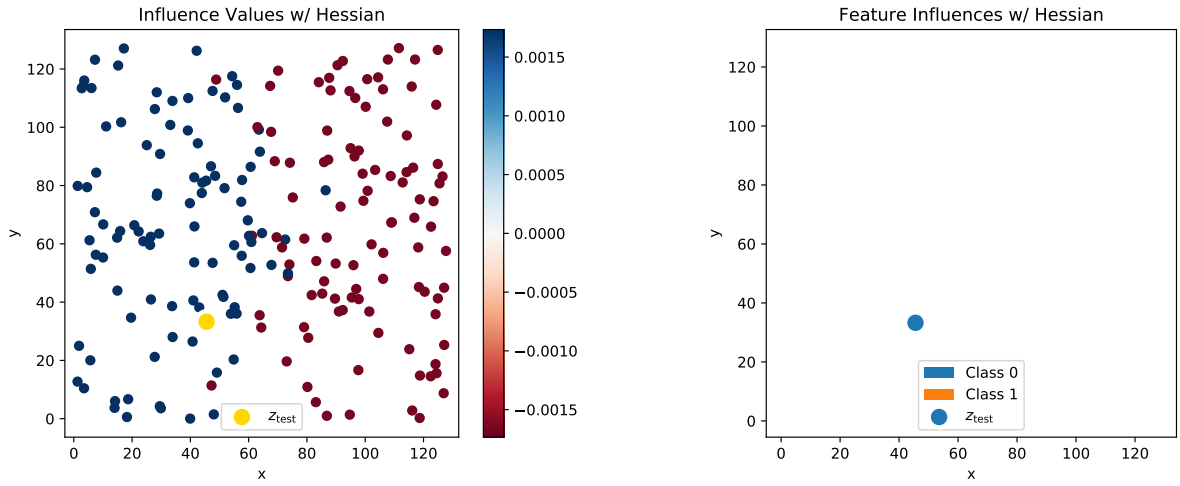


Figure 4.2: Influences where **only weights** are changeable. Left: influence values $I_{\text{up,loss}}$ of each train sample on the loss of the yellow test sample. Right: vector field of feature influences $I_{\text{pert,loss}}$.

Figure 4.2 visualizes the results of $I_{\text{up,loss}}(z, z_{\text{test}})$ and $I_{\text{pert,loss}}(z, z_{\text{test}})$ for each $z \in Z_{\text{train}}$ regarding test sample z_{test}. Indeed, the plots do meet the expectations. Samples of the same class as the one of z_{test} have a uniformly positive influence. All others have a uniformly negative influence. Perturbing the features of a sample (which is the same as moving the sample inside the 2D plane) has no influence on z_{test}'s loss.

4.2.2 Influence Contribution of the Means

Now, assume that only the means θ_{means} are changeable. The SPN S is learned generatively, which means that the sub-SPNs S_0 and S_1 are learned individually and independently on the corresponding classes "0" and "1". Therefore, samples of one class have no influence on the parameters of the sub-SPN of the other class. As an example, a train sample $z = (x, 1)$ from class "1" does not influence the parameters of S_0. Moreover, $S_i(x, y) = 0$ if $i \neq y$, i.e. a sub-SPN's value is zero if the sample fed to the SPN is from an other class than the one represented by the sub-SPN. Hence, since z_{test} is from class "0", the parameters of S_1 have no relevance for the loss of z_{test}. In other words

$$L^{\text{JLL}}(z_{\text{test}}, \theta_{\text{means}}) \overset{\text{Def. 4.1.1}}{=} -S(z_{\text{test}}) \overset{\text{Def. 2.1.1}}{=} -w_{r0} \cdot S_0(z_{\text{test}}) - w_{r1} \cdot S_1(z_{\text{test}}) = -w_{r0} \cdot S_0(z_{\text{test}}).$$

Since samples of class "1" have no influence on the parameters of S_0, they do not contribute to $L^{\text{JLL}}(z_{\text{test}}, \theta_{\text{means}})$ at all. Hence, samples of class "1" are expected to have no influence and samples of class "0" are expected to have influence unequal to zero in general.

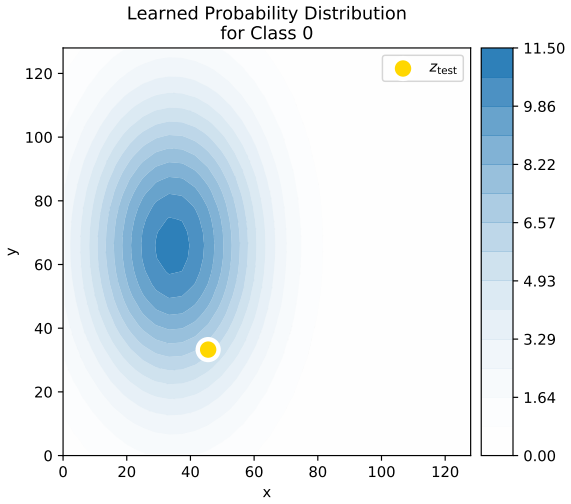


Figure 4.3: Learned probability distribution for class 0. The yellow point is the regarded test sample.

But, how are the influence values of samples of class 0 are expected to look like? In order to answer this question, consider Figure 4.3 which depicts probability for class "0" and the location of z_{test}. A sample has a positive influence if it helps to decrease the loss on z_{test}, i.e. to increase the likelihood $S_0(z_{\text{test}})$. Because only means are allowed to be changed, the mean of the blue distribution must be "moved" closer to z_{test} such that the likelihood increases the most. The increase of likelihood is the highest when the mean is moved into the same direction as the vector of the steepest descent $-\nabla_x S(z_{\text{test}})$ of the likelihood at z_{test}. Therefore, samples located in the direction of the steepest descent ("pulling" the mean into the direction of $-\nabla_x S(z_{\text{test}})$) are expected to have a positive influence. Conversely, samples located in the opposite direction are expected to have a negative influence.

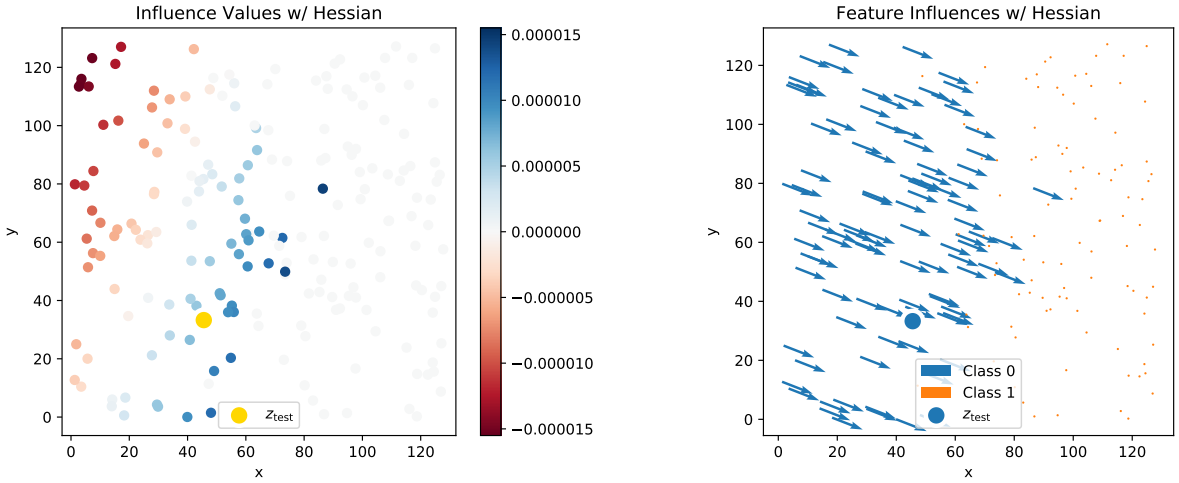


Figure 4.4: Influences where **only means** are changeable. Left: influence values $I_{\text{up,loss}}$ of each train sample on the loss of the yellow test sample. Right: vector field of feature influences $I_{\text{pert,loss}}$.

Regarding $I_{\text{pert,loss}}$, the location of the samples is expected to have no effect because "moving" a sample into a certain direction of the 2D plane always has the same influence on the mean, no matter where

the sample is located. This fact leads to the expectation that the vectors of the feature influences are all equally long and point all into the same direction. The direction is expected to be $-\nabla_x S(z_{\text{test}})$.

Figure 4.4 depicts the visualizations for the influence values where only the means are changeable. The aforementioned expectations match the plots. In a nutshell, samples located (compared to the mean) in the direction of the steepest descent of the likelihood in z_{test} have a positive influence. The higher the distance (in direction of the steepest descent) to the mean the stronger is the influence. Samples have a bad influence when located in the opposite direction. Feature influences are uniform and point into the direction of the steepest likelihood descend. Moreover, (features of) samples of the class other than the one of z_{test} have no influence.

4.2.3 Influence Contribution of the Variances

Next, consider that only variances are changeable. Analogously to the case where only means are changeable, samples from class "1" are expected to have no influence. Considering samples of class "0", how can the likelihood of z_{test} be increased now? To answer this question, we must first realize that the answer is case dependent. Figure 4.5 visualizes the different cases for one dimension where the likelihood of a sample with a fixed location is considered under different Gaussian variances. It can be easily observed, that a high variance must be decreased in order to raise the likelihood of the sample. Conversely, a low variance must be increased.

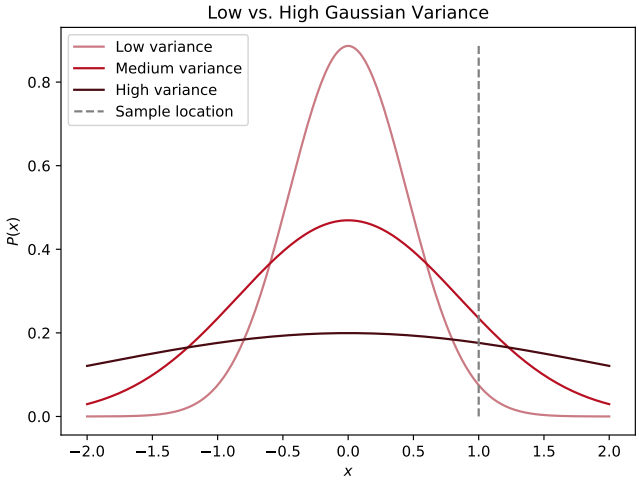


Figure 4.5: Gaussian distributions with different given variances. The dashed line marks a notional sample located at 1. Whether increasing/decreasing the variance increases/decreases likelihood of the sample depends on the given variance.

Variance can be decreased by upweighting samples close to the mean or by downweighting samples far away from the mean. Therefore, if variance is high, samples close to the mean are expected to have a positive influence and samples far away from the mean are expected to have a negative influence (contrary for a low variance). Moreover, a high variance can be reduced by "moving" the samples "closer" to the mean. Hence the feature influence vector field is expected to have vectors pointing radially into the direction of the mean (contrary for a low variance).

The influence values can be viewed in Figure 4.6. Both, the expectation for $I_{\text{up,loss}}$ and for $I_{\text{pert,loss}}$ are only partially true. The left plot contains many samples having a similar influence but having very different distances to the mean. More precisely, the expectation is met in x-direction, but not in y-direction. Similar in the right plot: Influence vectors are pointing mainly in parallel to the x-axis, into the x-direction of the mean. Moreover, vectors are longer (i.e. feature influences are higher) when samples are farther away

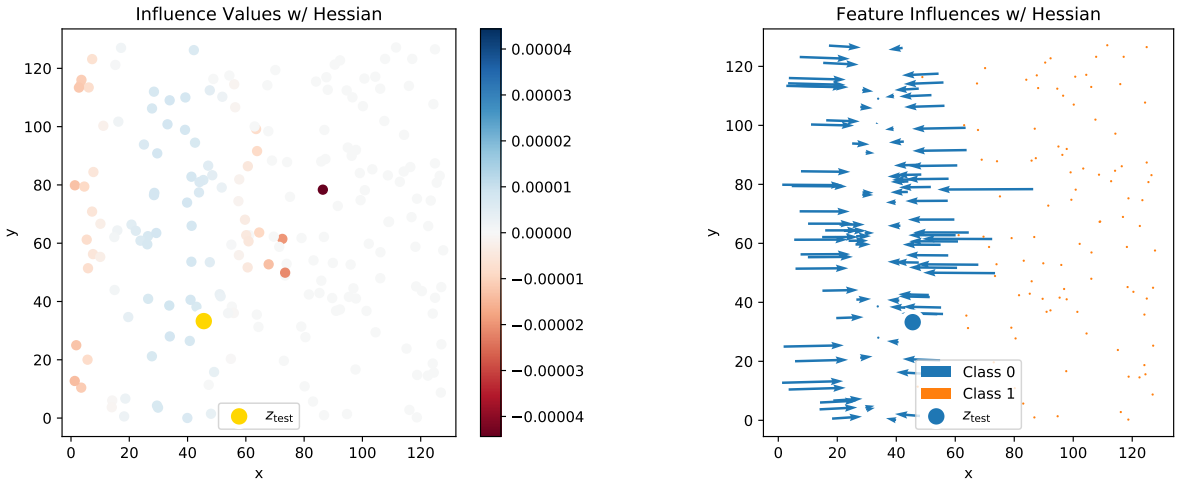


Figure 4.6: Influences where **only variances** are changeable. Left: influence values $I_{\text{up,loss}}$ of each train sample on the loss of the yellow test sample. Right: vector field of feature influences $I_{\text{pert,loss}}$.

from the mean in x-direction. The reason for this outcome is that the variances in x- and in y-direction belong to two different, univariate Gaussian distributions. In this example, the variance in x-direction has a much higher influence than the variance in y-direction such that the influence in y-direction is almost not recognizable anymore.

In a nutshell, samples far away from or very close to the mean have strong influence. Sign of influence depends on the location of z_{test} and the a priori value of the learned variance. Furthermore, influence contribution of variance parameters from different Gaussians can deviate heavily, causing some feature influences (cf. y-direction) to be irrecognizable because other feature influences (cf. x-direction) drown them out. Feature influence vectors point onto the mean regarding their univariate feature space (but not necessarily regarding the combined, multivariate feature space). Feature influence rises with increasing sample-mean distance.

4.2.4 The Contributions Altogether

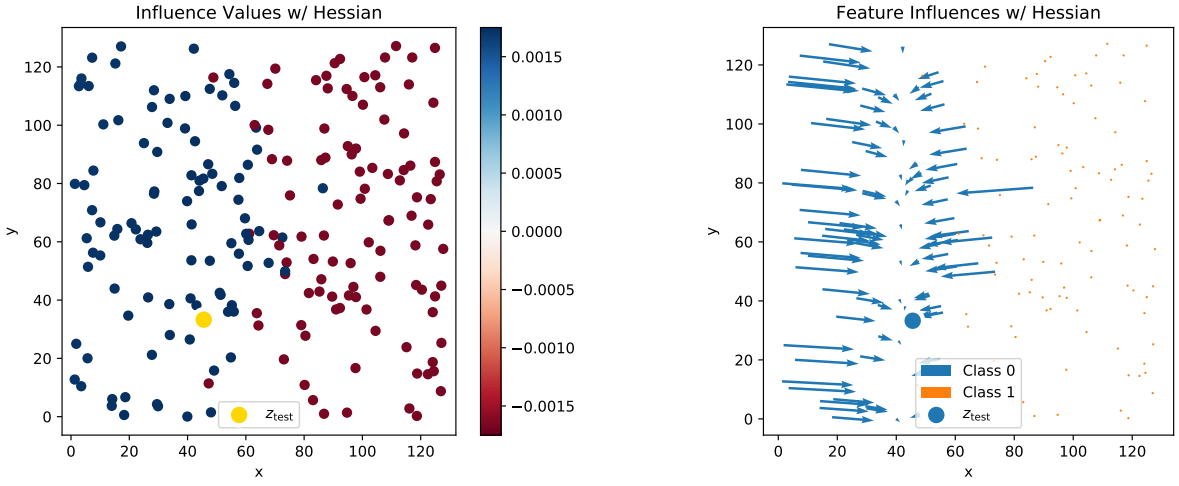


Figure 4.7: Influences where **all parameters** are changeable. Left: influence values $I_{\text{up,loss}}$ of each train sample on the loss of the yellow test sample. Right: vector field of feature influences $I_{\text{pert,loss}}$.

Putting the observations of the last three sections together and considering the formulas of $I_{\text{up,loss}}$ and $I_{\text{pert,loss}}$, the overall influence is expected to be similar to a linear combination of the previously observed influences regarding the three parameter types. According to the absolute influence values, the contribution of the weights (values in order of 10^{-3}) is expected to be the greatest compared to the other two parameter types (values in order of 10^{-5}).

As we can see in Figure 4.7, the expectations are met. In the left plot, the weight influences do outweigh the influences of the other parameters significantly, such that the contribution of means and variances is not recognizable. In contrast to that, the right plot shows a vector field which is comprehensibly a combination of the vector plots in Figures 4.4 and 4.6.

In short, the overall influence values are a combination of the influences of all the single parameters, where some parameters may outweigh the influence of others dramatically.

4.3 Contribution of the Hessian

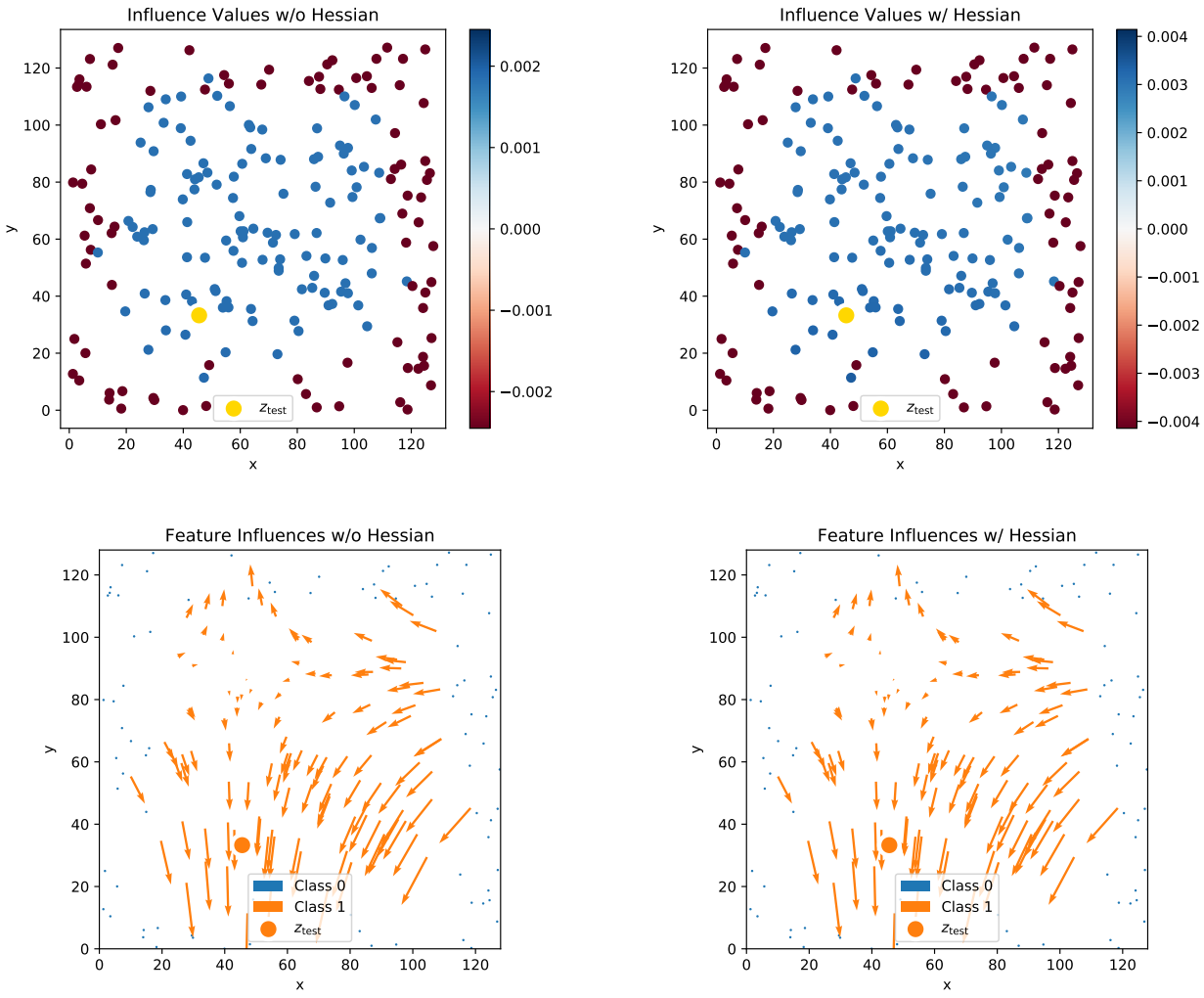


Figure 4.8: Influences for the non-linear two-class data set. The upper two plots compare the influences of $I_{\text{up,loss}}$ with and without Hessian. The lower two plots compare the feature influences of $I_{\text{pert,loss}}$ with and without Hessian. Arrow colors indicate true class label.

Since the influence functions originate from second order Taylor approximations, they contain the Hessian matrix of the loss regarding the model parameters. However, the number of Hessian entries grows

quadratic compared to the number of model parameters. Hence, the Hessian can become very huge for models with several thousands of parameters. Furthermore, taking the inverse of that matrix is computationally very expensive. Therefore, one might ask whether the Hessian is necessary for influence computation at all. Indeed, there are some cases where the Hessian does not have any remarkable effect, as one can see in Figure 4.8.

Let's look at the entries of the Hessian. According to Definition 3.3.1, the Hessian $H_{\hat{\theta}}$ is defined as the matrix of all second partial derivatives of the train loss function $L_{\text{train}}^{\text{JLL}}$ regarding the parameters θ, i.e. $H_{\hat{\theta}} := \frac{1}{n_{\text{train}}} \sum_{z \in Z_{\text{train}}} \nabla_\theta^2 L(z, \hat{\theta})$. Assume S to be an SPN with parameter vector $\theta = (w_{r0}, w_{r1}, \mu_0, \mu_1, \sigma_0, \sigma_1)$, representing two classes of different 1D distributions, with a single univariate Gaussian each. Then, S can be written as a function $S(x, y)$ which takes a feature x and a label y as input and which has the form

$$S(x, y) = w_{r0} \cdot G_{\mu_0,\sigma_0}(x) \cdot \chi_0(y) + w_{r1} \cdot G_{\mu_1,\sigma_1}(x) \cdot \chi_1(y),$$

where $G_{\mu,\sigma}(x)$ is a Gaussian distribution with mean μ and variance σ, and where χ_i is an indicator function. Calculating each matrix entry by hand gives us the Hessian

$$\nabla_\theta^2 L(x, y, \hat{\theta}) = -\nabla_\theta^2 S(x, y) = -\begin{pmatrix} \frac{\partial^2 S}{\partial w_{r0} \partial w_{r0}}(x, y) & \cdots & \frac{\partial^2 S}{\partial w_{r0} \partial \sigma_1}(x, y) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 S}{\partial \sigma_1 \partial w_{r0}}(x, y) & \cdots & \frac{\partial^2 S}{\partial \sigma_1 \partial \sigma_1}(x, y) \end{pmatrix} = \begin{pmatrix} 0 & 0 & * & 0 & * & 0 \\ 0 & 0 & 0 & * & 0 & * \\ * & 0 & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 & * \\ * & 0 & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 & * \end{pmatrix},$$

where asterisks $(*)$ indicate values which are generally $\neq 0$, and where rows and columns follow the order $(w_{r0}, w_{r1}, \mu_0, \mu_1, \sigma_0, \sigma_1)$. Since (according to Definition 3.3.1) the Hessian $H_{\hat{\theta}}$ is the sum of matrices of the above form it follows that almost the half of the entries of the Hessian are expected to be $\neq 0$. Transferring this fact to the general case with arbitrary SPNs S with Gaussian and categorical leaves, one expects that there are situations where the Hessian does have an effect on the influence values. The multiplication with a matrix is a linear map. Thus, one would expect that the Hessian adjusts the relation between the influences of different parameters and distributions.

As a matter of fact, Figure 4.9 depicts an example where the Hessian has a strong effect on the influences. Influence values returned by $I_{\text{up,loss}}$ as well as feature influences returned by $I_{\text{pert,loss}}$ change severely when the Hessian is omitted. In particular, consider the samples inside the rectangle of the left plot. The IF $I_{\text{pert,loss}}$ considers the samples to "belong" to the top left blue distribution because the vectors point radially away from the distribution's approximate mean location. Here, we have a similar situation as in Figure 4.6: Samples far away from the mean have a high influence on the variance parameters of the corresponding distribution. This is also the case here, with regard to the variances of the top left blue distribution. Nevertheless, the framed samples appear to have a quite different influence when Hessian is taken into account. Why? Since the Hessian is the second derivative of the loss, it provides second order information about the influence contribution of the different parameters. Hence, influence values which are overestimated by an IF, such as the framed feature influences, are corrected by the Hessian. In other words, what the Hessian does is the balancing of influence values in the manner that the whole entirety of distributions is taken into consideration, i.e. parameters of adjacent distributions are considered together instead of focusing on only a single distribution (as the top left blue one).

Recapitulated, the Hessian has indeed an effect on influence values. There are cases where the effect is very strong, and there are also cases where the effect is not visible. Moreover, the Hessian provides second order information helping to balance the contribution of different distributions and parameters to the influence.

Besides, the inverse Hessian can be approximated with the *Hessian Vector Product (HVP)* as Koh et al. [7] did it, avoiding the computationally expensive inversion of many second order derivatives. In this work, the *Linear time Stochastic Second-Order Algorithm (LiSSA)* [16] is used for Hessian approximation.

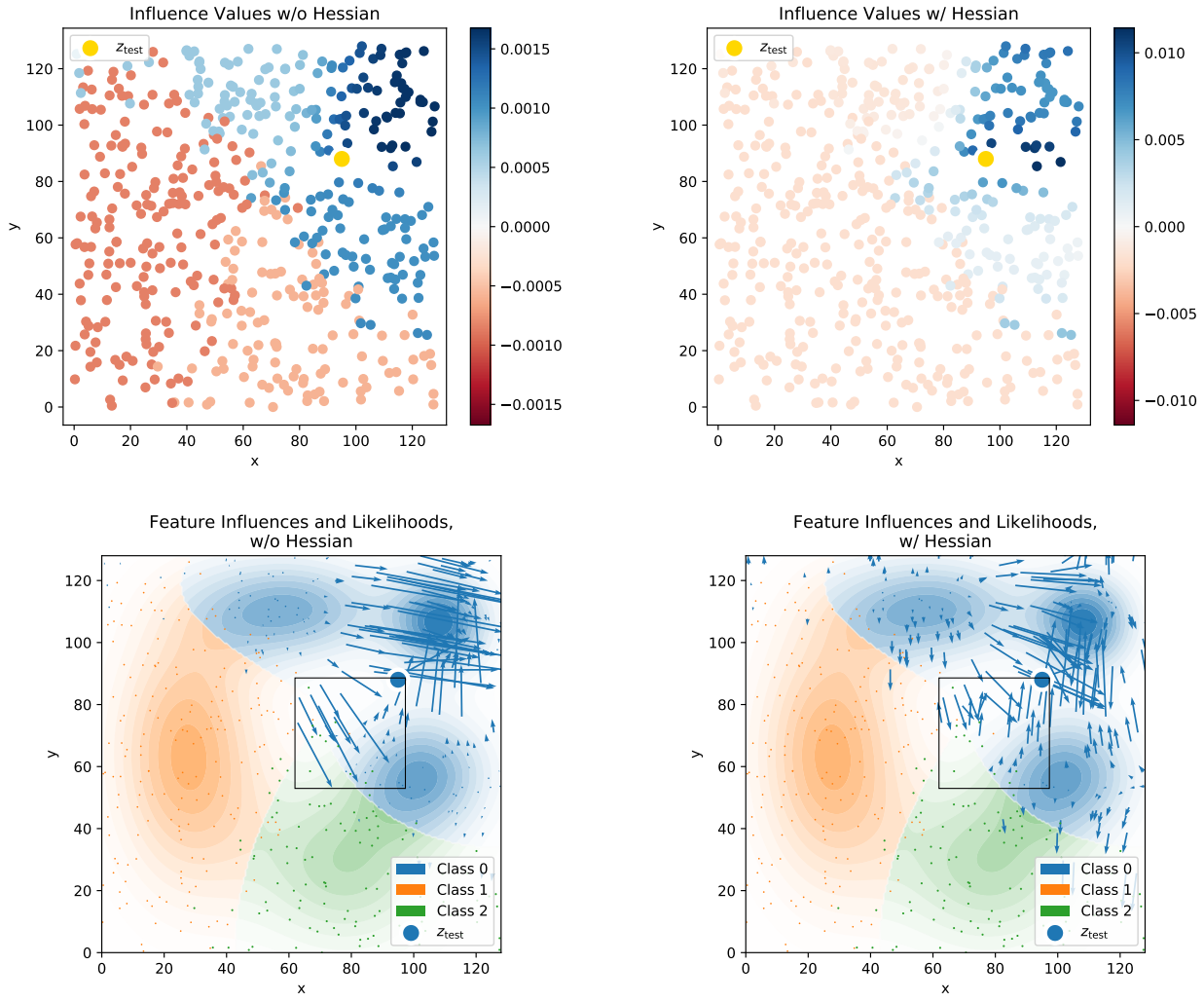


Figure 4.9: Influences for the three-class data set with 500 samples where a mixture of three Gaussians is used for representation of the blue class. The upper two plots compare the influences of $I_{\text{up,loss}}$ with and without Hessian. The lower two plots compare the feature influences of $I_{\text{pert,loss}}$ with and without Hessian. Arrow colors indicate true class label. Additionally, the marginal likelihood is incorporated in the lower plots, visualizing the learned distributions. Samples inside the rectangle show the effect of the Hessian especially well.

4.4 Selecting a Suitable Loss Function

We defined two different loss functions in Section 4.1: the train loss $L_{\text{train}}^{\text{JLL}}$ regarding the JLL and the train loss $L_{\text{train}}^{\text{CLL}}$ regarding the CLL. This section is going to examine and compare the explanatory expressiveness of both losses.

As stated in Section 4.2, samples from a class other than the class of the regarded test sample z_{test} have quite homogeneous influence and their feature influences are always null vectors (cf. Figures 4.7, 4.8 and 4.9). Brief, the reasons are that samples z do not have an influence on the sub-SPN representing the class of z_{test} unless z and z_{test} are from the same class, and sub-SPNs which do not represent the class of z_{test} do not contribute to the loss $L^{\text{JLL}}(z_{\text{test}}, \theta)$ if z fed to S is of z_{test}'s class.

However, $L_{\text{train}}^{\text{CLL}}$ can overcome this problem because it incorporates the marginal probability $S(x, 1)$. More precisely, $S(x, 1)$ is a weighted sum of all sub-SPN's values, i.e.

$$S(x,1) \overset{S \text{ valid}}{=} \sum_{y \in Y} S(x,y) \overset{\text{Def. 2.1.1}}{=} \sum_{y \in Y} \sum_{i \in Y} w_{ri} \cdot S_i(x,y) = \sum_{y \in Y} w_{ry} \cdot S_y(x,y).$$

Therefore, each sub-SPN contributes to the loss, no matter what label z_{test} does have. It follows that also samples of other classes are expected to have heterogeneous influence values providing us a different insight into the influence.

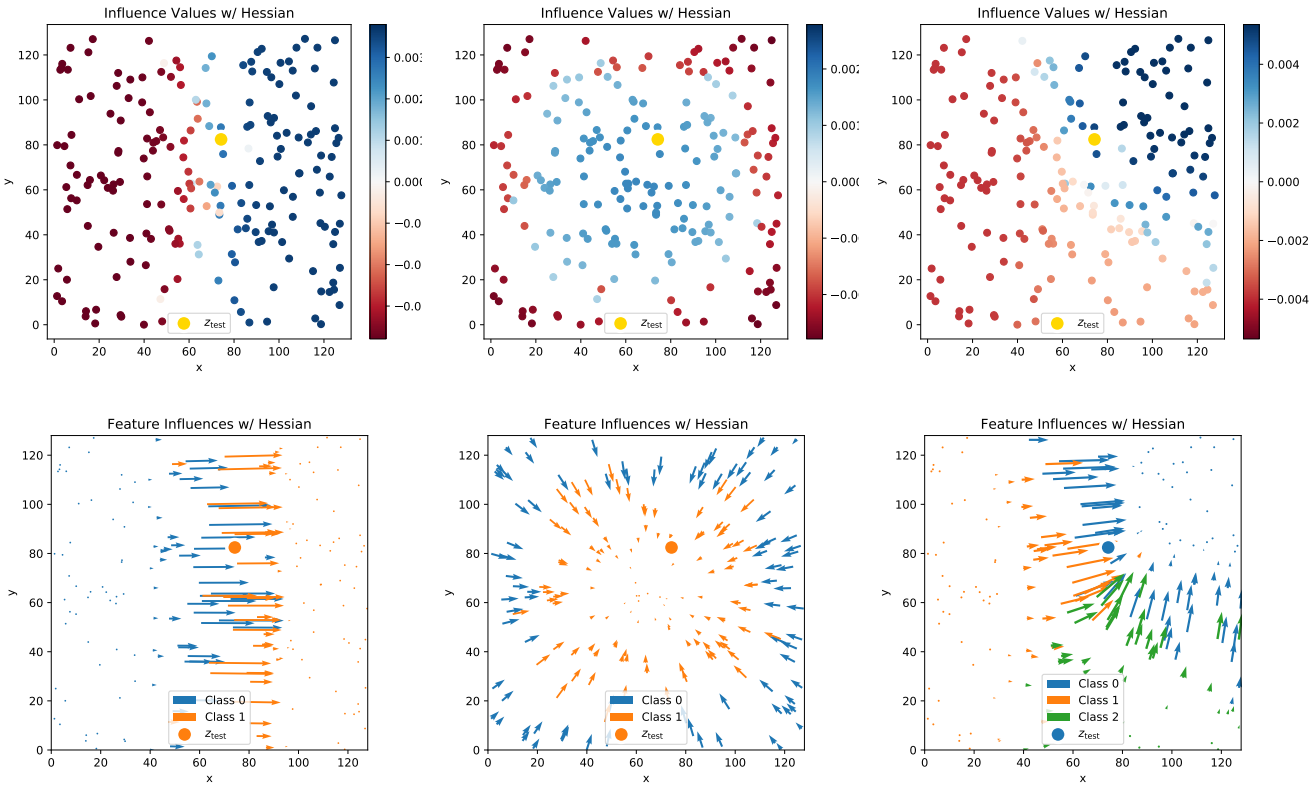


Figure 4.10: Influence for the 2D classification problems using the conditional log-likelihood (CLL).

Figure 4.10 shows the influences regarding $L_{\text{train}}^{\text{CLL}}$. Indeed, as expected, (features of) samples of other classes now have a more heterogeneous influence. It can be observed that feature influences are especially high in the region close to the decision boundary and that their vectors are normal to the decision boundary, reflecting the discriminating manner of the CLL. This is especially helpful to identify features relevant for classification.

Back to MNIST. In the next step, we want to apply IFs to an SPN S trained on the MNIST digit data set in order to see how $L_{\text{train}}^{\text{JLL}}$ and $L_{\text{train}}^{\text{CLL}}$ behave. To this end, a compact-sized SPN with five layers was trained on a data set of 60,000 images with a reduced image resolution of 8×8. Test set accuracy is 58.0%, thus, acceptable for meaningful results. We arbitrarily choose a "1" as the test sample z_{test} and look at the influences of the train samples depicted in Figure 4.11. Before looking at the influence plots, we state two expectations:

First, similar to the outcomes of $I_{\text{pert,loss}}$ in Figures 4.7, 4.8 and 4.9, one expects that, regarding the loss $L_{\text{train}}^{\text{JLL}}$, the feature influences of images other than "1" are zero. Regarding $L_{\text{train}}^{\text{CLL}}$, according to Figure 4.10, feature influences also of samples other than "1" are expected to have values unequal to zero.

Second, handwritten images can be distinguished best by looking at their characteristic, typographical shape. Following this assumption, the feature influences (i.e. the contribution of pixels to the classification of z_{test}) are expected to be especially high in regions where striking typographical characteristics of

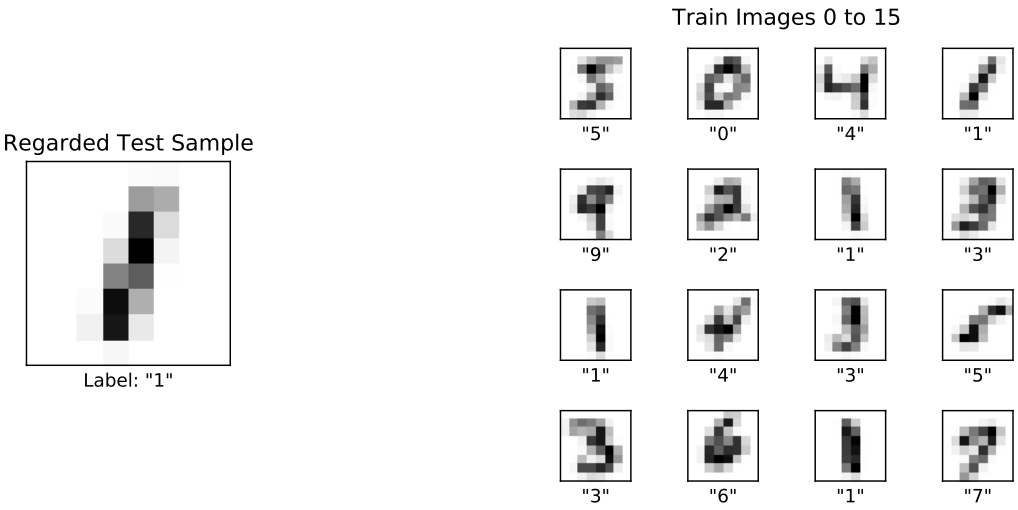


Figure 4.11: The test sample z_{test} on the left and the train samples which are going to be investigated on the right.

the particular digit image are located. As an example, pixels covered by the horizontal line of a "7" are expected to have a strong influence on the classification of the test sample "1". Imagine, the horizontal line of the "7" would be omitted, then it would look like a "1", but it still would have the label "7". This would then be a malicious sample, bad for classification of z_{test} which is a "1".

Figure 4.12: Feature influences returned from $I_{\text{pert,loss}}$ of the train samples from Figure 4.11. Left: feature influences regarding $L^{\text{JLL}}_{\text{train}}$, right: feature influences regarding $L^{\text{CLL}}_{\text{train}}$. Blue pixels indicate that increasing the corresponding feature (= making the pixel more black) has a positive influence, i.e. decreases loss on z_{test} (red pixels indicate the opposite). Gray regions indicate weak influence. For a better visualization, the color map is adjusted to the min./max. influence value for each train sample individually.

The feature influences regarding both, $L^{\text{JLL}}_{\text{train}}$ and $L^{\text{CLL}}_{\text{train}}$, returned by $I_{\text{pert,loss}}$ are visualized as heatmaps in Figure 4.12. Our first expectation is met, since images which have not the label "1" have influence equal

to zero regarding $L^{\text{JLL}}_{\text{train}}$. In contrast to that, feature influences regarding $L^{\text{CLL}}_{\text{train}}$ are also recognizable for samples from classes other than "1". Thus, the corresponding visualization provides much more information which can be used for interpretation compared to the visualization using $L^{\text{JLL}}_{\text{train}}$.

Nevertheless, as we can see, the second expectation is not met. Instead, mainly pixels in the corners or at the edges of the images are influential. Interpreting this plot would result in the statement that the corners and edges are especially relevant for classification of z_{test}. Since the corners and edges are mostly white, they actually should not be relevant for classification. As a result, an interpretation of this outcome would be that S is not right for the right reasons.

This outcome can have two reasons: First, remember that S has only a test accuracy of 58.0%, hence it may not have learned the true concepts of digit classification sufficiently. Second, regarding $L^{\text{CLL}}_{\text{train}}$ practically each feature of each instance has an influence unequal to zero, probably leading to a somewhat "noisy" influence, meaning that highly influential (noisy) corners and edges (which actually are expected to have a low influence) drown out the influence of pixels located in the center of the image. We will have a short look on that phenomena in the next section.

To sum up, when considering feature influences from $I_{\text{pert,loss}}$, the selection of the loss function impacts the explanatory expressiveness significantly. In particular, we saw that $I_{\text{pert,loss}}$ returns more valuable influences when using $L^{\text{CLL}}_{\text{train}}$ instead of $L^{\text{JLL}}_{\text{train}}$.

4.5 Examination of "Influence Noise"

In the previous section, the existence of a "noisy influence" was assumed, stating that in some cases features may outweigh other, actually more relevant features. This section deals with the examination of that assumption.

As one can view in the left plot of Figure 4.10, samples far away from the decision boundary have quite weak feature influences, visualized as dots. If the existence of "influence noise" is true, then in these low-influential regions the influence in y-direction may drown out influence in x-direction for a significant number of samples.

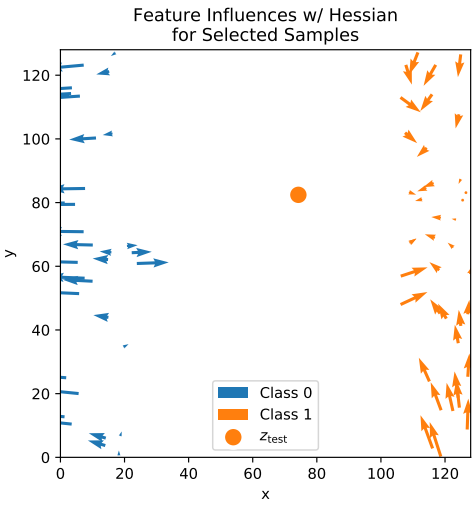


Figure 4.13: Same feature influences as in Figure 4.10, but only for samples far away from decision boundary. IF values were scaled up linearly for better visualization.

Figure 4.13 presents several samples of the low-influential region. It can be viewed that several samples have influence in y-direction which is much higher than influence in x-direction, though the y-coordinate actually is not relevant for the discrimination of this linearly separable data set.

Recall the three different parameter types from Section 4.2 which contribute to the feature influence regarding L^{JLL}. Since the individual loss L^{JLL} is part of L^{CLL}, we can transfer the observations from Section 4.2 to L^{CLL} to some extent. There, we have seen that there may be feature influence vectors

pointing not normal to the decision boundary (cf. Figure 4.7), which was a result of the combined influence contribution of the different parameter types. This could be the source for the observed noise.

In short, regions in feature space with generally low influence may suffer from influence noise, meaning that features which are actually not relevant for classification can have higher influence than other features.

4.6 Evaluating Samples with Strong or Weak Influence

Which MNIST images are most helpful, most malicious or largely uninvolved in prediction of z_{test}, respectively? This section gives a short glimpse of strong or weak influential MNIST digit images according to $L_{\text{train}}^{\text{CLL}}$ and finishes with a hypothetical interpretation of the observed influences.

Recall that the IF $I_{\text{up,loss}}(z, z_{\text{test}})$ returns a real value which tells us how upweighting z affects loss on z_{test}. High (positive) influence indicates loss reduction, i.e. z is helpful for prediction of z_{test} (conversely for negative values). Therefore, the question asked in the beginning can be answered by looking at the train samples having the highest, lowest (most negative) or lowest absolute influence, respectively.

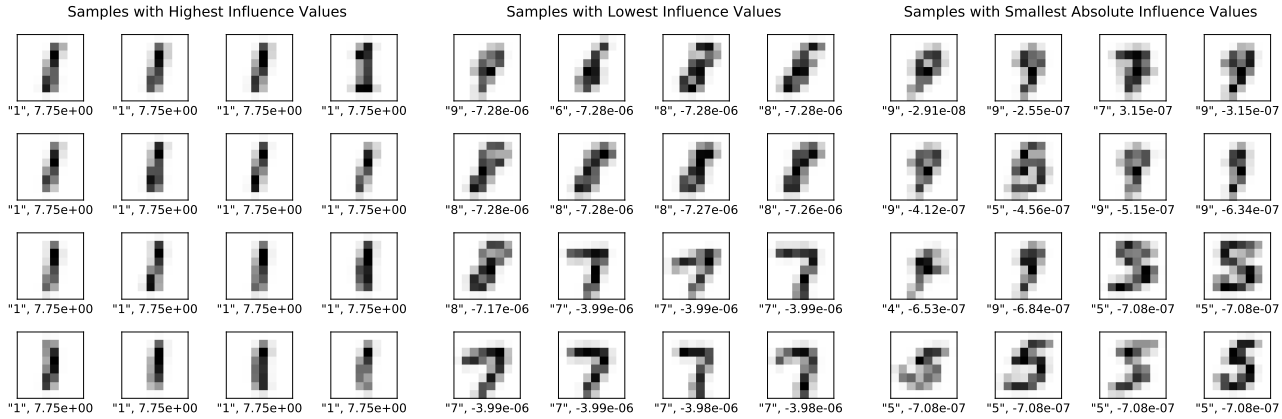


Figure 4.14: MNIST images with (left) highest influence, (middle) lowest (most negative) influence and (right) lowest absolute influence. Numbers indicate labels and influence values.

Again, we choose z_{test} to be "1" as depicted in Figure 4.11. Then, we compute influence of 1,000 randomly selected train samples and sort them by their influence accordingly. Figure 4.14 shows the most influential samples in the left two plots and samples with very weak influences in the right plot. The Hessian was used for a more balanced approximation.

The following observations are remarkable: Only samples labeled with "1" count to the 16 most positive influential samples. Furthermore, the samples of the left plot have a quite uniform influence and their influence is orders of magnitudes stronger compared to the samples in the middle plot. The samples in the middle plot also have partly uniform influence though labels differ. Moreover, the negative influence seems to correlate with similarity of z to z_{test} to some extent. Especially, the second digit in the middle plot which is actually a "6" is almost indistinguishable from a "1".

Translating these observations to interpretations: Samples labeled as "1" seem to be the most helpful samples for discrimination of "1" from other digits. In contrast to that, images which look especially similar to our test sample are malicious for prediction of "1". However, maliciousness is very small compared to helpfulness.

Again, remember that the observations made in this section only belong to the very specific SPN trained on MNIST which represents only mixtures of univariate Gaussians and has only moderate accuracy.

5 Future Work

This presented thesis is only the tip of the iceberg regarding the application of IFs to SPNs. Therefore, this chapter is added to provide some ideas and approaches for future SPN explanation projects which also can benefit from the contribution of this thesis.

To begin with, this thesis only covers SPNs using just univariate Gaussian and categorical nodes as leaves. Indeed, many natural processes to follow a Gaussian principle, but there are also many other types of continuous distributions such as uniform, Poisson, Beta, Gamma, etc. The parameters used to describe these distributions have a quite different contribution to the loss compared to means and variances from Gaussians. Therefore, influence plots are expected to have interesting new shapes.

Moreover, the used MNIST data set contained integer pixel intensities, ranging from 0 to 255. It would be interesting to see if and how IF values change when the pixel intensities are discretized to 0 and 1. Alternatively, instead of discretizing, it would also be interesting to see how normalization of pixel intensities affects influence outcomes.

Because only images with low resolution and small SPNs were investigated with IFs, one may train a deep SPN with $> 95\%$ test set accuracy on 28×28 MNIST images. The influence results are then expected to be more accurate and more meaningful. Besides, it can also be interesting to test SPNs on a different train set such as the toy color set developed by Ross et al. [8] It contains images with many pixels having one of four colors, but just a few pixels determine class of that image.

Consider the image completion problem from Section 2.5.3. One could construct a loss function L assessing the completed image regarding the ground truth. This serves as an anchor point for applying $I_{\text{up,loss}}$ and $I_{\text{pert,loss}}$ in order to investigate influence of (pixels of) images on the loss of a test sample. This can be helpful in the case where a good test accuracy does not suffice for decent image completion.

Another very exciting idea is to measure influence at *hidden nodes*, i.e. nodes inside of the SPN instead of leaves. Selected hidden nodes are considered as features for which $I_{\text{pert,loss}}$ delivers influences. Because influence of whole mixtures of distributions can then be expressed by a single value, this method can be somewhat considered as higher-abstracted influence investigation. As an example, compared to the univariate Gaussian leaves used for 2D classification in Section 2.5.1 where x- and y-axis are approximated separately, this method allows to gain influence values which are not restricted to the very special, univariate representation.

In order to check the robustness of SPNs, one could perform train set attacks by means of $I_{\text{pert,loss}}$. As demonstrated by Koh et al. [7], perturbing a train sample into the opposite direction of its feature influences maximally increases loss on the regarded test sample.

Moreover, through summing up the (feature) influences over each test sample, one could find samples which are helpful or malicious for prediction of the entire test set useful for data cleaning. In this context, experiments trying to figure out poisoned samples from train set attacks would be especially interesting.

Finally, a novel approach called *explanatory interactive learning* was proposed recently by Teso and Kersting. [17] Here, explanations are incorporated into an user-interactive learning process of the model where the user provides feedback not only for a prediction but also for the explanation. When training SPNs interactively, IFs could serve as an explanatory method in order to guide SPN learning.

6 Conclusion

We internalized the fundamental concepts of sum-product networks (SPNs) and saw that SPNs are powerful, tractable graphical models for probabilistic modeling. The basic functionality of SPNs with Gaussian leaves was demonstrated by means of classification and regression tasks. Moreover, we learned the basics of influence functions (IFs), an explanatory tool for making black-box models more understandable.

Through the demonstrative application of IFs to SPNs, some important aspects of the influence investigation of SPNs were evaluated. In particular, we observed that the different SPN parameters contribute quite differently to the influence, possibly outweighing the contribution of other parameters entirely. Moreover, single features may be able to outweigh influences of other features systematically.

Furthermore, we concluded that the selection of a proper loss function is crucial for the explanatory expressiveness of the IF results. For instance, we observed that influence values are quite homogeneous, and many train samples have feature influences equal to zero when choosing the negative joint log-likelihood (JLL) as the loss. Instead of the JLL, influences regarding the negative conditional log-likelihood (CLL) can be more expressive.

Besides, an effect of the Hessian matrix, which is contained in the IFs, could be observed in situations with many parameters and distributions. In detail, the Hessian provides second order information helping the IFs to gain more balanced influence results. Nevertheless, there are also situations where the Hessian has no remarkable effect.

Finally, we also saw that regions in feature space with low feature influence might suffer from influence noise. This means that features which are actually not responsible for classification have influence stronger than other, more relevant features.

List of Figures

Bibliography

[1] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, p. 115, 2017.

[2] M. T. Ribeiro, S. Singh, and C. 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*, pp. 1135–1144, ACM, 2016.

[3] S. G. Finlayson, H. W. Chung, I. S. Kohane, and A. L. Beam, "Adversarial attacks against medical deep learning systems," *arXiv preprint arXiv:1804.05296*, 2018.

[4] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, p. 15, 2012.

[5] H. Poon and P. M. Domingos, "Sum-product networks: A new deep architecture," *CoRR*, vol. abs/1202.3732, 2012.

[6] R. D. Cook and S. Weisberg, "Characterizations of an empirical influence function for detecting influential cases in regression," *Technometrics*, vol. 22, no. 4, pp. 495–508, 1980.

[7] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70, pp. 1885–1894, PMLR, Aug. 2017.

[8] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, "Right for the right reasons: Training differentiable models by constraining their explanations," *CoRR*, vol. abs/1703.03717, 2017.

[9] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[10] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM computing surveys (CSUR)*, vol. 51, no. 5, p. 93, 2018.

[11] C. Molnar, "Interpretable machine learning." `https://christophm.github.io/interpretable-ml-book/`. [accessed July 27th, 2019].

[12] A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito, and K. Kersting, "Mixed sum-product networks: A deep architecture for hybrid domains," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[13] R. Gens and D. Pedro, "Learning the structure of sum-product networks," in *International conference on machine learning*, pp. 873–880, 2013.

[14] R. Gens and P. Domingos, "Discriminative learning of sum-product networks," in *Advances in Neural Information Processing Systems*, pp. 3239–3247, 2012.

[15] A. Rashwan, P. Poupart, and C. Zhitang, "Discriminative training of sum-product networks by extended baum-welch," in *Proceedings of the Ninth International Conference on Probabilistic Graphical Models* (V. Kratochvíl and M. Studený, eds.), vol. 72 of *Proceedings of Machine Learning Research*, (Prague, Czech Republic), pp. 356–367, PMLR, 11–14 Sep 2018.

[16] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization for machine learning in linear time," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4148–4187, 2017.

[17] S. Teso and K. Kersting, "Explanatory interactive machine learning," *Association for the Advancement of Artificial Intelligence (AAAI)*, 2019.

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Mark Rothermel, die vorliegende Bachelor-Thesis ohne gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

English translation for information purposes only:

Thesis Statement pursuant to § 23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Mark Rothermel, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB identical in content.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum/Date Unterschrift/Signature