# Masterarbeit

Lukas Schulth
lukas.schulth@uni.kn

27. April 2021

**Zusammenfassung**

abstract

***Keywords***— one, two, three, four

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings

1 Verfügbare Schichten und Aktivierungsfunktionen . . . . . 6
2 Implementierte Regeln fhvilshoj . . . . . 7
3 Kleines Netzwerk . . . . . 9

# Inhaltsverzeichnis

**1 Einführung** **5**
1.1 Neuronale Netzwerke . . . . . 5
1.1.1 CNNS . . . . . 5
1.2 Poisoning-Angriffe . . . . . 5

**2 Erklärbare KI** **5**
2.1 Lokale Methoden . . . . . 5
2.2 Globale Methoden . . . . . 5

**3 Layer-wise Relevance Propagation** **5**
3.1 Idee . . . . . 5
3.2 Deep Taylor Decomposition . . . . . 5
3.3 Verschiedene Verfahren . . . . . 5
3.4 LRP als DTD . . . . . 5
3.5 Eigenschaften . . . . . 6
3.6 Implementierung . . . . . 6
3.6.1 Tensorflow . . . . . 6
3.6.2 pytorch . . . . . 6

**4 Detektion von Poisoning-Angriffen basierend auf LRP** **7**
4.1 Idee . . . . . 7
4.2 Verwendete Distanzen . . . . . 8
4.2.1 Euklidische Distanz . . . . . 8
4.2.2 Gromov-Wasserstein-Distanz . . . . . 8
4.3 Anwendung auf unterschiedliche Poisoning-Angriffe . . . . . 8
4.3.1 Standard Poisoning-Angriffe . . . . . 8
4.3.2 Label-konsistente Poisoning-Angriffe . . . . . 8

**5 Vergleich mit anderen Verfahren** **8**
5.1 Activation Clustering . . . . . 8
5.2 Räumliche Transformationen . . . . . 8

**6 Weitere mögliche Schritte** **8**

**A Verwendete Netzwerke** **9**
A.1 Net . . . . . 9

**B Parameter für Training und Einlesen der Daten** **9**

**C Datensätze** **9**

**D Programmcode** **9**

# 1 Einführung

A Complete List of All (arXiv) Adversarial Example Papers [1]
In sicherheitskritischen Anwendungsgebieten ist die Interpretation einer Entscheidung genauso wichtig wie die Entscheidung selbst[1].

## 1.1 Neuronale Netzwerke

### 1.1.1 CNNS

Idee, Abstraktion, high level, low level features, bekannte Netzwerke

## 1.2 Poisoning-Angriffe

# 2 Erklärbare KI

## 2.1 Lokale Methoden

## 2.2 Globale Methoden

# 3 Layer-wise Relevance Propagation

## 3.1 Idee

In [4] wird die Layer-wise Relevance Propagation erstmalig vorgestellt. Zudem wird eine Taylor Zerlegung präsentiert, die eine Approximation der LRP darstellt.

## 3.2 Deep Taylor Decomposition

Laut [3] ist die in [4] vorgestellte Layer-wise Relevance Propagation eher heuristisch. In diesem Paper wird nun eine solide theoretische Grundlage geliefert.

LRP in verschiedenen Anwendungsgebieten [9], 10.2. In diesem Paper:LRP-0 schlechter als LRP-$\varepsilon$ schlechter alsLRP-$\gamma$ schlechter als Composite-LRP.

## 3.3 Verschiedene Verfahren

- LRP-0
- LRP-$\varepsilon$
- LRP-$\gamma$

## 3.4 LRP als DTD

siehe [8], Kapitel 10.

[1] `https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html`

## 3.5 Eigenschaften

- Numerische Stabilität
- Konsistenz (mit Linearer Abbildung)
- Erhaltung der Relevanz

## 3.6 Implementierung

### 3.6.1 Tensorflow

### 3.6.2 pytorch

**Allgemeines Tutorial**:[2]
pytorch-LRP für VGG16 wird vorgestellt.

**GiorgioML**[3]:
Alternative pytorch-Implementierung basierend auf Tensorflow paper.

**moboehle**[4]:

Unterstützte Netzwerkschickten[5]:

```
torch.nn.BatchNorm1d,
torch.nn.BatchNorm2d
torch.nn.BatchNorm3d,
torch.nn.ReLU,
torch.nn.ELU,
Flatten,
torch.nn.Dropout,
torch.nn.Dropout2d,
torch.nn.Dropout3d,
torch.nn.Softmax,
torch.nn.LogSoftmax,
torch.nn.Sigmoid


```

Listing 1: Verfügbare Schichten und Aktivierungsfunktionen

**fhvilshoj**[6]:

LRP für linear und Convolutional layers

- Die Klassen
  torch.nn.Sequential, torch.nn.Linear und torch.nn.Conv2d werden erweitert, um autograd für die Berechnung der Relevanzen zu berechnen.
- Ausgabe der Relevanzen von Zwischenschichten ist möglich

[2] https://git.tu-berlin.de/gmontavon/lrp-tutorial
[3] https://giorgiomorales.github.io/Layer-wise-Relevance-Propagation-in-Pytorch/
[4] https://github.com/moboehle/Pytorch-LRP
[5] https://github.com/moboehle/Pytorch-LRP/blob/master/inverter_util.py
[6] https://github.com/fhvilshoj/TorchLRP

- : Implementierte Regeln: epsilon Regeln mit epsilon=1e-1, gamma-regel mit gamma=1e-1. alphabeta-Reagel mit a1b0 und a2b1
- Netz muss hier umgeschrieben werden, sodass die Anwendung des Algorithmus möglich wird.

```
1
2  conv2d = {
3    "gradient":              F.conv2d,
4    "epsilon":               Conv2DEpsilon.apply,
5    "gamma":                 Conv2DGamma.apply,
6    "gamma+epsilon":         Conv2DGammaEpsilon.apply,
7    "alpha1beta0":           Conv2DAlpha1Beta0.apply,
8    "alpha2beta1":           Conv2DAlpha2Beta1.apply,
9    "patternattribution":    Conv2DPatternAttribution.apply,
10   "patternnet":            Conv2DPatternNet.apply,
11 }
12
```

Listing 2: Implementierte Regeln fhvilshoj

**Zennit**:[7] Zennit (Zennit explains neural networks in torch)

- Modell wird mithilfe eines Canonizers so aufbereitet, dass LRP möglich wird
- Backward pass wird modifiziert, um Heatmaps zu erhalten.
- VGG- und ResNet-Beispiel

# 4 Detektion von Poisoning-Angriffen basierend auf LRP

## 4.1 Idee

Die Idee zur Detektion von Poisoning-Angriffen besteht aus den folgenden Schritten:

- Berechnung der Heatmaps mit Hilfe der LRP
- Berechnung einer Distanzmatrix basierend auf $L-$ oder GMW-Distanz
- Spektrale Relevanzanalyse (Bestimmung der verschiedenen Cluster innerhalb einer Klasse)

**Bemerkung:** Anstatt das Clustering nur auf den Heatmaps durchzuführen, könnten die LRP-Ausgaben und/oder Aktivierungen bestimmer NEtzwerkschichten hinzugenommen werden.

[7] `https://github.com/chr5tphr/zennit`

## 4.2 Verwendete Distanzen

### 4.2.1 Euklidische Distanz

### 4.2.2 Gromov-Wasserstein-Distanz

## 4.3 Anwendung auf unterschiedliche Poisoning-Angriffe

### 4.3.1 Standard Poisoning-Angriffe

### 4.3.2 Label-konsistente Poisoning-Angriffe

# 5 Vergleich mit anderen Verfahren

## 5.1 Activation Clustering

## 5.2 Räumliche Transformationen

- ASR ist sehr stark vom Ort des Triggers abhängig.
- Ort des Triggers kann nicht direkt geändert werden.
- Benutze Transformationen(Flipping, Scaling), um den Trigger wirkungslos zu machen.
- Somit kann die ASR während der Inferenz verringert werden. Es lässt sich aber keine Auussage darüber treffen, ob ein Angriff vorliegt

# 6 Weitere mögliche Schritte

- Automatische Platzierung des Auslösers an fest gewählter Position auf dem Verkehrsschild anstatt zufälligem Platzierem in einem Fenster mit vorher festgelegter Größe. In [7] wird Faster-RCNN (F-RCNN) zur Klassifikation des LISA-Datensatzes[8] benutzt. Es ist die Aufgabe, die Verkehrsschilder in die 3 Superklassen Stoppschild, Geschwindigkeitsbegrenzung und Warnschild einzuteilen. Der Datensatz enthält zudem die BoundingBoxen, sodass der Auslöser genauer angebracht werden kann.
- Verbesserte Version der Layer-wise Relevance Propagation
- Untersuchung anderer Verfahren, die die Interpretierbarkeit ermöglichen, beispielsweise: VisualBackProp: efficient visualization of CNNs[9]
- Vergleich mit Cifar-10/Cifar-100 Datensatz[10][11]

[8] http://cvrr.ucsd.edu/LISA/lisa-traffic-sign-dataset.html
[9] https://arxiv.org/abs/1611.05418
[10] https://www.cs.toronto.edu/~kriz/cifar.html
[11] https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

# A Verwendete Netzwerke

## A.1 Net

```
class Net(nn.Module):

  def __init__(self, ):
    super(Net, self).__init__()
    self.size = 64 * 4 * 4
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=12,
  kernel_size=5, padding=2)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    self.conv1_in = nn.InstanceNorm2d(12)
    self.conv2 = nn.Conv2d(in_channels=12, out_channels=32,
  kernel_size=5, padding=2)

    self.conv2_bn = nn.BatchNorm2d(32)

    self.conv3 = nn.Conv2d(in_channels=32, out_channels=64,
  kernel_size=5, padding=2)

    self.fc1 = nn.Linear(self.size, 256)
    self.fc1_bn = nn.BatchNorm1d(256)
    self.fc2 = nn.Linear(256, 128)
    self.fc3 = nn.Linear(128, 43)


  def forward(self, x):
    x = self.pool(F.relu(self.conv1_in(self.conv1(x))))
    x = self.pool(F.relu(self.conv2_bn(self.conv2(x))))
    x = self.pool(F.relu(self.conv3(x)))
    x = x.view(-1, self.size)
    x = F.relu(self.fc1_bn(self.fc1(x)))
    x = F.dropout(x)
    xx = F.relu(self.fc2(x))
    x = F.dropout(xx)
    x = self.fc3(x)

    return x, xx


```

Listing 3: Kleines Netzwerk

# B Parameter für Training und Einlesen der Daten

Die in [2] gewählten Parameter wären ein guter Ausgangspunkt.

# C Datensätze

# D Programmcode

# Literatur

[1] Layer-wise Relevance Propagation for Deep Neural Network Architectures. Alexander Binder, Sebastian Bach, Gregoire Montavon, Klaus-Robert Müller und Wojciech Samek

[2] Finding and Removing Clever Hans: Using Explanation Methods to Debug and Improve Deep Models

[3] Explaining nonlinear classification decisions with deep Taylor decomposition

[4] On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation

[5] S. Bazen, X. Joutard, The Taylor decomposition: a unified generalization of the Oaxaca method to nonlinear models, Technical Report 2013-32, Aix-Marseille University, 2013.

[6] R. Oaxaca, Male-female wage differentials in urban labor markets, Int. Econ. Rev. 14 (3) (1973) 693–709

[7] BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

[8] Montavon G., Binder A., Lapuschkin S., Samek W., Müller KR. (2019) Layer-Wise Relevance Propagation: An Overview. In: Samek W., Montavon G., Vedaldi A., Hansen L., Müller KR. (eds) Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Lecture Notes in Computer Science, vol 11700. Springer, Cham. https://doi.org/10.1007/978-3-030-28954-6_10

[9] Layer-Wise Relevance Propagation: An Overview Grégoire Montavon 1 , Alexander Binder 2 , Sebastian Lapuschkin 3 , Wojciech Samek 3 , and Klaus-Robert Müller 1,4,5