Evaluating Recurrent Neural Network Explanations

Leila Arras[1], **Ahmed Osman**[1], **Klaus-Robert Müller**[2,3,4], **and Wojciech Samek**[1]
[1]Machine Learning Group, Fraunhofer Heinrich Hertz Institute, Berlin, Germany
[2]Machine Learning Group, Technische Universität Berlin, Berlin, Germany
[3]Department of Brain and Cognitive Engineering, Korea University, Seoul, Korea
[4]Max Planck Institute for Informatics, Saarbrücken, Germany
{leila.arras, wojciech.samek}@hhi.fraunhofer.de

Abstract

Recently, several methods have been proposed to explain the predictions of recurrent neural networks (RNNs), in particular of LSTMs. The goal of these methods is to understand the network's decisions by assigning to each input variable, e.g., a word, a *relevance* indicating to which extent it contributed to a particular prediction. In previous works, some of these methods were not yet compared to one another, or were evaluated only qualitatively. We close this gap by systematically and quantitatively comparing these methods in different settings, namely (1) a toy arithmetic task which we use as a sanity check, (2) a five-class sentiment prediction of movie reviews, and besides (3) we explore the usefulness of word relevances to build sentence-level representations. Lastly, using the method that performed best in our experiments, we show how specific linguistic phenomena such as the negation in sentiment analysis reflect in terms of relevance patterns, and how the relevance visualization can help to understand the misclassification of individual samples.

1 Introduction

Recurrent neural networks such as LSTMs (Hochreiter and Schmidhuber, 1997) are a standard building block for understanding and generating text data in NLP. They find usage in pure NLP applications, such as abstractive summarization (Chopra et al., 2016), machine translation (Bahdanau et al., 2015), textual entailment (Rocktäschel et al., 2016); as well as in multimodal tasks involving NLP, such as image captioning (Karpathy and Fei-Fei, 2015), visual question answering (Xu and Saenko, 2016) or lip reading (Chung et al., 2017).

As these models become more and more widespread due to their predictive performance, there is also a need to understand *why* they took a particular decision, i.e., when the input is a sequence of words: *which* words are determinant for the final decision? This information is crucial to unmask "Clever Hans" predictors (Lapuschkin et al., 2019), and to allow for transparency of the decision-making process (EU-GDPR, 2016).

Early works on *explaining* neural network predictions include Baehrens et al. (2010); Zeiler and Fergus (2014); Simonyan et al. (2014); Springenberg et al. (2015); Bach et al. (2015), with several works focusing on explaining the decisions of convolutional neural networks (CNNs) for image recognition. More recently, this topic found a growing interest within NLP, amongst others to explain the decisions of general CNN classifiers (Arras et al., 2017a; Jacovi et al., 2018), and more particularly to explain the predictions of recurrent neural networks (Li et al., 2016, 2017; Arras et al., 2017b; Ding et al., 2017; Murdoch et al., 2018; Poerner et al., 2018).

In this work, we focus on RNN explanation methods that are solely based on a trained neural network model and a single test data point[1]. Thus, methods that use additional information, such as training data statistics, sampling, or are optimization-based (Ribeiro et al., 2016; Lundberg and Lee, 2017; Chen et al., 2018) are out of our scope. Among the methods we consider, we note that the method of Murdoch et al. (2018) was not yet compared against Arras et al. (2017b); Ding et al. (2017); and that the method of Ding et al. (2017) was validated only visually. Moreover, to the best of our knowledge, no recurrent neural network explanation method was tested so far on a toy problem where the *ground truth* rele-

[1]These methods are deterministic, and are essentially based on a *decomposition* of the model's current prediction. Thereby they intend to reflect the *sole* model's "point of view" on the test data point, and hence are not meant to provide an averaged, smoothed or denoised explanation of the prediction by additionally exploiting the data's distribution.

vance value is known.

Therefore our contributions are the following: we evaluate and compare the aforementioned methods, using two different experimental setups, thereby we assess basic properties and differences between the explanation methods. Along-the-way we purposely adapted a simple toy task, to serve as a testbed for recurrent neural networks explanations. Lastly, we explore how word relevances can be used to build sentence-level representations, and demonstrate how the relevance visualization can help to understand the (mis-)classification of selected samples w.r.t. semantic composition.

2 Explaining Recurrent Neural Network Predictions

First, let us settle some notations. We suppose given a trained recurrent neural network based model, which has learned some scalar-valued prediction function $f_c(\cdot)$, for each class c of a classification problem. Further, we denote by $\mathbf{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_T})$ an unseen input data point, where $\boldsymbol{x_t}$ represents the t-th input vector of dimension D, within the input sequence $\mathbf{x}$ of length T. In NLP, the vectors $\boldsymbol{x_t}$ are typically word embeddings, and $\mathbf{x}$ may be a sentence.

Now, we are interested in methods that can explain the network's prediction $f_c(\mathbf{x})$ for the input $\mathbf{x}$, and for a chosen *target* class c, by assigning a scalar relevance value to each input variable or word. This relevance is meant to quantify the variable's or word's importance *for* or *against* a model's prediction *towards* the class c. We denote by R_{x_i} (index i) the relevance of a *single* variable. This means x_i stands for any arbitrary input variable $x_{t,d}$ representing the d-th dimension, $d \in \{1, ..., D\}$, of an input vector $\boldsymbol{x_t}$. Further, we refer to $R_{\boldsymbol{x_t}}$ (index t) to designate the relevance value of an *entire* input vector or word $\boldsymbol{x_t}$. Note that, for most methods, one can obtain a word-level relevance value by simply summing up the relevances over the word embedding dimensions, i.e. $R_{\boldsymbol{x_t}} = \sum_{d \in \{1,...,D\}} R_{x_{t,d}}$.

2.1 Gradient-based explanation

One standard approach to obtain relevances is based on partial derivatives of the prediction function: $R_{x_i} = \left|\frac{\partial f_c}{\partial x_i}(\mathbf{x})\right|$, or $R_{x_i} = \left(\frac{\partial f_c}{\partial x_i}(\mathbf{x})\right)^2$ (Dimopoulos et al., 1995; Gevrey et al., 2003; Simonyan et al., 2014; Li et al., 2016).

In NLP this technique was employed to visualize the relevance of single input variables in RNNs for sentiment classification (Li et al., 2016). We use the latter formulation of relevance and denote it as *Gradient*. With this definition the relevance of an entire word is simply the squared L_2-norm of the prediction function's gradient w.r.t. the word embedding, i.e. $R_{\boldsymbol{x_t}} = \|\nabla_{\boldsymbol{x_t}} \, f_c(\mathbf{x})\|_2^2$.

A slight variation of this approach uses partial derivatives multiplied by the variable's value, i.e. $R_{x_i} = \frac{\partial f_c}{\partial x_i}(\mathbf{x}) \cdot x_i$. Hence, the word relevance is a dot product between prediction function gradient and word embedding: $R_{\boldsymbol{x_t}} = (\nabla_{\boldsymbol{x_t}} \, f_c(\mathbf{x}))^T \boldsymbol{x_t}$. We refer to this variant as *Gradient*$\times$*Input*.

Both variants are general and can be applied to any neural network. They are computationally efficient and require one forward and backward pass through the net.

2.2 Occlusion-based explanation

Another method to assign relevances to single variables, or entire words, is by occluding them in the input, and tracking the difference in the network's prediction w.r.t. a prediction on the original unmodified input (Zeiler and Fergus, 2014; Li et al., 2017). In computer vision the occlusion is performed by replacing an image region with a grey or zero-valued square (Zeiler and Fergus, 2014). In NLP word vectors, or single of its components, are replaced by zero; in the case of recurrent neural networks, the technique was applied to identify important words for sentiment analysis (Li et al., 2017).

Practically, the relevance can be computed in two ways: in terms of prediction function differences, or in the case of a classification problem, using a difference of probabilities, i.e. $R_{x_i} = f_c(\mathbf{x}) - f_c(\mathbf{x}_{|x_i=0})$ or $R_{x_i} = P_c(\mathbf{x}) - P_c(\mathbf{x}_{|x_i=0})$, where $P_c(\cdot) = \frac{\exp f_c(\cdot)}{\sum_k \exp f_k(\cdot)}$. We refer to the former as *Occlusion*$_{\text{f-diff}}$, and to the latter as *Occlusion*$_{\text{P-diff}}$. Both variants can also be used to estimate the relevance of an entire word, in this case the corresponding word embedding is set to zero in the input. This type of explanation is computationally expensive and requires T forward passes through the network to determine one relevance value per word in the input sequence $\mathbf{x}$.

2.3 Layer-wise relevance propagation

A general method to determine input space relevances based on a backward decomposition of the neural network prediction function is layer-wise relevance propagation (LRP) (Bach et al., 2015). It was originally proposed to explain feed-forward neural networks such as convolutional neural networks (Bach et al., 2015; Lapuschkin et al., 2016), and was recently extended to recurrent neural networks (Arras et al., 2017b; Ding et al., 2017; Arjona-Medina et al., 2018).

LRP consists in a standard forward pass, followed by a specific backward pass which is defined for each type of layer of a neural network by dedicated propagation rules. Via this backward pass, each neuron in the network gets assigned a relevance, starting with the output neuron whose relevance is set to the prediction function's value, i.e. to $f_c(\mathbf{x})$. Each LRP propagation rule redistributes iteratively, layer-by-layer, the relevance from higher-layer neurons to lower-layer neurons, and verifies a relevance conservation property[2]. These rules were initially proposed in Bach et al. (2015) and were subsequently justified by Deep Taylor decomposition (Montavon et al., 2017) for deep ReLU nets.

In practice, for a linear layer of the form $z_j = \sum_i z_i w_{ij} + b_j$, and given the relevances of the output neurons R_j, the input neurons' relevances R_i are computed through the following summation: $R_i = \sum_j \frac{z_i \cdot w_{ij}}{z_j + \epsilon \cdot sign(z_j)} \cdot R_j$, where ϵ is a stabilizer (small positive number), this propagation rule is commonly referred as ϵ-LRP or ϵ-rule[3]. Further, for an element-wise non-linear activation layer, the output neurons' relevances are redistributed identically onto the input neurons.

In addition to the above rules, in the case of a multiplicative layer of the form $z_j = z_g \cdot z_s$, Arras et al. (2017b) proposed to redistribute zero relevance to the *gate* (the neuron that is sigmoid activated) i.e. $R_g = 0$, and assign all the relevance to the remaining "signal" neuron (which is usually tanh activated) i.e. $R_s = R_j$. We call this variant *LRP-all*, which stands for "signal-take-all" redistribution. An alternative rule was proposed in Ding et al. (2017); Arjona-Medina et al. (2018), where the output neuron's relevance R_j is redistributed onto the input neurons via: $R_g = \frac{z_g}{z_g + z_s} R_j$ and $R_s = \frac{z_s}{z_g + z_s} R_j$. We refer to this variant as *LRP-prop*, for "proportional" redistribution. We also consider two other variants. The first one uses absolute values instead: $R_g = \frac{|z_g|}{|z_g| + |z_s|} R_j$ and $R_s = \frac{|z_s|}{|z_g| + |z_s|} R_j$, we call it *LRP-abs*. The second uses equal redistribution: $R_g = R_s = 0.5 \cdot R_j$ (Arjona-Medina et al., 2018), we denote it as *LRP-half*. We further add a stabilizing term to the denominator of the *LRP-prop* and *LRP-abs* variants, it has the form $\epsilon \cdot sign(z_g + z_s)$ in the first case, and simply ϵ in the latter.

Since the relevance can be computed in one forward and backward pass, the LRP method is efficient. Besides, it is general as it can be applied to any neural network made of the above layers: it was applied successfully to CNNs, LSTMs, GRUs, and QRNNs (Poerner et al., 2018; Yang et al., 2018)[4].

2.4 Contextual Decomposition

Another method, specific to LSTM networks, is contextual decomposition (*CD*) (Murdoch et al., 2018). It is based on a "linearization" of the activation functions that enables to decompose the forward pass by distinguishing between two contributions: those made by a chosen contiguous subsequence (a word or a phrase) within the input sequence $\mathbf{x}$, and those made by the remaining part of the input. This decomposition results in a final hidden state vector h_T (see the Appendix for a full specification of the LSTM architecture) that can be rewritten as a sum of two vectors: β_T and γ_T, where the former corresponds to the contribution from the "relevant" part of interest, and the latter stems from the "irrelevant" part. When the LSTM is followed by a linear output layer of the form $w_c^T h_T + b_c$ for class c, then the relevance of a given word (or phrase) and for the *target* class c, is given by the dot product: $w_c^T \beta_T$.

The method is computationally expensive as it requires T forward passes through the LSTM to compute one relevance value per word. Although it was recently extended to CNNs (Singh et al., 2019), it is yet not clear how to compute the *CD* relevance in other recurrent architectures, or in networks with multi-modal inputs. See Table 1 for an overview of the explanation methods considered in this work.

[2]Methods based on a similar conservation principle include contribution propagation (Landecker et al., 2013), Deep Taylor decomposition (Montavon et al., 2017), and DeepLIFT (Shrikumar et al., 2017).

[3]Such a rule was employed by previous works with recurrent neural networks (Arras et al., 2017b; Ding et al., 2017; Arjona-Medina et al., 2018), although there exist also other LRP rules for linear layers (see e.g. Montavon et al., 2018)

[4]Note that in the present work we apply LRP to *standard* LSTMs, though Arjona-Medina et al. (2018) showed that some LRP rules for products can benefit from simultaneously adapting the LSTM architecture.

Method	Relevance Formulation	Redistributed Quantity ($\sum_i R_{x_i}$)	Complexity
Gradient	$R_{x_i} = \big(\frac{\partial f_c}{\partial x_i}(\mathbf{x})\big)^2$	$\lVert \nabla_{\mathbf{x}} f_c(\mathbf{x}) \rVert_2^2$	$\Theta(2 \cdot T)$
Gradient×Input	$R_{x_i} = \frac{\partial f_c}{\partial x_i}(\mathbf{x}) \cdot x_i$	$(\nabla_{\mathbf{x}} f_c(\mathbf{x}))^T \mathbf{x}$	$\Theta(2 \cdot T)$
Occlusion	$R_{x_i} = f_c(\mathbf{x}) - f_c(\mathbf{x}_{\vert x_i=0})$	-	$\Theta(T^2)$
LRP	"backward decomposition of the neurons' relevance"	$f_c(\mathbf{x})$	$\Theta(2 \cdot T)$
CD	"linearization of the activation functions"	$f_c(\mathbf{x})$	$\Theta(T^2)$

Table 1: Overview of the explanation methods studied in this paper. T denotes the length of the input sequence.

2.5 Methods not considered

Other methods to compute relevances include Integrated Gradients (Sundararajan et al., 2017). It was previously compared against *CD* in Murdoch et al. (2018), and against the LRP variant of Arras et al. (2017b) in Poerner et al. (2018), where in both cases it was shown to deliver inferior results. Another method is DeepLIFT (Shrikumar et al., 2017), however according to its authors DeepLIFT was not designed for multiplicative connections, and its extension to recurrent networks remains an open question[5]. For a comparative study of explanation methods with a main focus on feed-forward nets, see Ancona et al. (2018)[6]. For a broad evaluation of explanations, including several recurrent architectures, we refer to Poerner et al. (2018). Note that the latter didn't include the *CD* method of Murdoch et al. (2018), and the LRP variant of Ding et al. (2017), which we compare here.

3 Evaluating Explanations

3.1 Previous work

How to generally and objectively evaluate explanations, without resorting to *ad-hoc* procedures that are domain and task specific, is still active research (Alishahi et al., 2019).

In computer vision, it has become common practice to conduct a perturbation analysis (Bach et al., 2015; Samek et al., 2017; Shrikumar et al., 2017; Lundberg and Lee, 2017; Ancona et al., 2018; Chen et al., 2018; Morcos et al., 2018): hereby a few pixels in an image are perturbated (e.g. set to zero or blurred) according to their relevance (most relevant or least relevant pixels are perturbated first), and then the impact on the network's prediction is measured. The higher the impact, the more accurate is the relevance.

Other studies explored in which way relevances are consistent or helpful w.r.t. human judgment (Ribeiro et al., 2016; Lundberg and Lee, 2017; Nguyen, 2018). Some other works relied solely on the visual inspection of a few selected relevance heatmaps (Li et al., 2016; Sundararajan et al., 2017; Ding et al., 2017).

In NLP, Murdoch et al. (2018) proposed to measure the correlation between word relevances obtained on an LSTM, and the word importance scores obtained from a linear Bag-of-Words. However, the latter ignores the word ordering and context, which the LSTM can take into account, hence this type of evaluation is not adequate[7]. Other evaluations in NLP are task specific. For example Poerner et al. (2018) use the subject-verb agreement task proposed by Linzen et al. (2016), where the goal is to predict a verb's number, and use the relevances to verify that the most relevant word is indeed the correct subject (or a noun with the predicted number).

Other studies include an evaluation on a synthetic task: Yang et al. (2018) generated random sequences of MNIST digits and train an LSTM to predict if a sequence contains zero digits or not, and verify that the explanation indeed assigns a high relevance to the zero digits' positions.

A further approach uses randomization of the model weights and data as sanity checks (Adebayo et al., 2018) to verify that the explanations are indeed dependent on the model and data. Lastly, some evaluations are "indirect" and use relevances to solve a broader task, e.g. to build document-

[5]Though Poerner et al. (2018) showed that, when using only the Rescale rule of DeepLIFT, and combining it with the product rule proposed in Arras et al. (2017b), then the resulting explanations perform on-par with the LRP method of Arras et al. (2017b)

[6]Note that in order to redistribute the relevance through multiplicative layers, Ancona et al. (2018) simply relied on standard gradient backpropagation. Such a redistribution scheme is not appropriate for methods such as LRP, since it violates the relevance conservation property, hence their results for recurrent nets are not conclusive.

[7]The same way Murdoch et al. (2018) try to "match" phrase-level relevances with n-gram linear classifier scores or human annotated phrases, but again this might be misleading, since the latter scores or annotations ignore the whole sentence context.

level representations (Arras et al., 2017a), or to redistribute predicted rewards in reinforcement learning (Arjona-Medina et al., 2018).

3.2 Toy Arithmetic Task

As a first evaluation, we ask the following question: if we add two numbers within an input sequence, can we recover from the relevance the true input values? This amounts to consider the adding problem (Hochreiter and Schmidhuber, 1996), which is typically used to test the long-range capabilities of recurrent models (Martens and Sutskever, 2011; Le et al., 2015). We use it here to test the faithfulness of explanations. To that end, we define a setup similar to (Hochreiter and Schmidhuber, 1996), but without *explicit* markers to identify the sequence start and end, and the two numbers to be added. Our idea is that, in general, it is not clear what the ground truth relevance for a marker should be, and we want only the relevant numbers in the input sequence to get a non-zero relevance. Hence, we represent the input sequence $\mathbf{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_T})$ of length T, with two-dimensional input vectors as:

$$\left(\begin{smallmatrix} 0 & 0 & 0 & n_a & 0 & 0 & 0 & n_b & 0 & 0 & 0 \\ n_1 & \dots & n_{a-1} & 0 & n_{a+1} & \dots & n_{b-1} & 0 & n_{b+1} & \dots & n_T \end{smallmatrix}\right)$$

where the non-zero entries n_t are random real numbers, and the two relevant positions a and b are sampled uniformly among $\{1, ..., T\}$ with $a < b$.

More specifically, we consider two tasks that can be solved by an LSTM model with a hidden layer of size *one* (followed by a linear output layer with no bias[8]): the *addition* of signed numbers (n_t is sampled uniformly from $[-1, -0.5] \cup [0.5, 1.0]$) and the *subtraction* of positive numbers (n_t is sampled uniformly from $[0.5, 1.0]$[9]). In the former case the target output y is $n_a + n_b$, in the latter it is $n_a - n_b$. During training we minimize Mean Squared Error (MSE). To ensure that train/val/test sets do not overlap we use 10000 sequences with lengths $T \in \{4, ..., 10\}$ for training, 2500 sequences with $T \in \{11, 12\}$ for validation, and 2500 sequences with $T \in \{13, 14\}$ as test set. For each task we train 50 LSTMs with a validation MSE $< 10^{-4}$, the resulting test MSE is $< 10^{-4}$.

Then, given the model's predicted output y_{pred}, we compute one relevance value $R_{\boldsymbol{x_t}}$ per input vector $\boldsymbol{x_t}$ (for the occlusion method we compute only *Occlusion*$_{\text{f-diff}}$ since the task is a regression; we also don't report *Gradient* results since it performs poorly). Finally, we track the correlation between the relevances and the two input numbers n_a and n_b. We also track the portion of relevance assigned to the relevant time steps, compared to the relevance for all time steps. Lastly, we calculate the "MSE" between the relevances for the relevant positions a and b and the model's output. Our results are compiled in Table 2.

Interestingly, we note that on the addition task several methods perform well and produce a relevance that correlates perfectly with the input numbers: *Gradient*×*Input*, *Occlusion*, *LRP-all* and *CD* (they are highlighted in bold in the Table). They further assign all the relevance to the time steps a and b and almost no relevance to the rest of the input; and present a relevance that sum up to the predicted output. However, on subtraction, only *Gradient*×*Input* and *LRP-all* present a correlation of near one with n_a, and of near minus one with n_b. Likewise these methods assign only relevance to the relevant positions, and redistribute the predicted output entirely onto these positions.

The main difference between our addition and subtraction toy tasks, is that the former requires only summing up the first dimension of the input vectors and can be solved by a Bag-of-Words approach, while our subtraction task is truly *sequential* and requires the LSTM model to remember which number arrived first, and which number arrived second.

Since in NLP several applications require the word *ordering* to be taken into account to accurately capture a sentence's meaning (e.g. in sentiment analysis or in machine translation) our experiment, albeit being an abstract numerical task, is pertinent and can serve as a first sanity check to check whether the relevance can reflect the *ordering* and the *value* of the input vectors.

3.3 5-Class Sentiment Prediction

As a sentiment analysis dataset, we use the Stanford Sentiment Treebank (Socher et al., 2013) which contains labels (very negative $--$, negative $-$, neutral 0, positive $+$, very positive $++$) for resp. 8544/1101/2210 train/val/test sentences and their constituent phrases. As a classifier we employ the bidirectional LSTM from Li et al.

[8]We omit the final layer bias since all considered explanation methods ignore it in the relevance computation, and we want to explain the "full" prediction function's value.

[9]We avoid small numbers by using 0.5 as a minimum magnitude only to simplify learning, since otherwise this would encourage the model weights to grow rapidly.

	$\rho(R_{\boldsymbol{x_a}}, n_a)$ (in %)	$\rho(R_{\boldsymbol{x_b}}, n_b)$ (in %)	$E[\frac{\lvert R_{\boldsymbol{x_a}}\rvert + \lvert R_{\boldsymbol{x_b}}\rvert}{\sum_t \lvert R_{\boldsymbol{x_t}}\rvert}]$ (in %)	$E[((R_{\boldsymbol{x_a}} + R_{\boldsymbol{x_b}}) - y_{\text{pred}})^2]$ ("MSE")
Toy Task Addition				
Gradient×Input	**99.960** (0.017)	**99.954** (0.019)	**99.68** (0.53)	$\mathbf{24.10^{-4}}$ (8.10^{-4})
Occlusion	**99.990** (0.004)	**99.990** (0.004)	**99.82** (0.27)	$\mathbf{20.10^{-5}}$ (8.10^{-5})
LRP-prop	0.785 (3.619)	10.111 (12.362)	18.14 (4.23)	1.3 (1.0)
LRP-abs	7.002 (6.224)	12.410 (17.440)	18.01 (4.48)	1.3 (1.0)
LRP-half	29.035 (9.478)	51.460 (19.939)	54.09 (17.53)	1.1 (0.3)
LRP-all	**99.995** (0.002)	**99.995** (0.002)	**99.95** (0.05)	$\mathbf{2.10^{-6}}$ (4.10^{-6})
CD	**99.997** (0.002)	**99.997** (0.002)	**99.92** (0.06)	$\mathbf{4.10^{-5}}$ (12.10^{-5})
Toy Task Subtraction				
Gradient×Input	**97.9** (1.6)	**-98.8** (0.6)	**98.3** (0.6)	$\mathbf{6.10^{-4}}$ (4.10^{-4})
Occlusion	99.0 (2.0)	-69.0 (19.1)	25.4 (16.8)	0.05 (0.08)
LRP-prop	3.1 (4.8)	-8.4 (18.9)	15.0 (2.4)	0.04 (0.02)
LRP-abs	1.2 (7.6)	-23.0 (11.1)	15.1 (1.6)	0.04 (0.002)
LRP-half	7.7 (15.3)	-28.9 (6.4)	42.3 (8.3)	0.05 (0.06)
LRP-all	**98.5** (3.5)	**-99.3** (1.3)	**99.3** (0.6)	$\mathbf{8.10^{-4}}$ (25.10^{-4})
CD	-25.9 (39.1)	-50.0 (29.2)	49.4 (26.1)	0.05 (0.05)

Table 2: Statistics of the relevance w.r.t. the input numbers n_a and n_b and the predicted output y_{pred}, on toy arithmetic tasks. ρ denotes the correlation and E the mean. Each statistic is computed over 2500 test data points. Reported are the mean (and standard deviation in parenthesis) over 50 trained LSTM models.

(2016)[10], which achieves 82.9% binary performance and 46.3% five-class accuracy on full sentences.

Perturbation Experiment. In order to evaluate the *selectivity* of word relevances, we perform a perturbation experiment aka "pixel-flipping" in computer vision (Bach et al., 2015; Samek et al., 2017), i.e. we remove words from the input sentences according to their relevance, and track the impact on the classification performance. A similar experiment has been conducted in previous NLP studies (Arras et al., 2017a; Nguyen, 2018; Chen et al., 2018); and besides, such type of experiment can be seen as the input space pendant of *ablation*, which is commonly used to identify "relevant" intermediate neurons, e.g. in (Lakretz et al., 2019). For our experiment we retain test sentences with a length ≥ 10 words (i.e. 1849 sentences), and remove 1, 2, and 3 words per sentence[11], according to their relevance obtained on the original sentence with the *true* class as the *target* class. Our results are reported in Table 3. Note that we distinguish between sentences that are initially correctly classified, and those that are initially falsely classified by the LSTM model. Further, in order to condense the "ablation" results in a single number per method, we compute the accuracy decrease (resp. increase) *proportionally* to two cases: i) random removal, and ii) removal according to *Occlusion*$_{\text{P-diff}}$. Our idea is that random removal is the least informative approach, while *Occlusion*$_{\text{P-diff}}$ is the most informative one, since the relevance for the latter is computed in a similar way to the perturbation analysis itself, i.e. by deleting words from the input and tracking the change in the classifier's prediction. Thus, with this normalization, we expect the accuracy change (in %) to be mainly rescaled to the range $[0, 100]$.

When removing words in decreasing order of their relevance, we observe that *LRP-all* and *CD* perform on-par with near 100% accuracy change, followed by *Gradient×Input* which performs only 66%. When removing words in increasing order of their relevance (which mainly corresponds to remove words with a negative relevance), *Occlusion*$_{\text{P-diff}}$ performs best, followed by *Occlusion*$_{\text{f-diff}}$ and *LRP-all* (around 50%). Unsurprisingly, *Gradient* performs worse than random, since its relevance is positive and thus low relevance is more likely to identify unimportant words for the classification (such as stop words), rather than identifying words that *contradict* a decision. Lastly *Occlusion*$_{\text{f-diff}}$ is less informative than *Occlusion*$_{\text{P-diff}}$, since the former is not normalized by the classification scores for all classes.

Sentence-Level Representations. In addition to testing selectivity, we explore if the word relevance can be leveraged to build sentence-level representations that present some regularities akin word2vec vectors. For this purpose we linearly combine word embeddings using their re-

[10] https://github.com/jiweil/Visualizing-and-Understanding-Neural-Models-in-NLP

[11] In order to remove a word we simply discard it from the input sentence and concatenate the remaining parts. We also tried setting the word embeddings to zero, which gave us similar results.

Accuracy Change (in %)	random	Grad.	Grad.×Input	LRP-prop	LRP-abs	LRP-half	LRP-all	CD	Occlusion$_{\text{f-diff}}$	Occlusion$_{\text{P-diff}}$
decreasing order (std<16)	0	35	66	15	-1	-3	**97**	**92**	**96**	**100**
increasing order (std<5)	0	-18	31	11	-1	3	**49**	36	**50**	**100**

Table 3: Average change in accuracy when removing up to 3 words per sentence, either in *decreasing* order of their relevance (starting with correctly classified sentences), or in *increasing* order of their relevance (starting with falsely classified sentences). In both cases, the relevance is computed with the *true* class as the *target* class. Results are reported *proportionally* to the changes for i) random removal (0% change) and ii) removal based on *Occlusion*$_{\text{P-diff}}$ (100% change). For all methods, the higher the reported value the better. We boldface those methods that perform on-par with the occlusion-based relevances.

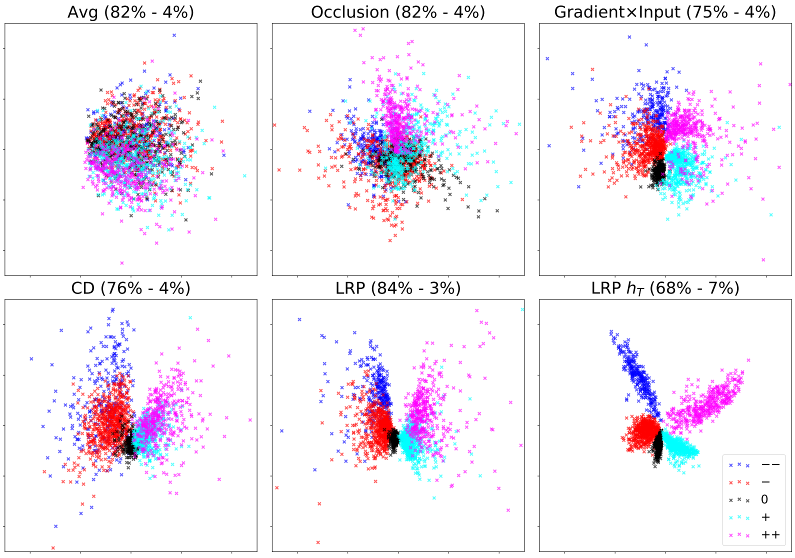

Figure 1: PCA projection of sentence-level representations built on top of word embeddings that were linearly combined using their respective relevance. *Avg* corresponds to simple averaging of word embeddings. For LRP h_T the last time step hidden layer was reweighted by its relevance. In parenthesis we indicate the percentage of variance explained by the first two PCA components (those that are plotted) and by the third PCA component. The resulting representations were roughly ordered (row-wise) from less structured to more structured.

spective relevance as weighting[12]. For methods such as LRP and *Gradient×Input* that deliver also relevances for *single* variables, we perform an element-wise weighting, i.e. we construct the sentence representation as: $\sum_t R_{\boldsymbol{x_t}} \odot \boldsymbol{x_t}$. For every method we report the best performing variant from previous experiments, i.e. *Occlusion*$_{\text{P-diff}}$, *Gradient×Input*, *CD* and *LRP-all*. Additionally we report simple averaging of word embeddings (we call it Avg). Further, for LRP, we consider an element-wise reweighting of the last time step hidden layer h_T by its relevance, since LRP delivers also a relevance for each intermediate neuron (we call it LRP h_T). We also tried using h_T directly: this gave us a visualization similar to Avg. The resulting 2D whitened PCA projections of the test set sentences are shown in Fig. 1. Qualitatively LRP delivers the most structured representations, although for all methods the first two PCA components explain most of the data variance. Intuitively it makes also sense that the neutral sentiment is located between the positive and negative sentiments, and that the very negative and very positive sentiments depart from their lower counterparts in the same vertical direction.

[12] W.l.o.g. we use here the *true* class as the *target* class, since the classifier's 5-class performance is very low. In a practical use-case one would use the *predicted* class instead.

4 Interpreting Single Predictions

Next, we analyze single predictions using the same task and model as in Section 3.3, and illustrate the usefulness of relevance visualization with *LRP-all*, which is the method that performed well in both our previous quantitative experiments.

Semantic Composition. When dealing with real data, one typically has no ground truth relevance available. And the visual inspection of single heatmaps for isolated samples can be counterintuitive for two reasons: the relevance is not accu-

Composition	Predicted	Heatmap	Relevance			# samples
1. "negated positive sentiment"	−	<not> <good>	2.5 $_{0.3}$	-1.4 $_{0.5}$		213
2. "amplified positive sentiment"	++	<very> <good>	1.1 $_{0.3}$	4.5 $_{0.7}$		347
3. "amplified negative sentiment"	−−	<very> <bad>	0.8 $_{0.2}$	4.3 $_{0.6}$		173
4. "negated amplified positive sentiment"	−	<not> <very> <good>	2.74 $_{0.54}$	-0.34 $_{0.17}$	-2.00 $_{0.40}$	1745

Table 4: Typical heatmaps for various types of semantic compositions (indicated in first column), computed with the *LRP-all* method. The LSTM's *predicted* class (second column) is used as the *target* class. The remaining columns contain the average heatmap (positive relevance is mapped to red, negative to blue, and the color intensity is normalized to the maximum absolute relevance), the corresponding word relevance mean (and std as subscript), and the number of bigrams (resp. trigrams) considered for each type of composition.

№	Predicted	Heatmap
1	−−	`it never fails to engage us .`
1a	+	`engages us .`
1b	−	`never engages us .`
1c	−−	`fails to engage us .`
2	++	`ecks this one off your must-see list .`
2a	++	`a must-see film .`
2b	−−	`neither a must-see film .`
2c	++	`never a must-see film .`

Table 5: Misclassified test sentences (1 and 2), and manually constructed sentences (1a-c, 2a-c). The LSTM's *predicted* class is used as the *target* class for the *LRP-all* heatmaps.

rately reflecting the reasons for the classifier's decision (the explanation method is bad), or the classifier made an error (the classifier doesn't work as expected). In order to avoid the latter as much as possible, we automatically constructed bigram and trigram samples, which are built *solely* upon the classifier's predicted class, and visualize the resulting average heatmaps for different types of semantic compositions in Table 4. For more details on how these samples were constructed we refer to the Appendix, note though that in our heatmaps the negation `<not>`, the intensifier `<very>` and the sentiment words act as placeholders for words with similar meanings, since the representative heatmaps were averaged over several samples. In these heatmaps one can see that, to transform a positive sentiment into a negative one, the negation is predominantly colored as red, while the sentiment word is highlighted in blue, which intuitively makes sense since the explanation is computed *towards* the negative sentiment, and in this context the negation is responsible for the sentiment prediction. For sentiment intensification, we note that the amplifier gets a relevance of the same sign as the amplified word, indicating the amplifier is *supporting* the prediction for the considered target class, but still has less importance for the decision than the sentiment word itself (deep red colored). Both previous identified patterns also reflect consistently in the case of a negated amplified positive sentiment.

Understanding Misclassifications. Lastly, we inspect heatmaps of misclassified sentences in Table 5. In sentence 1, according to the heatmap, the classifier didn't take the negation `never` into account, although it identified it correctly in sentence 1b. We postulate this is because of the strong sentiment assigned to `fails` that overshadowed the effect of `never`. In sentence 2, the classifier obviously couldn't grasp the meaning of the words preceding `must-see`. If we use a negation instead, we note that it's taken into account in the case of `neither` (2b), but not in the case of `never` (2c), which illustrates the complex dynamics involved in semantic composition, and that the classifier might also exhibit a bias towards the types of constructions it was trained on, which might then feel more "probable" or "understandable" to him.

Besides, during our experimentations, we empirically find that the *LRP-all* explanations are more helpful when using the classifier's *predicted* class as the *target* class (rather than the sample's *true* class), which intuitively makes sense since it's the class the model is the most confident about. Therefore, to understand the classification of single samples, we generally recommend using this setup.

5 Conclusion

In our experiments with standard LSTMs, we find that the LRP rule for multiplicative connections introduced in Arras et al. (2017b) performs consistently better than other recently proposed rules, such as the one from Ding et al. (2017). Further, our comparison using a sentiment prediction task highlighted that LRP is not equivalent to *Gradient*×*Input* (as sometimes inaccurately stated in the literature, e.g. in Shrikumar et al., 2017) and is more selective than the latter, which is consistent with findings of Poerner et al. (2018). Indeed, the equivalence between *Gradient*×*Input* and LRP holds only if using the LRP ϵ-rule with *no* stabilizer (i.e. with $\epsilon = 0$), and if the network contains *only* ReLU activations and max pooling as non-linearities (Kindermans et al., 2016; Shrikumar et al., 2016). When using other LRP rules, or if the network contains other types of activations, or contains product non-linearities (such as this is the case for LSTMs), then the equivalence does not hold (see Montavon et al. (2018) for a broader discussion).

Besides, we discovered that a few methods such as Occlusion (Li et al., 2017) and CD (Murdoch et al., 2018), are not reliable and get inconsistent results on a simple toy task using an LSTM with only *one* hidden unit.

In the future, we expect decomposition-based methods such as LRP to be further useful to analyze character-level models, to explore the role of single word embedding dimensions, and to discover important hidden layer neurons.

Acknowledgments

We thank Grégoire Montavon for helpful discussions. This work was supported by the German Federal Ministry for Education and Research through the Berlin Big Data Centre (01IS14013A), the Berlin Center for Machine Learning (01IS18037I) and the TraMeExCo project (01IS18056A). Partial funding by DFG is acknowledged (EXC 2046/1, project-ID: 390685689). This work was also supported by the Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (No. 2017-0-00451).

References

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. 2018. Sanity Checks for Saliency Maps. In *Advances in Neural Information Processing Systems 31 (NIPS)*, pages 9505–9515.

Afra Alishahi, Grzegorz Chrupala, and Tal Linzen. 2019. Analyzing and Interpreting Neural Networks for NLP: A Report on the First BlackboxNLP Workshop. *arXiv:1904.04063*. Version 1.

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

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, and Sepp Hochreiter. 2018. RUDDER: Return Decomposition for Delayed Rewards. *arXiv:1806.07857*. Version 2.

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

Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. 2017b. Explaining Recurrent Neural Network Predictions in Sentiment Analysis. In *Proceedings of the 2017 EMNLP Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 159–168. Association for Computational Linguistics.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLoS ONE*, 10(7):e0130140.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. 2010. How to Explain Individual Classification Decisions. *Journal of Machine Learning Research (JMLR)*, 11:1803–1831.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations (ICLR)*.

Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. 2018. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 883–892.

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive Sentence Summarization with Attentive Recurrent Neural Networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 93–98. Association for Computational Linguistics.

Joon Son Chung, Andrew Senior, Oriol Vinyals, and Andrew Zisserman. 2017. Lip Reading Sentences in the Wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3444–3453.

Yannis Dimopoulos, Paul Bourret, and Sovan Lek. 1995. Use of some sensitivity criteria for choosing networks with good generalization ability. *Neural Processing Letters*, 2(6):1–4.

Yanzhuo Ding, Yang Liu, Huanbo Luan, and Maosong Sun. 2017. Visualizing and Understanding Neural Machine Translatio. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1150–1159. Association for Computational Linguistics.

EU-GDPR. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union L119*, 59:1–88.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to Forget: Continual Prediction with LSTM. In *International Conference on Artificial Neural Networks (ICANN)*, volume 2, pages 850–855.

Muriel Gevrey, Ioannis Dimopoulos, and Sovan Lek. 2003. Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological Modelling*, 160(3):249–264.

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2017. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.

Sepp Hochreiter and Jürgen Schmidhuber. 1996. Lstm can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9 (NIPS)*, pages 473–479.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. 2018. Understanding Convolutional Neural Networks for Text Classification. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 56–65. Association for Computational Linguistics.

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3128–3137.

Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. 2016. Investigating the influence of noise and distractors on the interpretation of neural networks. *arXiv:1611.07270*. Version 1.

Yair Lakretz, Germán Kruszewski, Theo Desbordes, Dieuwke Hupkes, Stanislas Dehaene, and Marco Baroni. 2019. The emergence of number and syntax units in LSTM language models. In *To appear in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Will Landecker, Michael D. Thomure, Luís M. A. Bettencourt, Melanie Mitchell, Garrett T. Kenyon, and Steven P. Brumby. 2013. Interpreting Individual Classifications of Hierarchical Networks. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 32–38.

Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. 2016. Analyzing Classifiers: Fisher Vectors and Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2912–2920.

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2019. Unmasking Clever Hans Predictors and Assessing What Machines Really Learn. *Nature Communications*, 10:1096.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. arXiv:1504.00941. Version 2.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and Understanding Neural Models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 681–691. Association for Computational Linguistics.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2017. Understanding Neural Networks through Representation Erasure. arXiv:1612.08220. Version 3.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 4765–4774.

James Martens and Ilya Sutskever. 2011. Learning Recurrent Neural Networks with Hessian-Free Optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1033–1040.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, 65:211–222.

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

Ari S. Morcos, David G.T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. 2018. On the importance of single directions for generalization. In *International Conference on Learning Representations (ICLR)*.

W. James Murdoch, Peter J. Liu, and Bin Yu. 2018. Beyond Word Importance: Contextual Decomposition to Extract Interactions from LSTMs. In *International Conference on Learning Representations (ICLR)*.

Dong Nguyen. 2018. Comparing Automatic and Human Evaluation of Local Explanations for Text Classification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1069–1078. Association for Computational Linguistics.

Nina Poerner, Benjamin Roth, and Hinrich Schütze. 2018. Evaluating neural network explanation methods using hybrid documents and morphosyntactic agreement. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 340–350. Association for Computational Linguistics.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. 2016. Reasoning about Entailment with Neural Attention. In *International Conference on Learning Representations (ICLR)*.

Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. 2017. Evaluating the Visualization of what a Deep Neural Network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning Important Features Through Propagating Activation Differences. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3145–3153.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. 2016. Not Just A Black Box: Interpretable Deep Learning by Propagating Activation Differences. *1605.01713*. Version 1.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *International Conference on Learning Representations (ICLR)*.

Chandan Singh, W. James Murdoch, and Bin Yu. 2019. Hierarchical interpretations for neural network predictions. In *International Conference on Learning Representations (ICLR)*.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642. Association for Computational Linguistics.

Jost T. Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmillers. 2015. Striving for Simplicity: The All Convolutional Net. In *International Conference on Learning Representations (ICLR)*.

Florian Strohm and Roman Klinger. 2018. An Empirical Analysis of the Role of Amplifiers, Downtoners, and Negations in Emotion Classification in Microblogs. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 673–681.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3319–3328.

Huijuan Xu and Kate Saenko. 2016. Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering. In *Computer Vision - ECCV 2016*, pages 451–466.

Yinchong Yang, Volker Tresp, Marius Wunderle, and Peter A Fasching. 2018. Explaining Therapy Predictions with Layer-wise Relevance Propagation in Neural Networks. In *IEEE International Conference on Healthcare Informatics (ICHI)*.

Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Computer Vision - ECCV 2014*, pages 818–833.

A Appendix

A.1 Long-Short Term Memory (LSTM) model

All LSTMs used in the present work have the following recurrence form (Hochreiter and Schmidhuber, 1997; Gers et al., 1999), which is also the most commonly used in the literature (Greff et al., 2017):

$$\begin{aligned}
i_t &= \texttt{sigm}\left(W_i\, h_{t-1} + U_i\, x_t + b_i\right)\\
f_t &= \texttt{sigm}\left(W_f\, h_{t-1} + U_f\, x_t + b_f\right)\\
o_t &= \texttt{sigm}\left(W_o\, h_{t-1} + U_o\, x_t + b_o\right)\\
g_t &= \texttt{tanh}\left(W_g\, h_{t-1} + U_g\, x_t + b_g\right)\\
c_t &= f_t \odot c_{t-1} \;+\; i_t \odot g_t\\
h_t &= o_t \odot \texttt{tanh}(c_t)
\end{aligned}$$

where $\mathbf{x} = (x_1, x_2, ..., x_T)$ is the input sequence, `sigm` and `tanh` are element-wise activations, and $\odot$ is an element-wise multiplication. The matrices W's, U's, and vectors b's are connection weights and biases, and the initial states h_0 and c_0 are set to zero. The resulting last time step hidden vector h_T is ultimately fed to a fully-connected linear output layer yielding a prediction vector $\boldsymbol{f}(\mathbf{x})$, with one entry $f_c(\mathbf{x})$ per class.

The bidirectional LSTM (Schuster and Paliwal, 1997) we use for the sentiment prediction task, is a concatenation of two separate LSTM models as described above, each of them taking a different sequence of word embedding vectors as input. One LSTM takes as input the words in their original order, as they appear in the input sentence/phrase. The other LSTM takes as input the same word sequence but in *reversed* order. Each of these LSTMs yields a final hidden vector, say $h_T^{\rightarrow}$ and $h_T^{\leftarrow}$. The concatenation of these two vectors is then fed to a fully-connected linear output layer, retrieving one prediction score $f_c(\mathbf{x})$ per class.

A.2 Layer-wise Relevance Propagation (LRP) implementation

We employ the code released by the authors (Arras et al., 2017b) (`https://github.com/ArrasL/LRP_for_LSTM`), and adapt it to work with different LRP product rule variants.

In the toy task experiments, we didn't find it necessary to add any stabilizing term for numerical stability (therefore we use $\epsilon = 0$ for all LRP rules). In the sentiment analysis experiments, we use $\epsilon = 0.001$ (except for the *LRP-prop* variant where we use $\epsilon = 0.2$, we tried the following values: [0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 1.0] and took the lowest one to achieve numerical stability).

A.3 Contextual Decomposition (CD) implementation

We employ the code released by the authors (Murdoch et al., 2018) (`https://github.com/jamie-murdoch/ContextualDecomposition`), and adapt it to work with a bidirectional LSTM. We also made a slight modification w.r.t. the author's latest available version (commit e6575aa from March 30, 2018). In particular in file `sent_util.py` we changed line 125 to: `if i>=start and i<stop`, to exclude the stop index, and call the function `CD` with the arguments `start=k` and `stop=k+1` to compute the relevance of the `k`-th input vector, or word, in the input sequence. This consistently led to better results for the *CD* method in all our experiments.

A.4 Toy task setup

As an LSTM model we consider a unidirectional LSTM with a hidden layer of size one (i.e. with one memory cell c_t), followed by a linear output layer with no bias. Since the input is two-dimensional, this results in an LSTM model with 17 learnable parameters. The weights are randomly initialized with the uniform distribution $U(-1.0, 1.0)$, and biases are initialized to zero. We train the model with Pytorch's LBFGS optimizer, with an initial learning rate of 0.002, for 1000 optimizer steps, and reduce the learning rate by a factor of 0.95 if the error doesn't decrease within 10 steps. We also clip the gradient norm to 5.0. With this setting around 1/2 of the trained models on addition and 1/3 of the models for subtraction converged to a good solution with a validation MSE $< 10^{-4}$.

A.5 Semantic composition: generation of representative samples

In a first step, we build a list of words with a positive sentiment (+), resp. a negative sentiment (−), as identified by the bidirectional LSTM model. To that end, we predict the class of each word contained in the model's vocabulary, and select for each sentiment a list of 50 words with the highest prediction scores. This way we try to ensure that the considered sentiment words are clearly identified by the model as being from the positive sentiment (+), resp. the negative sentiment (−) class.

In a second step, we build a list of negations and amplifiers. To that end, we start by considering the same lists of 39 negations and 69 amplifiers as in Strohm and Klinger (2018), from which we retain only those that are classified as neutral (class 0) by the LSTM model, which leaves us with a list of 8 negations and 29 amplifiers. This way we discard modifiers that are *biased* towards a specific sentiment, since our goal is to analyze the compositional effect of modifiers.

Then, for each type of considered semantic composition (see Table 4), we generate bigrams resp. trigrams by using the previously defined lists of modifiers and sentiment words.

For compositions of type 1 ("negation of positive sentiment"), we note that among the constructed bigrams 60% are classified as negative ($-$) by the LSTM model, 26% are predicted as neutral (0), and for the remaining 14% of bigrams the negation is not identified correctly and the corresponding bigram is classified as positive ($+$). In order to remove negations that are ambiguous to the classifier, we retain only those negations which in at least 40% of the cases predict the bigram as negative. These negations are: ['neither', 'never', 'nobody', 'none', 'nor']. Then we average the results over all bigrams classified as negative ($-$).

For compositions of type 2 and 3 we proceed similarly. For type 2 compositions ("amplification of positive sentiment"), we note that 29% of the constructed bigrams are classified as very positive ($++$), and for type 3 compositions ("amplification of negative sentiment"), 24% are predicted as very negative ($--$), while the remaining bigrams are of the same class as the original sentiment word (thus the amplification is not identified by the classifier). Here again we retain only unambiguous modifiers, which in at least 40% of the cases amplified the corresponding sentiment. The resulting amplifiers are: ['completely', 'deeply', 'entirely', 'extremely', 'highly', 'insanely', 'purely', 'really', 'so', 'thoroughly', 'utterly', 'very'] for type 2 compositions; and ['completely', 'entirely', 'extremely', 'highly', 'really', 'thoroughly', 'utterly'] for type 3 compositions. Then we average the results over the corresponding bigrams which are predicted as very positive ($++$), resp. very negative ($--$).

For type 4 compositions ("negation of amplified positive sentiment"), we construct all possible trigrams with the initial lists of negations, amplifiers and positive sentiment words. We keep for the final averaging of the results only those trigrams where both the effect of the amplifier, and of the negation are correctly identified by the LSTM model. To this end we classify the corresponding bigram formed by combining the amplifier with the positive sentiment word, and keep the corresponding sample if this bigram is predicted as very positive ($++$). Then we average the results over trigrams predicted as negative ($-$) (this amounts to finally retain 1745 trigrams).