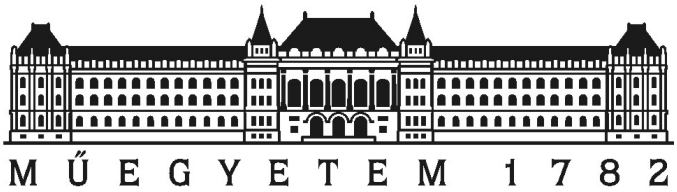

Budapest University of Technology and Economics
Institute of Mathematics

Bachelor Thesis

Explanation of neural language models using SHAP

Emese Vastag

Supervisor: András Benczúr, Ph.D.
SZTAKI

2022

Contents

1 Introduction

Explainability of machine learning models is increasing in importance. The reason for this is that the complexity of most algorithms makes it difficult or impossible for humans to understand why a model made a certain prediction. This is especially problematic in deep learning and artificial neural network approaches, which contain many hidden layers of nodes.

At the same time, in several cases being able to explain a model's prediction has at least that importance as having a model with a good performance.

There are different methods as to how we can explain a model's predictions, we focused on SHAP values, a method introduced by Lundberg and Lee in 2017 [1]. This explanation method is based on Shapley values known from game theory, Lundberg et al. showed that using these values is the unique solution among local feature attribution methods while satisfying some desirable properties.

We examined SHAP values on natural language models. To predict the next word for a given text, we used an LSTM model, which is an improved version of recurrent neural networks. We made predictions for different English texts from Wikipedia and compared the results for different computational approaches of SHAP values.

2 Explaining machine learning models

In order to understand a model prediction, sometimes it is enough to take a look at the model parameters or structure. In simple cases such as linear models or decision trees, this can give us an adequate insight into how the model works and which features carry significant importance.

2.1 Interpretable models

If we have a linear model

$$f(x_1, x_2, \ldots, x_p) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p \tag{1}$$

and take its derivative by x_i, we can see that $\frac{\delta f}{\delta x_i} = w_i$, which means that if x_i increases by 1 unit, we can expect f to increase by w_i units locally. In other words, the weights given to each feature give us a measure of their importance.

Another example is a decision tree, where the algorithmic process is similar to a human decision making: it splits the data into smaller and smaller categories based on certain features by forming nodes of a tree until the leaves of the tree are as homogeneous as possible. When a new data point arrives to be predicted, the tree just checks its feature values to see which leaf it belongs to and gives it the leaf category label. To avoid overfitting, the decision tree is allowed to split the data only a certain amount of times, this way it only uses the features carrying the most relevance in terms of the label. So if we want to see which features had the greatest influence on the prediction, we just have to check which attributes made the model a split on. We can also consider how many training instances the split was made on.

However in more complex cases such as ensemble models or deep neural networks, because of their complexity, it is difficult or impossible for humans to visualize why these models make a certain prediction. In these cases, we have to use an *explanation model* which is an interpretable approximation of the original model.

In their 2017 paper [1] Lundberg and Lee introduce SHAP (SHapley Additive exPlanation) values, a new method to interpret machine learning model predictions. They describe some desirable properties which we would intuitively expect from an explanation model and point out that previous methods violated these properties. SHAP unifies these methods thus it allows for a more intuitive explanation of why a certain outcome is being predicted.

The paper of Lundberg and Lee introduces the concept of additive feature attribution methods. Instead of global interpretability, these models focus on interpreting specific predictions. Local methods take only a single data point and examine what the model predicts for this input, and explain why.

Let f be the original machine learning model we want to explain and g the explanation model to approximate it. Some explanation models use simplified inputs x' that can be transformed to the original input space x through a

mapping function $x = h_x(x')$. In most cases x' is a binary vector representing feature presence or absence. Local methods are approximating $f(h_x(x'))$ with $g(x')$ as accurately as possible.

Definition 1 *Additive feature attribution methods* have an explanation model g which can be described as

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2}$$

where M is the number of simplified input features, $z' \in \{0,1\}^M$ and $\phi_i \in \mathbb{R}$.

Here ϕ_i are the importances assigned to each feature and by summing for all features we get the approximation of the original model's prediction. We can see g is a linear model, therefore easy to interpret.

When introducing SHAP, Lindberg and Lee demonstrated that six previously existing methods can also be considered as additive feature attribution methods.

2.1.1 LIME

Ribeiro, Singh and Guestrin introduced LIME [2] (Local Interpretable Model-agnostic Explanations) in 2016. It is a model-agnostic method: all models are treated as black boxes, and LIME does not know their parameters, only observes their predictions. Therefore it can be applied to any machine learning model.

LIME approximates the original model f with *interpretable models* (eg. linear models). Let G be the class of these models. The original data space needs to be transformed into simpler, more interpretable representations (this is equivalent to the simplified inputs in the SHAP paper). In the case of image data, this often means superpixels, based on image segmentation. For text data, groups of words are frequently used as interpretable variables.

LIME perturbates instances for example by removing some features or adding noise to continuous ones. The interpretable models are then trained on these perturbations and the impact on the output is observed. We want to find an

interpretable model g that best approximates f by minimizing a loss function:

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3}$$

where model g belongs to class G, π_x defines a neighborhood of x in which approximation is sought, L is the loss function measuring the discrepancy between models f and g in the neighborhood π_x, and $\Omega(g)$ is a penalty for the complexity of model g. The penalty is used to favour simpler models from class G.

Since this is a local explanation model that is a linear function, LIME can be described with Equation 2 and is thus an additive feature attribution method.

2.1.2 DeepLIFT

DeepLIFT (Deep Learning Important FeaTures) [3] is a method for deep learning algorithms which is based on the idea that feature importances could be calculated by comparing the activation of each neuron to its *reference activation* and assigning contribution scores according to the difference.

In most cases, choosing the reference is not trivial, but basically, it should be an instance that gives no information about its expected label. For example in the case of image recognition, an empty picture could be a reasonable choice.

If x_0 is the reference input vector with score y_0 and x is the input we want to explain with score y, then Δx and Δy can be defined as $\Delta x = x - x_0$, $\Delta y = y - y_0$. DeepLIFT assigns contribution scores to Δx_i, called *summation-to-delta* property:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta y} = \Delta y. \tag{4}$$

$C_{\Delta x_i \Delta y}$ can be thought of as how much of the difference between y and reference y_0 can be explained with the difference between x_i and its reference.

If we let $\phi_i = C_{\Delta x_i \Delta y}$ and $\phi_0 = f(x_0)$ then this explanation model can be described with Equation 2 and DeepLIFT is also an additive feature attribution method.

2.1.3 Layer-Wise Relevance Propagation

Layer-Wise Relevance Propagation is also a method to be used on deep neural networks. It is equivalent to DeepLIFT with the reference activations of all neurons fixed to zero. It is also an additive feature attribution method.

3 SHAP

3.1 Shapley values

Shapley values were introduced by Lloyd Shapley [4] during his work on game theory. They assign a value to each player in an n-person cooperative game, trying to quantify the individual contribution to the overall gain.

Definition Let U be the universe of players. Then a *game* is a function v that maps subsets of U to real numbers: $v : 2^U \to \mathbb{R}$. It should also satisfy $v(\emptyset) = 0$ and $v(S) \geq v(S \cap T) + v(S \setminus T)$ for all $S, T \subseteq U$.

Definition A *carrier* of v is a set $N \subseteq U$ if $v(S) = v(N \cap S)$ for $\forall S \subseteq U$.

The Shapley value $\phi(v)$ of a game v is a function that assigns a real number to each player $i \in U$. For example, Shapley values can be interpreted as the following.

A coalition of players takes part in a game and obtains a certain overall gain from cooperation. Not all team members participated equivalently in the game, some of them contributed more to the net gain, some of them contributed less. When they are cooperating, their overall gain may differ from just summing up individual results, they may boost or hold back each other. Shapley values are the answer to how to distribute the gain among the players. Higher values are generally given to players who contributed more overall.

Let $\Pi(U)$ be the set of permutations of U, $\pi \in \Pi(U)$ is a given permutation. We can define a function πv by $\pi v(\pi(S)) = v(S)\ \forall S \subseteq U$ where $\pi(S)$ is a permutation of the elements in S. For example, let $U = \{1, 2, 3\}$ and $\pi(U) = (312)$. In that case, some of the values of πv are $\pi v(1) = v(2)$, $\pi v(2) = v(3)$, $\pi v(12) = v(23)$, and $\pi v(123) = v(123)$.

The Shapley values uniquely satisfy a group of axioms described below.

3.1.1 Axioms

Symmetry axiom For each $\pi \in \Pi(U)$, $\phi_{\pi(i)}(\pi v) = \phi_i(v)$.

Efficiency axiom For each carrier N of v, $\sum_N \phi_i(v) = v(N)$.

Linearity axiom For any two games v and w, $\phi(v + w) = \phi(v) + \phi(w)$.

The uniqueness of Shapley values can be shown with the help of three lemmas given by Shapley.

3.1.2 Uniqueness of Shapley values

Lemma 1. Let N be a finite carrier of v. Then for every $i \notin N$, $\phi_i(v) = 0$.

Proof (Lemma 1.) If a set N is a carrier of v, then all supersets of N are carriers of v as well. Therefore $N \cup \{i\}$ is a carrier of v and $v(N) = v(N \cup \{i\})$. From the efficiency axiom it follows that $\phi_i(v) = 0$. □

Shapley defined a simple game $v_R(S)$ and showed that every game with finite carrier is a linear combination of such v_R games.

For any carrier $R \subseteq U, R \neq 0$ we define $v_R(S) = \begin{cases} 1 & \text{if } S \supseteq R \\ 0 & \text{if } S \not\supseteq R. \end{cases}$

This is a simple game for a given subset R of players, which takes the value of 1 if we evaluate it on a superset of R and 0 otherwise. cv_R is also a game for any non-negative c.

We will denote the number of players in R, S, T, N with r, s, t, n respectively.

Lemma 2. If $c \geq 0$ and $0 < r < \infty$, then $\phi_i(cv_R) = \begin{cases} \frac{c}{r} & \text{if } i \in R \\ 0 & \text{if } i \notin R \end{cases}$

Proof (Lemma 2.) Let i and j be players in R, and $\pi(U)$ a permutation of U so that $\pi(R) = R$ and $\pi(i) = j$. Then $\pi v_R = v_R$ and because of the symmetry axiom $\phi_j(cv_R) = \phi_i(cv_R)$. Then we can use the efficiency axiom so that $c = cv_R(R) = \sum_{j \in R} \phi_j(cv_R) = r\phi_i(cv_R)$ for any $i \in R$. From this we easily get $\phi_i(cv_R) = \frac{c}{r}$ if $i \in R$ and because of Lemma 1 $\phi_i(cv_R) = 0$ if $i \notin R$.

□

Lemma 3. We can write any game v with a finite carrier N as a linear combination of v_R games:

$$v = \sum_{\substack{R \subseteq N \\ R \neq 0}} c_R(v) v_R \tag{5}$$

Here the coefficients are $c_R(v) = \sum_{T \subseteq R} (-1)^{r-t} v(T)$ where $0 < r < \infty$.

Proof (Lemma 3.) We must show that Equation 5 holds for any subset of players $S \subseteq U$:

$$v(S) = \sum_{\substack{R \subseteq N \\ R \neq 0}} c_R(v) v_R(S) \tag{6}$$

When $R \subseteq S \subseteq N$, we can see from its definition that $v_R(S) = 1$. Therefore the equation reduces to

$$v(S) = \sum_{R \subseteq S} \sum_{T \subseteq R} (-1)^{r-t} v(T) \tag{7}$$

$$= \sum_{T \subseteq S} \left[\sum_{r=t}^{s} (-1)^{r-t} \binom{s-t}{r-t} \right] v(T) \tag{8}$$

In Equation 7 subsets T were counted for each R containing them, and there are $\binom{s-t}{r-t}$ ways T can have a superset in S, this way we get Equation 8.

In Equation 8, if $s \neq t$, the sum in the brackets is the alterning sum

$$\sum_{i=0}^{s-t} (-1)^i \binom{s-t}{i} = (1-1)^{s-t} = 0 \tag{9}$$

but when $s = t$ the sum equals 1. So from Equation 8 we get $v(S) = v(S)$.

Using the definition of a carrier, we have

$$v(S) = v(N \cap S) = \sum_{\substack{R \subseteq N \\ R \neq 0}} c_R(v) v_R(N \cap S) = \sum_{\substack{R \subseteq N \\ R \neq 0}} c_R(v) v_R(S) \tag{10}$$

□

If we apply Lemma 2 to Lemma 3 we get

$$\phi_i(v) = \sum_{\substack{R \subseteq N \\ i \in R}} \frac{c_R(v)}{r} \tag{11}$$

for all $i \in N$. Substituting the $c_R(v)$ values given in Lemma 3 and by simplifying the result we get

$$\phi_i(v) = \sum_{\substack{S \subseteq N \\ i \in S}} \frac{(s-1)!(n-s)!}{n!} v(S) - \sum_{\substack{S \subseteq N \\ i \notin S}} \frac{s!(n-s-1)!}{n!} v(S) \tag{12}$$

for all $i \in N$.

Theorem *If v is a game with a finite carrier N, a unique function ϕ satisfies the symmetry, efficiency and linearity axioms. For all $i \in U$:*

$$\phi_i(v) = \sum_{S \subseteq N} \frac{(s-1)!(n-s)!}{n!} \left[v(S) - v(S \setminus \{i\}) \right] \tag{13}$$

These ϕ_i values are the Shapley values. They are independent of the carrier N, hence they are well defined.

Proof

If we apply Lemma 1 to Equation 12 we immediately get Equation 13. Its uniqueness is obvious by its derivation. With Lemma 3 we can see that it satisfies all three axioms. □

3.1.3 Methods using Shapley values

Shapley values can be used in machine learning model explanation as well: we can think of each feature as a player and the prediction as to the gain. Shapley values then represent the effect each feature has on the prediction.

An example of how they can be used for linear models is **Shapley regression values**. To examine what effect a feature has, we train models on subsets of features. It is not enough to train a model with and without a certain feature and see what difference in the prediction it causes since features may interact and strengthen each other's importance. Thus it is necessary to retrain a

model on each subset of features.

To compute the ith feature's effect on the prediction, we have to train a model f_S on $S \subseteq F \setminus \{i\}$ and another $f_{S\cup\{i\}}$ on $S \cup \{i\}$ where F is the set of features. The effect of the feature being present is $f_S(x_S) - f_{S\cup\{i\}}(x_{S\cup\{i\}})$ where in x_S only the features in S are present. Then we have to compute this difference for all possible S subsets and take their average weighted by the number of all their possible permutations:

$$\phi_i = \sum_{S \subseteq F\setminus\{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_S(x_S) - f_{S\cup\{i\}}(x_{S\cup\{i\}})\right] \tag{14}$$

Shapley regression values match Equation 2 and are also an additive feature attribution method if we let $\phi_0 = f_\emptyset(\emptyset)$. Here the simplified input x' is a binary vector in which 1 indicates a feature is included in the model and 0 that is excluded. Function h_x maps this to the original input space.

Since we have to train a regression model on all $2^{|F|}$ possible subsets of the features, this method has high computational complexity.

Shapley sampling values is a similar method to Shapley regression values but it can be used to explain any model. It works by applying sampling approximations to Equation 14 and then approximating the effect of removing a variable from the model by integrating over samples from the training dataset. This way there is no need to retrain the model and therefore this method allows fewer differences to be computed due to computational cost. The form of the explanation model is the same as in the case of Shapley regression values, therefore it is also an additive feature attribution method.

Quantitative input influence is another method that is similar to Shapley sampling values. It is an additive feature attribution method.

3.2 Properties of SHAP values

For the class of additive feature attribution methods, the SHAP paper [1] describes three desirable properties: local accuracy, missingness, and consistency, all of them are based on human intuition. There is a strong similarity between Shapley's axioms and the ones described below.

Lundberg and Lee found that these properties restrict the space of possible functions that might describe a feature attribution method. Specifically, among the functions which describe additive feature attribution methods, there is a unique function with unique coefficients ϕ_i which satisfies all three.

Local accuracy

Since the explanation model $g(x')$ is an approximation of the original model $f(x)$, naturally we expect $g(x')$ to match $f(x)$ for a simplified input x'. If we set all x'_i to 1 this is equivalent to the Efficiency axiom of Shapley values.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{15}$$

Here $x = h_x(x')$, and $\phi_0 = f(h_x(\mathbf{0}))$ represents the model output with all simplified inputs set to missing.

Missingness

The simplified input x' usually represents the presence or lack of a feature. So it is intuitive to expect that features which were missing in the original input should have no effect on the prediction.

$$x'_i = 0 \Rightarrow \phi_i = 0 \tag{16}$$

Missingness is required since local accuracy is specified as a linear model. In theory, x' could have a missing feature that has a non-zero ϕ_i Shapley value. But local accuracy would still hold since it would be multiplied with $x'_i = 0$, so to have a unique solution we have to enforce that missing features get a Shapley value of 0.

All methods discussed previously in Sections 2.1 and 3.1.3 satisfy the missingness property.

Consistency

Consistency states that if a model changes in a way such that the marginal contribution of a feature value increases or remains the same regardless of the other features, that feature's Shapley value should not decrease.

Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$. For arbitrary models f and f', if

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i) \tag{17}$$

for all inputs $z' \in \{0,1\}^M$, then $\phi_i(f', x) \geq \phi_i(f, x)$.

Theorem 1 *There is only one explanation model g that is an additive feature attribution method and satisfies Properties 1-3. The coefficients in g are:*

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} \left[f_x(z') - f_x(z' \setminus i) \right] \tag{18}$$

where M is the number of features and $|z'|$ is the number of non-zero features in z'. We take the sum for all $z' \subseteq x'$, this means for all z' vectors where the non-zero entries are a subset of the non-zero entries in x'.

These ϕ_i values in Equation 3.2 are the Shapley values. This means that methods not based on Shapley values violate at least one of the three previously described properties, as a result, they are less consistent with human intuition.

3.3 Computing SHAP

SHAP (SHapley Additive exPlanation) values are the Shapley values of a conditional expectation function of the original model. This means they are the solution to Equation 3.2, where $f_x(z') = f(h_x(z')) = \mathbb{E}[f(z)|z_S]$ and S is the set of non-zero indexes in z'.

Simplified input z' is a binary vector which takes the value of 1 if a feature is present and 0 if it is absent. $h_x(z')$ maps these vectors into the original data space. Most models though cannot handle missing features, so we have to approximate $f(h_x(z'))$. For the approximation we choose the expected value of the model output conditioned on the features whose values are not missing. This can be described as $\mathbb{E}\left[f(z)|z_S\right]$ where in z' only the features in the set S are present. For example, if we have $z' = (1, 0, 0, 1)$, meaning attributes 1 and 4 are present while 2 and 3 are absent, we approximate $f(h_x(z')) \approx \mathbb{E}(f(z)|z_1 = x_1, z_4 = x_4)$ where x_i is the ith feature in $h_x(z')$.

If all of the features in z' are missing we predict $\mathbb{E}(f(z))$. SHAP values credit each feature the change in the expected model prediction when conditioning on that feature. The SHAP paper suggests computing the expected value $\mathbb{E}[f(z)|z_S]$ by sampling from the training dataset. However, this method ignores the correlations between present and absent features. This sometimes causes inconsistency with intuition: the estimation puts too much weight on unlikely instances.

3.3.1 Kernel SHAP

Kernel SHAP is a model-agnostic approximation method based on LIME that allows the calculation of Shapley values with much fewer coalition samples. Since LIME is an additive feature attribution method, based on Theorem 3.2 Shapley values are the only possible solution to Equation 2 if we want to satisfy local accuracy, missingness and consistency. To achieve these values different L, π and Ω have to be chosen then suggested in the LIME paper.

In Kernel SHAP we use a weighted linear regression where the coefficients of the solution are the Shapley values. To build the weighted linear model, n sample coalitions are taken. For each coalition, we calculate the model's prediction for it (using the conditioned expectation) and the weight that will be given to the coalition, which is the SHAP kernel. Finally, we fit a weighted linear model to these values. In their paper [1], Lundberg et al. prove that the resulting coefficients approximate SHAP values.

Theorem 2 *Let f be the model to be explained and g the explanation model. If we want to satisfy Properties 1-3 in the LIME Equation 3, the choices for Ω, $\pi_{x'}(z')$ and L are*

$$\begin{aligned}
\Omega(g) &= 0 \\
\pi_{x'}(z') &= \frac{M-1}{\binom{M}{|z'|}|z'|(M-|z'|)} \\
L(f,g,\pi_{x'})) &= \sum_{z'\in Z}\left[f(h_x(z')) - g(z')\right]^2 \pi_{x'}(z')
\end{aligned}$$

where M is the number of features and $|z'|$ is the number of non-zero features present in z'.

We set the regularization term Ω to 0, L is a squared loss function. Shapley kernel $\pi_{x'}(z')$ is the weight we give to a coalition z', it can be thought of as:

$$\frac{\text{number of features} - 1}{(\text{number of coalitions of size } |z'|) \cdot |z'| \cdot (\text{number of features not present in } z')}$$

We calculate the kernel for each coalition and with these weights, we fit a linear regression with a squared loss function. The coefficients in the solution will be the SHAP values assigning each feature their contributions.

Proof (Theorem 2)

The Shapley kernel is the sample weight given to each coalition $z' \in \{0,1\}^M$. If we denote $|z'|$, the number of present features in z' with s, the kernel for z' is

$$k(z') = k(M,s) = \frac{M-1}{\binom{M}{s} s(M-s)}$$

This means coalitions with the same number of present features will get the same weight.

Let X be a matrix of size $2^M \times M$ which contains all possible $z' \in \{0,1\}^M$ binary vectors. W is a diagonal matrix with the Shapley kernel weights computed for each row of X. We use the Shapley kernel to compute the Shapley values using weighted linear regression:

$$\phi = (X^T W X)^{-1} X^T W y$$

where $y_i = f_x(S_i)$ values are the function outputs for each row of X (where S_i is the set of ones in $X_{i,*}$).

We have to take attention to the cases where s is either 0 or M, since in these cases $k(M,0) = k(M,M) = \infty$. This means W is infinity in the rows linked to the row of all zeros and all ones in X, we should set these infinite weights to a large constant.

In $X^T W X$ the diagonal values are larger then other elements in the matrix

with the following term:

$$\sum_{m=0}^{M-2} \binom{M-2}{m} \frac{M-1}{\binom{M}{m+1}(m+1)(M-m-1)} =$$
$$= \sum_{m=0}^{M-2} \frac{(M-2)!(M-1)(m+1)!(M-m-1)!}{m!(M-m-2)!M!(m+1)(M-m-1)} =$$
$$= \sum_{m=0}^{M-2} \frac{1}{M} = \frac{M-1}{M}$$

This means, $X^T W X = \frac{M-1}{M} I + cJ$ for some positive constant c (where I is the identity matrix and J is the matrix of all ones). The inverse of $X^T W X$ can be written as

$$\left(\frac{M-1}{M} I + cJ\right)(aI + bJ) = a\frac{M-1}{M} I + acJ + b\frac{M-1}{M} J + cbMJ \tag{19}$$

for some $a, b \in \mathbb{R}$. If we choose $a = \frac{M}{M-1}$, this becomes

$$I + \frac{M}{M-1} cJ + b\frac{M-1}{M} J + cbMJ \tag{20}$$

From this,

$$\frac{M}{M-1} cJ + b\frac{M-1}{M} J + cbMJ = 0$$
$$b\left(\frac{M-1}{M} + cM\right) = -\frac{M}{M-1} c$$
$$b = -\frac{\frac{M}{M-1} c}{\frac{M-1}{M} + cM} = -\frac{-M^2 c}{(M-1)^2 + cM^2(M-1)}$$

As $c \to \infty$, b converges to $-\frac{1}{M-1}$. With these a, b values the inverted form becomes $(X^T W X)^{-1} = I + \frac{1}{M-1}(I - J)$.

The term $X^T W$ is a matrix where all the ones in X^T have been replaced with corresponding weights $k(M, s)$, where s is the number of ones in that column of X^T. Multiplying $X^T W$ by $(X^T W X)^{-1}$ creates a matrix of weights to apply to the function outputs in y. If we only consider the Shapley value of a single feature ϕ_j, then we only need to consider a single row of this $M \times 2^M$ matrix, which is equivalent to only using the jth row of $(X^T W X)^{-1}$. When we do

this we see that the value of the weight for row i is

$$
\begin{aligned}
w_i &= k(M, s_i)\left[\mathbf{1}_{X_{i,j}=1} - \frac{(s_i - \mathbf{1}_{X_{i,j}=1})}{M-1}\right] = \\
&= \frac{M-1}{\binom{M}{s_i} s_i(M-s_i)}\left[\mathbf{1}_{X_{i,j}=1} - \frac{(s_i - \mathbf{1}_{X_{i,j}=1})}{M-1}\right] = \\
&= \frac{(M-1)(M-s_i)!\ s_i!}{M!\ s_i(M-s_i)}\left[\mathbf{1}_{X_{i,j}=1} - \frac{(s_i - \mathbf{1}_{X_{i,j}=1})}{M-1}\right] = \\
&= \frac{(M-1)(M-s_i-1)!(s_i-1)!}{M!}\left[\mathbf{1}_{X_{i,j}=1} - \frac{(s_i - \mathbf{1}_{X_{i,j}=1})}{M-1}\right] = \\
&= \frac{(M-s_i-1)!(s_i-1)!}{M!}\left[(M-1)\mathbf{1}_{X_{i,j}=1} - (s_i - \mathbf{1}_{X_{i,j}=1})\right]
\end{aligned}
$$

where s_i is the number of ones in the ith row of X, and $\mathbf{1}_{X_{i,j}=1}$ is one if $X_{i,j} = 1$ and zero otherwise. When $\mathbf{1}_{X_{i,j}=1} = 0$ we get

$$w_i = -\frac{(M-s_i-1)!s_i!}{M!}$$

and when $\mathbf{1}_{X_{i,j}=1} = 1$ we get

$$w_i = \frac{(M-s_i-1)!(s_i-1)!}{M!}[(M-1) - (s_i-1)] = \frac{(M-s_i)!(s_i-1)!}{M!}.$$

Taking the dot product of these values with y leads to the following equation:

$$\phi_j = \sum_{S \subseteq N \setminus \{j\}} \frac{(M-s_i-1)!s_i!}{M!}[f_x(S \cup \{i\}) - f_x(S)]$$

which is a classic form of estimating the Shapley value ϕ_j (N is the set of all features). □

3.3.2 Deep SHAP

Although Kernel SHAP is a model-agnostic method, meaning it can be applied to any machine learning model, there is a more effective method for deep models when assuming feature independence and model linearity.

Deep SHAP is an improved version of DeepLIFT where the method is modified

in a way that it approximates the SHAP values. Theorem 3.2 states that the Shapley values are the unique solution in the class of additive feature attribution methods to satisfy local accuracy, missingness, and consistency. As a consequence, Deep SHAP satisfies all of these properties.

In SHAP, if we assume feature independence and model linearity, $f_x(z')$ can be approximated in the following way:

$$f_x(z') = \mathbb{E}[f(z)|z_S] = \mathbb{E}_{z_{\bar{S}}|z_S}[f(z)] \approx \mathbb{E}_{z_{\bar{S}}}[f(z)] \approx f([z_S, \mathbb{E}[z_{\bar{S}}]]) \tag{21}$$

If we choose $\mathbb{E}[x]$ for reference in DeepLIFT, then it approximates SHAP values assuming independent features and a linear deep model.

In DeepLIFT $C_{\Delta x \Delta y}$ shows how much of the difference between y and its reference can be explained with the difference of x from its reference. A *multiplier* $m_{\Delta x \Delta y}$ is defined as:

$$m_{\Delta x \Delta y} = \frac{C_{\Delta x \Delta y}}{\Delta x} \tag{22}$$

which is the contribution of Δx to Δy divided by Δx. In Deep SHAP, these contributions are the SHAP values, and Δx is the difference between x and its reference, $\mathbb{E}[x]$.

In Deep SHAP, a network's SHAP values are computed by combining SHAP values of smaller components of the network. We can see an example in Figure 1.

First, Deep SHAP gives the contribution as 1 for the whole component. Then for each layer, it divides this contribution in a ratio based on the values of the multipliers. Here a multiplier can be written as:

$$m_{x_j f_3} = \frac{\phi_i(f_3, x)}{x_j - \mathbb{E}[x_j]} \tag{23}$$

This means we normalize the SHAP value so that the sum of multipliers in one layer equals 1. For every layer, the multipliers are rearranged based on how much each neuron contributed, while their sum is still 1 in all layers.

Given the multipliers for each neuron to its immediate successors, we can

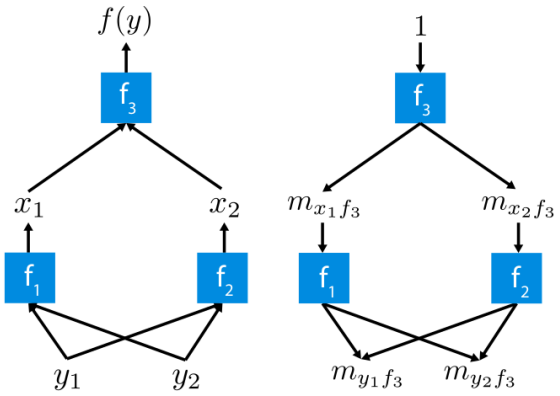

Figure 1: Source: SHAP paper

compute the multipliers for any neuron to a given target neuron efficiently via backpropagation.

The *chain rule for multipliers* is:

$$m_{y_i f_3} = \sum_{j=1}^{2} m_{y_i f_j} m_{x_j f_3} \tag{24}$$

With a linear approximation we get

$$\phi_i(f_3, x) \approx m_{y_i f_3}(y_i - \mathbb{E}[y_i]). \tag{25}$$

In cases when the simple network components are linear, max pooling, or an activation function with just one input, SHAP values can be efficiently calculated for them. This composition rule enables a fast approximation of SHAP values for the whole model.

3.3.3 Linear SHAP

If we have a linear model

$$f(x) = w_0 + \sum_{i=1}^{M} w_i x_i \tag{26}$$

its SHAP values can be easily approximated if we assume feature independence: $\phi_0(f, x) = w_0$, $\phi_i(f, x) = w_i(x_i - \mathbb{E}[x_i])$.

3.4 Choice of value function

The definition of a Shapley value is

$$\phi_v(i) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \left[v(S \cup \{i\}) - v(S) \right]. \tag{27}$$

Here the value function $v(S)$ originally indicates the value that a coalition of players S would earn without their other teammates. When explaining machine learning models, to compute Shapley values, we must define a value function representing the model's output on a coalition x_S, where the feature in S are present and the features in $\bar{S}$ are absent.

According to Chen et al. [5], for $v(S)$ we can choose either observational conditional expectation or interventional conditional expectation.

3.4.1 Interventional conditional expectation

In the case of interventional conditional expectation, the choice of the value function is:

$$v(S) = \mathbb{E}_{p(x')}[f(x_S \sqcup x'_{\bar{S}})]. \tag{28}$$

When using this method, we have to sample features that are not in the coalition ($x'_{\bar{S}}$) unconditionally from the data distribution.

Interventional conditional expectation is used in the SHAP paper [1] to approximate observational conditional expectation. For the missing feature values, we take random samples from the dataset. If we have tabular data we can simply take random values of a given feature, if we have image data, we can take the average of surrounding pixels. After doing this, since we have no missing values anymore, we can use our model to make the prediction.

However, this method breaks dependencies between features in S and $\bar{S}$ and leads to evaluating the model on impossible data points. For example, suppose we are predicting a person's BMI (Body mass index) from the features Age, Height, and Body weight. When both Height and Body weight are missing, we take random samples from the training dataset to replace them. It is possible that we will evaluate our model on a person with 195 cm height and 43 kg weight which seems quite impossible. Basically, we are not taking into

account the inherent correlation between the two features.

Therefore, if we want this method to be viable, the features must be completely independent. However, in reality, feature independence is quite rare. We should use this method if we want to be *true to the model*, meaning that we want the attributions of features to reflect the behavior of a particular model, not the correlations in the data.

3.4.2 Observational conditional expectation

In the case of observational conditional expectation, the choice of the value function is:

$$v(S) = \mathbb{E}_{p(x'|x_S)}[f(x')] \tag{29}$$

Here we have to take the expectation when conditioning out-of-coalition features $x'_{\bar{S}}$ on in-coalition features x_S, therefore observational SHAP values are hard to calculate.

When using observational conditional expectation, the correlation between features is taken into account. However this results in another disadvantage: irrelevant features which were not used by the model may get a nonzero SHAP value.

This violates the so-called Dummy axiom proposed by Sundararajan et al. in 2019 [6]. It states that only features that are referenced by the model should be given importance.

Dummy axiom: If for any x_i, x'_i and for every $x_{N\setminus i}$, $f(x_i; x_{N\setminus i}) = f(x'_i; x_{N\setminus i})$, then $\phi_i = 0$.

This axiom can be violated when a feature that was not used by the model has a high correlation with an important feature and therefore also gets a higher SHAP value. Another disadvantage is that if we have features with high correlations, the SHAP value will be distributed among them and all of them will have insignificant SHAP values.

On the other hand, this method highlights the correlations in the data well, that is, it remains *true to the data*. We should use observational Shapley values when this is more important than understanding the model.

Being able to explain features not included in the model can be useful for bias detection. For example, a linear model may use correlations between features to implicitly depend on a sensitive feature that was explicitly excluded. Observational Shapley values provide a tool to identify such bias.

4 Recurrent Neural Networks

We wanted to examine the SHAP explanation model on natural language predictions. One of the main challenges in natural language procession is to predict the next word based on some given text. For that, we used an RNN (recurrent neural network) model which is ideal for sequential data like text.

The main difference between a traditional neural network and a recurrent neural network is that RNNs contain directed cycles, this way network activation from a previous time step can be fed as inputs to the network to influence predictions at the current time step.

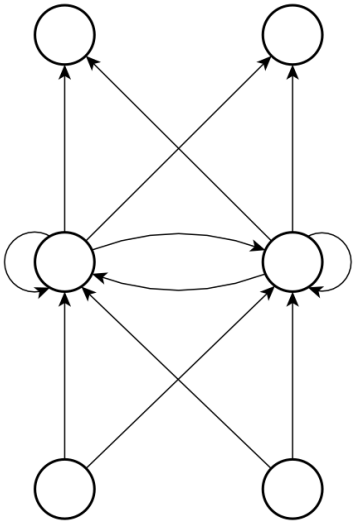

Figure 2: A partial recurrent neural network with a fully connected recurrent hidden layer. Source: [7]

For training a feed-forward neural network, backpropagation is probably the most popular algorithm we can use. In each step, an input is propagated through the network from the input layer until it reaches the output layer. After that, the weights of the network are updated based on what error the

prediction made. To do that, a loss function is chosen, for example the mean square error: $L = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$. If $W_{u,v}$ is the weight that the information neuron v passes as input to neuron u is multiplied with, the effect this weight had on the error can be expressed as $\frac{\delta L}{\delta W_{u,v}}$. To calculate this effect, we have to compute local gradients based on the chain rule. The new weights will be chosen as $W_{u,v} = \eta\frac{\delta L}{\delta W_{u,v}}$, where η is the learning rate, influencing how fast the algorithm converges. We repeat this process with a different input until the weights converge.

However, computing the gradients is not that simple when training an RNN, since we need to propagate through the recurrent connections as well. The most common solutions for this problem are backpropagation through time (BPTT) and real-time recurrent learning (RTRL).

4.1 Backpropagation through time

To backpropagate a recurrent neural network, we can *unfold it over time* to get a feed-forward network. We need a separate layer for each time step with the same weights for all layers. The number of steps to unfold is usually a fixed constant. If weights are identical to the RNN, both networks show the same behavior.

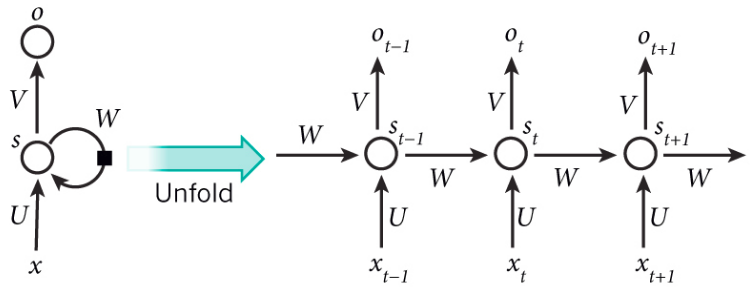

Figure 3: The same RNN in its original form and unfolded over time with a separate layer for each time step. Source: [8]

Since this is a feed-forward network, we can use the backpropagation algorithm.

A problem with the backpropagation of standard RNNs is that they cannot bridge more than 5–10 time steps. This is due to that back-propagated error

signals tend to either grow or shrink with every time step. Over many time steps the error therefore typically blows up or vanishes. This is called the *vanishing or exploding gradient problem*. Blown-up errors lead to oscillating weights, while with a vanishing error, learning takes an unacceptable amount of time, or does not work at all.

4.2 LSTM

A solution for the vanishing gradient problem was presented by Hochreiter in 1997 [7], he called it LSTM. An LSTM memory block consists of a cell, input, forget and output gates.

At a given time step t, the memory block receives an input vector x_t (which includes the bias term as well). Besides that, the block already contains information from the previous time step: the cell state (or memory) c_{t-1}, and the last output (hidden state) h_{t-1} the block made. In each time step, the LSTM memory block uses this information to create a new output and to update the cell state. The process of a memory cell update is shown in Figure 4.

As shown in the figure, the following calculations are made to compute the new output and cell state.

The LSTM block always takes its previous cell state c_{t-1} with a fixed weight 1 to preserve it over time. The problem is, the cell states tend to grow linearly during the progression of a time series presented in a continuous input stream. The main negative effect is that the entire memory cell loses its memorizing capability, and begins to function as an ordinary RNN network neuron.

A forget gate gives a solution to this problem. It is a sigmoid layer that gives an output

$$f_t = \sigma(W_f x_t + U_f h_{t-1})$$

in the range [0,1], which then will be multiplied elementwise with the previous cell state c_{t-1}. This layer selects the amount of information from the previous cell to be included. The closer the output to 0, the more information is forgotten.

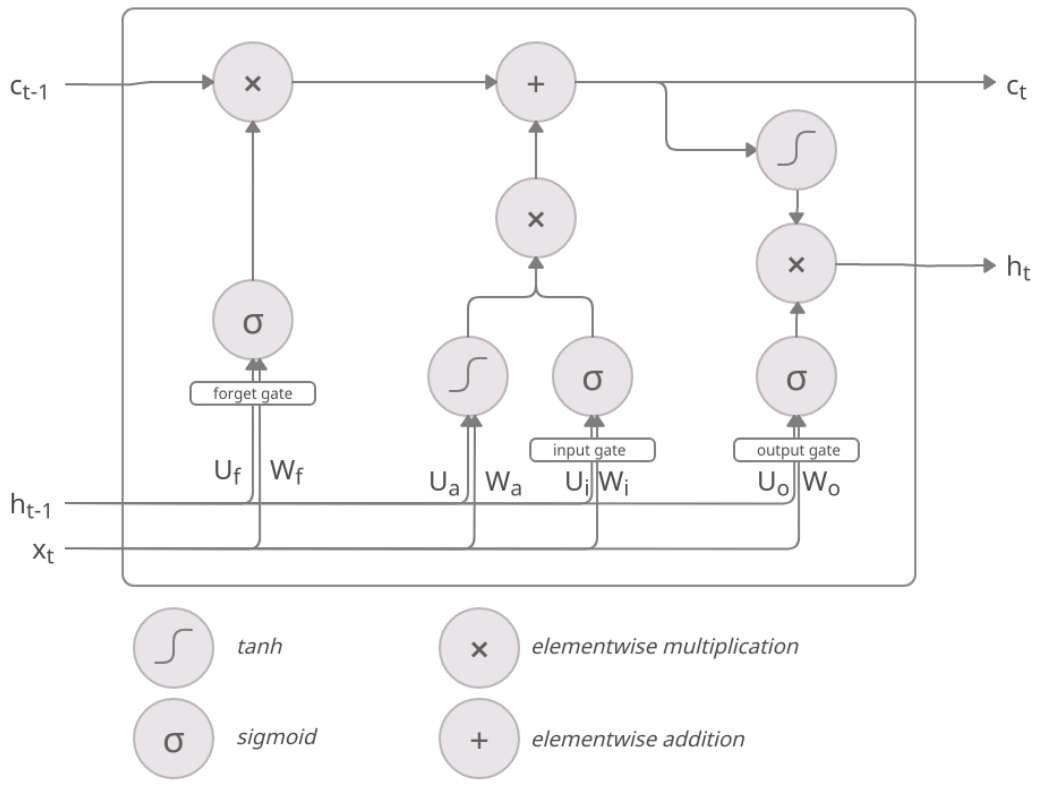

Figure 4: An LSTM memory block

The input gate decides which input informations are useful to be added to the cell state, preventing it from using irrelevant inputs. It is also a sigmoid layer which takes x_t and h_{t-1} as inputs:

$$i_t = \sigma(W_i x_t + U_i h_{t-1}).$$

Another layer uses a tanh activation on these inputs:

$$a_t = \tanh(W_a x_t + U_a h_{t-1}).$$

The elementwise product of these two layers is added to the cell state.

So we can see that the new cell state is given by:

$$c_t = i_t \odot a_t + f_t \odot c_{t-1}.$$

Here $\odot$ denotes elementwise product (Hadamard product).

Finally, the LSTM block computes an output value. For that, it passes the updated cell state through a tanh layer. The output gate is also a sigmoid layer determining how much of this computed output will be actually passed out of the cell, the formula for this is

$$o_t = \sigma(W_o x_t + U_o h_{t-1}).$$

The final output will be

$$h_t = o_t \odot \tanh c_t.$$

5 SHAP on natural language modeling

The models we wanted to explain with SHAP give predictions for a given text, specifically as to which word the text may continues with. Text continuation tasks are typically solved by embedding the words of the language in vector spaces, for which one of the most popular methods was proposed in [9], and later a large number of variants became popular [10].

The features the word embedding models use are the words themselves, as a consequence they have to deal with high-dimensional data, hence for explaining the models, we use the SHAP method of the previous section, since it is capable of handling a large number of features.

The data we used for making predictions on was the validation set of the wikitext-103 dataset introduced in [11], which consists of English text extracted from wikipedia.org. The model works with IDs instead of words, so each word in the text has its assigned unique ID.

5.1 LSTM model and observational SHAP

In order to use the LSTM model, we first *batchified* our data, meaning, we divided it into smaller components of size 10×35. For example, the first five columns of a batch of the validation data (transformed back from IDs into meaningful words) is shown in Table 1.

eastern	Atlantic	Ocean	,	Mediterranean
open	fishing	boat	out	to
's	basketball	team	=	<eos>
portion	of	the	Marine	lines
<eos>	Scientologists	in	Germany	face
backs	as	well	as	several
Google	Play	and	the	Amazon
a	"	sentence	enhancer	"
this	genetic	criteria	to	Indianness
cassette	,	and	CD	versions

Table 1: Each row comes from a different part of the data. Special characters such as commas or quotation marks are handled as independent words. There is a special character <eos>, meaning "end of sequence", used for marking the end of bigger sections in the text.

The LSTM implementation that we use processes one batch in one step, meaning several LSTM units are working on the data at the same time, each of them gives a prediction for a given word.

SHAP has an official implementation in Python. Not only Kernel SHAP (which is model-agnostic), but different model-specific methods of SHAP for several machine learning models such as decision trees, linear models, deep networks etc. However, for the LSTM model we did not use this implementation, since it uses interventional conditional expectation. We preferred observational SHAP values to highlight the correlations in the data.

The implementation we used calculates the weights given in Kernel SHAP while allowing the explainer to work while parts of the text are masked out.

In Table 2 we can see some examples of how SHAP explained the output of the model for a given text. The words are in reverse order, the text is meaningful if read from the bottom to the top. The last word of the text was predicted by the model. From the SHAP values, we can see that the explainer tends to give the most importance to the last words which are closer to the one being predicted. This is consistent with our intuitive expectation, that words much earlier should not have a huge influence later in the text.

Real word: of Predicted word: of	
words	**vals**
parts	3.95585
and	0.0
Sea	-0.38067
Mediterranean	-0.11029
,	0.0
Ocean	0.0
Atlantic	0.0
eastern	0.0
the	0.0
from	0.0
lobster	0.0
clawed	0.0
of	0.0
species	0.0
a	0.0
is	0.0
,	0.0
lobster	0.0
common	0.0
or	0.0
lobster	0.0
European	0.0
the	0.0
as	0.0
known	0.0
,	0.0
gammarus	0.0
Homarus	0.0

Real word: pots Predicted word: traps	
words	**vals**
lobster	5.22089
using	0.6819
caught	0.34544
widely	0.14871
is	0.17158
and	0.0
,	0.0
food	0.54282
esteemed	-0.17046
highly	0.0
a	0.0
is	0.0
gammarus	-0.10772
Homarus	-0.31372
.	0.0
larvae	0.0
planktonic	0.0
into	0.04648
hatching	0.0
before	0.0
year	0.0
a	0.0
to	0.0
up	0.0
for	0.0
females	0.0
the	0.0
by	0.0
carried	0.0
are	0.0
which	0.0
eggs	0.18043
producing	0.0
,	0.0
summer	0.0

Table 2: Examples of the SHAP values given by the explainer

In the first example, we can see that the model probably learned the phrase "parts of", for this reason, the word "parts" has remarkably high importance. In the second example, the model's prediction was incorrect, the SHAP values

here however give an explanation for how the model would predict the correct word.

5.2 GPT-2 model and interventional SHAP

We also tested the official SHAP implementation, the class `Explainer`, the primary explainer in the `shap` package. For that we used another model named GPT-2 first introduced by Radford et al. [12]. This is a transformer model trained on English text. We evaluated this model on the same data we used for the LSTM model, the validation set from the wikitext-103 dataset.

Unlike in the case of the LSTM model, to use GPT-2, the data first needs to be tokenized. Tokenization means dividing the text into smaller units called tokens. A token can be in some cases a whole word, a subword or sometimes just a character. For example, the tokenization of the word "Scientologists" gives us two tokens as result: "Scient" and "ologists". To tokenize the Wikipedia text we used `Autotokenizer` from the `transformers` package.

Since GPT-2 makes a prediction for a sequence of tokens, the SHAP values assign importance to tokens, not words. As a consequence, we could not fully compare the results with our implementation of SHAP.

With the `Explainer` class we made explanations for both the model's predicted word and the actual next word in the text. In Table 3 we can see examples for two predictions. The text can be read from bottom to top here as well. The SHAP values here were given to tokens of the text, not necessarily meaningful words.

In both cases, the model's prediction was not correct, this way we can see the explanations for both the real and predicted words. Similar to our implementation of SHAP for the LSTM model, the last (two) words of the sequence received a significantly higher importance than other words. However, this explainer gave all the other words some negligible SHAP values, but none of them received exact zero. This may happen for the reason the official SHAP implementation returns interventional SHAP values. This method does not take correlations in the data into attention, if features (words) are correlated it tends to divide the SHAP value among them. As a consequence, insignificant

words get nonzero values as well.

Real word: repulsed, predicted word: unable		
words	**pred_vals**	**real_vals**
were	6.236954	-3.416811
but	2.396131	-1.556849
,	0.138509	0.253354
lines	0.070708	0.089487
Marine	0.070708	0.089487
the	-0.031065	-0.050265
of	-0.031065	-0.050265
portion	-0.093174	-0.045004
same	-0.118673	-0.045004
the	0.068392	-0.238928
on	0.315031	-0.308206
attack	0.17514	-0.424449
another	0.082236	-0.235844
conducted	0.739154	-0.340585
battalion	0.339282	-0.373037
uma	-0.052661	-0.158357
K	-0.070273	-0.309621
the	-0.096328	0.005317
of	-0.125748	-0.021117
remnants	-0.148392	-0.069874
the	-0.13076	-0.05021
,	-0.243375	-0.036405
September	0.102513	-0.413121
14	0.157793	-0.235545
on	0.217802	-0.07665
0	-0.107874	0.088374
:	-0.021208	0.556902
23	0.079296	-0.270433
At	-0.00186	-0.012276

Real word: the, predicted word: it		
words	**pred_vals**	**real_vals**
use	3.608833	-3.289817
to	0.122835	-1.190802
never	0.315127	-0.228693
two	-0.12645	-0.324428
the	-0.060507	0.056838
warns	0.36508	-0.298364
he	-0.045031	-0.106871
,	-0.192369	0.59061
them	0.528427	0.276992
hears	0.464144	0.168874
abs	-0.004253	0.166313
Kr	-0.004253	-0.507712
.	-0.101008	0.1214
Mr	-0.709239	-0.8282
when	-0.519075	-0.348843
,	-0.257898	0.133714
However	0.356431	0.255362
.	-0.017904	0.004932
speak	0.143142	-0.007819
they	0.070385	0.002832
sentence	0.070385	0.002832
every	-0.058408	0.020403
in	-0.058408	0.020403
it	0.832759	0.020403
using	0.907465	0.020403
start	0.003851	0.01372
two	0.003851	0.01372
The	0.003851	0.01372

Table 3: Two examples of SHAP's explanations for the GPT-2 model.

5.3 Conclusion

The results were mostly consistent with human intuitions. Significant importance was given to words that are closer to the end of sequences. We would expect this behavior from the model, as words closer to each other have a higher effect on each other as well.

The observational SHAP values gave more intuitive attributions to words, it is able to show correlations in the data. The interventional method, on the other hand, gave all words insignificant SHAP values since it distributed the attribution among correlating features.

6 Acknowledgements

The research presented in this paper, carried out by the Institute for Computer Science and Control, was supported by the Ministry of Innovation and the National Research, Development and Innovation Office within the framework of the Artificial Intelligence National Laboratory Programme.

I am grateful to my supervisors, András Benczúr and Domokos Kelen, for their continued help and support in creating this thesis. Their reviews were an essential contribution to this work.

References

[1] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," *CoRR*, vol. abs/1705.07874, 2017.

[2] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," *CoRR*, vol. abs/1602.04938, 2016.

[3] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *CoRR*, vol. abs/1704.02685, 2017.

[4] L. S. Shapley, *A Value for N-Person Games*. Santa Monica, CA: RAND Corporation, 1952.

[5] H. Chen, J. D. Janizek, S. M. Lundberg, and S. Lee, "True to the model or true to the data?," *CoRR*, vol. abs/2006.16234, 2020.

[6] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," *CoRR*, vol. abs/1908.08474, 2019.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, nov 1997.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[10] Y. Li and T. Yang, "Word embedding for understanding natural language: a survey," in *Guide to big data applications*, pp. 83–104, Springer, 2018.

[11] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *CoRR*, vol. abs/1609.07843, 2016.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.