EMBEDDIA

Cross-Lingual Embeddings for Less-Represented Languages in European News Media

Research and Innovation Action
Call: H2020-ICT-2018-1
Call topic: ICT-29-2018 A multilingual Next generation Internet
Project start: 1 January 2019 Project duration: 36 months

D1.5: Initial Interpretability and Visualisation Technology (T1.4)

Executive summary

Deep neural networks are currently the most successful approach for natural language processing and understanding. Unfortunately, the internal functioning of neural networks is incomprehensible for humans. We have adapted several efficient existing methods for explanation of machine learning models to the specifics of text processing with deep neural networks. We produced TextExplainViz, an initial visualization approach for explanations, and AttViz a visualization method for the currently the most successful transformer neural networks. As current explanation methods can be misled by adversarial attacks, we have developed a new data generator based on Monte Carlo dropout autoencoders and used it in a new method that makes explanations more robust.

Partner in charge: UEDIN

Project co-funded by the European Commission within Horizon 2020 Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	–
RE	Restricted to a group specified by the Consortium (including the Commission Services)	–
CO	Confidential, only for members of the Consortium (including the Commission Services)	–

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825153

Deliverable Information

Document administrative information	
Project acronym:	**EMBEDDIA**
Project number:	**825153**
Deliverable number:	**D1.5**
Deliverable full title:	**Initial Interpretability and Visualisation Technology**
Deliverable short title:	**Initial Interpretability and Visualisation Technology**
Document identifier:	**EMBEDDIA-D15-InitialInterpretabilityAndVisualisationTechnology-T14-submitted**
Lead partner short name:	**UEDIN**
Report version:	**submitted**
Report submission date:	**30/06/2020**
Dissemination level:	**PU**
Nature:	**R = Report**
Lead author(s):	**Marko Robnik-Šikonja (UL), Enja Kokalj (UL), Shane Sheenan (UEDIN), Saturnino Luz (UEDIN)**
Co-author(s):	**Blaž Škrlj (JSI), Senja Pollak (JSI), Domen Vreš (UL)**
Status:	**__ draft, __ final, _x_ submitted**

The EMBEDDIA Consortium partner responsible for this deliverable has addressed all comments received. Changes to this document are detailed in the change log table below.

Change log

Date	Version number	Author/Editor	Summary of changes made
17/05/2020	v1.0	Marko Robnik-Šikonja (UL)	Added introduction, structure, descriptions.
28/05/2020	v1.1	Enja Kokalj (UL)	Description of SHAP, explanations adaptation.
29/05/2020	v1.2	Marko Robnik-Šikonja (UL)	Summary, conclusions, polishing.
30/05/2020	v1.3	Enja Kokalj (UL)	Explanation visualization.
30/05/2020	v1.4	Marko Robnik-Šikonja (UL)	Text polishing.
09/06/2020	v1.5	Shane Sheenan (UEDIN)	Internal review version.
15/06/2020	v1.6	Matthew Purver (QMUL)	Internal review.
16/06/2020	v1.7	Mark Granroth-Wilding (UH)	Internal review.
19/06/2020	v1.8	Marko Robnik-Šikonja (UL)	Response to reviews.
22/06/2020	v1.9	Enja Kokalj (UL)	Response to reviews.
22/06/2019	v1.10	Shane Sheenan (UEDIN)	Response to reviews.
24/06/2020	1.11	Nada Lavrač (JSI)	Report quality checked.
24/06/2020	1.12	Shane Sheenan (UEDIN)	Final improvements.
24/06/2020	1.13	Marko Robnik-Šikonja (UL)	Final improvements.
25/06/2020	final	Saturnino Luz (UEDIN)	Final improvements.
30/06/2020	submitted	Tina Anžič (JSI)	Report submitted.

Table of Contents

List of abbreviations

AI	Artificial Intelligence
ML	Machine Learning
JSON	JavaScript Object Notation
NLP	Natural Language Processing
ELMo	Embeddings from Language Models
BERT	Bidirectional Encoder Representations from Transformers
ANN	Artificial Neural Networks
SVM	Support Vector Machines
DNN	Deep Neural Network
RNN	REcurrent Neural Networks
GLUE	General Language Understanding Evaluation
IME	Interactions-based Method for Explanation
LIME	Local Interpretable Model-agnostic Explanations
SHAP	SHapley Additive exPlanations
BERT	Bidirectional Encoder Representations from Transformers
RoBERTa	Robustly Optimized BERT Pretraining Approach
XLNet	Generalized Autoregressive Pretraining for Language Understanding
TAHV	Text Attention Heatmap Visualization

1 Introduction

Recent developments in artificial intelligence (AI) and in particular in machine learning (ML) has brought this technology into the center of public interest and has increased the requirements for its transparency - it is natural that people affected by automated decisions of algorithms want to get feedback and understand the reasoning process and biases of the underlying models. Two types of technological approaches to increased transparency exist:i nterpretability approaches and visualization of ML models. These two approaches, focused on natural language processing (NLP) approaches, are the topic of this report, which summarizes the work done in task *T1.4 Interpretability and visualization* of the EMBEDDIA project.

1.1 EMBEDDIA and embeddings

The EMBEDDIA project aims to improve the cross-lingual transfer of language resources and trained models using word embeddings and deep neural networks. To process text, neural networks require numerical representation of the given text (words, sentences, documents), referred to as text embeddings. The embedding vectors are obtained from specialized learning tasks, based on neural networks, e.g., word2vec (Mikolov et al., 2013), fastText (Bojanowski et al., 2017), ELMo (Peters et al., 2018), or BERT (Devlin et al., 2019). For training, the embedding algorithms use large monolingual text collections (called corpora) to encode important information about word meaning as distances between vectors. In this way, the embeddings preserve semantic relations between words, and enable machine learning for text understanding tasks. The vector spaces of different languages are similar and this allows cross-lingual transfer of technologies and resources. The idea of contextual embeddings is to generate a different vector for each context a word appears in, and the context is typically defined sentence-wise. For polysemous words, this means that each sense of a word is projected into a significantly different vector which allows successful processing with machine learning algorithms. In this work, we mostly use BERT (Devlin et al., 2019), currently the most successful approach to contextual word embeddings.
.

1.2 Interpretability and visualization of machine learning models

Machine learning models are a crucial component of natural language processing applications. Unfortunately, most of the top performing machine learning models are "black boxes", in the sense that they do not offer an introspection into their decision processes or provide explanations of their predictions and biases. This is true for Artificial Neural Networks (ANN), Support Vector Machines (SVM), and all ensemble methods (for example, boosting, random forests, bagging, stacking, and multiple adaptive regression splines). Approaches that do offer an intrinsic introspection, such as decision trees or decision rules, do not perform so well or are not applicable in many cases (Meyer et al., 2003). To alleviate this problem, two types of model explanation techniques have been proposed. The first type of methods are general, based on perturbations of inputs, and therefore applicable to any prediction model. The second type of methods are specific to certain learning methods such as neural networks and exploit the internal information available during training of these methods.

The general explanation approaches try to efficiently capture the causal relationship between inputs and outputs of the given model. To this end, they perturb the inputs in the neighborhood of a given instance to observe effects of perturbations on model's output. Changes in the outputs are attributed to perturbed inputs and used to estimate their importance for a particular instance. Examples of this approach are methods IME (Štrumbelj & Kononenko, 2010), LIME (Ribeiro et al., 2016), and SHAP (Lundberg & Lee, 2017). These methods can explain models' decision for each individual predicted instance as well as for the model as a whole. The computed impacts of individual inputs can be visualized in the form of histograms. However, these explanation techniques and their visualizations are not adapted to text-based classifiers as their explanations are in the form of lists of impactful words for each

individual decision. For texts with their sequential and structurally dependent nature, this is insufficient. Another shortcoming is that explanation techniques are not adapted to state-of-the-art neural networks such as BERT (Devlin et al., 2019), which use subword input. We address adaptation of explanation techniques to modern neural networks for text classification in Section 3 of this deliverable. We propose adaptations of existing visualization in Section 4.1, and an improved visualization, called TextExplainViz, in Section 4.2.

The model specific explanation techniques exploit internal working of the particular learning algorithm. The explanation methods exploit the model's representation or learning process to gain insight into the presumptions, biases and reasoning leading to final decisions. Due to the rapidly changing landscape of neural network research, these techniques are currently an active area of research, with visualisation forming a core component of the explanations. Visualization of the attention mechanism for text has recently emerged as an active research area due to an increased popularity of attention based methods in natural language processing. Recent deep neural network language models such as BERT (Devlin et al., 2019) comprise multiple attention heads, making human interpretation and investigation of the attention matrices a time consuming and complex task. Visualisation has been proposed as a solution for gaining insight into the operation of attention based models (Vig, 2019). In Section 4.3, we contribute a new visualisation tool, called AttViz, that enhances the interpretability of classification results through visualised self-attention.

1.3 Contributions and structure of the deliverable

The objectives of workpackage WP1 of the EMBEDDIA project are to advance cross-lingual word embedding technologies, together with methods for deep learning, and methods for explanation and visualisation of their outputs. The aim of T1.4 is to address advancement of explanation and visualization technologies for text-based deep learning approaches.

This report describes the results of the work performed in T1.4 from the start of the task at M10 till M18. The main contributions presented in this report (in the order of appearance) are as follows:

1. Adaptation of explanation methods IME, LIME, and SHAP to state-of-the-art text classification method BERT, described in Section 3.1; and adaptation of their visualizations described in Sections 4.1 and 4.2;
2. A novel data generation approach using Monte Carlo dropout autoencoders and variational autoencoders , described in the appended paper by Miok et al. (2019); the generator will help in the development of more robust explanation approaches as outlined in Section 3.2;
3. Development of a new visualization method for the state-of-the-art text classification method BERT based on self-attention described in Section 4.3 and in the appended paper by Škrlj et al. (2020).

This report proceeds by providing background and related work in Section 2. Our work on adapting explanation methods to modern neural networks with subword input is presented in Section 3. Section 4.1 present our work on visualization: adaptation of existing visualizations to subword input, and the novel AttViz system for the visualisation of self-attention in text classification. The outputs associated with this deliverable are provided in Section 5. Finally, Section 6 provides conclusions and ideas for future work in interpretability and visualisation. The associated papers can be found in Appendices A and B.

2 Background and related work

In this section, we present a short overview of explanation and visualization techniques, with focus on deep neural networks and text.

2.1 Deep neural networks for text classification

Deep learning (LeCun et al., 2015; Goodfellow et al., 2016) differs from other machine learning algorithms by allowing computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Text is usually treated as a sequence by deep neural networks. Sequences can be of different length and typically there is some dependency between different positions in a sequence (e.g., a verb in a sentence may determine the subsequent choice of nouns). A standard choice of neural network architecture for sequences is recurrent neural networks (RNN) in which the state at each point in the sequence depends on not only the current input but also the previous state. The information from the previous processing steps persists in the network, effectively allowing the network to memorize previous processing, which is well suited for sequences. RNNs are very effective in processing speech, text, signals, and other sequential data. RNNs also introduce many challenges, for example the convergence of learning to stable weights is much slower.

Recently, BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019), a new state-of-the-art deep neural network approach to language modeling, text classification, and contextual embeddings was introduced. BERT generalises the idea of language models to masked language models—inspired by cloze (gap filling) tests—which test the understanding of a text by removing a certain portion of words that the participant is asked to replace. The masked language model randomly masks some of the tokens from the input, and the task of the language model is to predict the missing token based on its neighbourhood. BERT uses the transformer ANN architecture(Vaswani et al., 2017), which uses both left and right context in predicting the masked word and further introduces the task of predicting whether two sentences appear in a sequence. The input representations of BERT are sequences of tokens representing subword units. Some very common words are kept as single tokens, others are split into subwords (e.g., common stems, prefixes, suffixes—if needed down to single-letter tokens). The original BERT project offers pre-trained English, Chinese and multilingual models; the latter is trained on 104 languages simultaneously. BERT has shown excellent performance on 11 NLP tasks: 8 from the GLUE (General Language Understanding Evaluation) benchmark (Wang et al., 2018), question answering, named entity recognition, and common-sense inference.

Rather than training an individual classifier for every classification task, which is resource and time expensive, a pre-trained multilingual BERT language model is used and fine-tuned on a specific task. Trained on huge text collections, BERT stores general language representation that is successfully exploited in many tasks, Frequently, this approach requires less task-specific data.

The use of BERT in a token classification task only requires adding a final layer (number of neurons equals the number of class values in the intended task) with softmax activation, and connections between its last hidden layer and the new neuron. To classify a sequence, one usually takes the vector for the special CLS token before the classification layer to reduce the dimensionality. The fine-tuning process is then applied to either only the last layer of the network, or, more frequently, to the whole network. In the latter case all parameters of BERT and new class-specific weights are fine-tuned jointly to maximize the log-probability of the correct labels.

2.2 Explanation methods for text classification

Due to recent successes of Deep Neural Networks (DNNs) in image recognition and NLP, several explanation methods specific to these two application areas emerged. For example, in language processing, Arras et al. (2017) applied layer-wise relevance propagation to a convolutional neural network. The explanations indicate how much individual words contribute to the overall classification decision. In this section we focus on a general class of explanation methods, which are applicable to all types of classifiers but may need specific adaptations for use in text classification. While demonstrate our adaptations of these methods on the BERT model, these adaptations are applicable to other DNN models as well.

General explanation methods can be applied to any classification model which makes them a useful tool both for interpreting models (and their predictions) and comparing different types of models. By modification of feature values of interest, what-if analysis is also supported. Such methods cannot exploit any model-specific properties (e.g., gradients in ANN) and are limited to perturbing the inputs of the model and observing changes in the model's output (Robnik-Šikonja & Kononenko, 2008; Lemaire et al., 2008; Štrumbelj & Kononenko, 2010). Most explanation methods can provide two types of explanations for prediction models: explanation of predictions for individual instances, and model explanations. Model explanations work by summarizing a representative sample of instance explanations to show the properties of the whole model.

The key idea of the perturbation-based explanation method is that the contribution of a particular input value (or set of values) can be captured by "hiding" that input and observing how the output of the model changes. As such, the key component of general explanation methods is the expected conditional prediction - the prediction where only a subset of the input variables is known. Let Q be a subset of the set of input variables $Q \subseteq S = \{X_1, \ldots, X_a\}$. Let $p_Q(y_k|x)$ be the expected prediction for x, conditional on knowing only the input variables represented in Q:

$$p_Q(y_k|x) = \mathbf{E}(p(y_k)|X_i = x_{(i)}, \forall X_i \in Q). \tag{1}$$

Therefore, $p_S(y_k|x) = p(y_k|x)$. The difference between $p_S(y_k|x)$ and $p_Q(y_k|x)$ is a basis for explanations. In practical settings, the classification function of the model is not known - one can only access its prediction for any vector of input values. Therefore, exact computation of $p_Q(y_k|x)$ is not possible and sampling-based approximations are used.

In principle, to understand the behaviour of a prediction model, one would have to observe its behavior for all subsets of input features and their values. Such a procedure demands 2^a steps, where a is the number of attributes (i.e. features), and results in the exponential time complexity. A solution was proposed in (Štrumbelj & Kononenko, 2010) by observing that the contribution of each variable corresponds to the Shapley value for the coalitional game of a players. Here, players correspond to input features, and the coalitional game corresponds to the prediction of an individual instance.

The original Shapley values deal with a coalition of a players that cooperate and together generate a certain overall gain. Shapley values represent a solution to the problem of finding the fairest distribution of gain among all players, which takes into account the importance of each player in obtaining that gain (Shapley, 1953). In the case of explaining a prediction the instance's feature values form a coalition which causes a change in the prediction. This change represents the difference between the prediction for this instance and the expected prediction if no feature values are given (i.e. default class is predicted). The gain is divided among feature values in a way that reflects their impact (i.e. average marginal contribution across all possible sub-coalitions) on the change in the prediction.

The methods discussed in this work, IME, SHAP, and LIME, are the state-of-the-art explanation methods. They are all based on the Shapley value approximation principle (Lundberg & Lee, 2017). They estimate the impact of a particular feature on the prediction of a given instance by perturbing similar instances. For textual problems the perturbation process is nontrivial, as the generation of new perturbed text instances may produce completely uninformative texts. Below we describe the three methods, and in Section 3 we describe their adaptations to neural networks with subword input.

2.2.1 Explanation method IME

To produce fair and efficient explanations, Štrumbelj & Kononenko (2010) use an approximation to Shapley values based on sampling. Their method is called IME (Interactions-based Method for Explanation) and is based on an alternative formulation of the Shapley value (the classical one is presented in Eq. (7)). The contribution of the feature i is expressed as:

$$\phi_i(f, x) = \frac{1}{M!} \sum_{O \in \pi(N)} (f_x(Pre^i(O) \cup i) - f_x(Pre^i(O))), \qquad i = 1, \ldots, n. \tag{2}$$

where f is the original model, x is the original input, M is the number of features, N is a set representing all of the features, $\pi(N)$ is a set of all ordered permutations of N, $Pre^i(O)$ is the set of predecessors of the feature i in the order $O \in \pi(N)$, and f_x corresponds to a score returned by the explained prediction model.

The procedure for approximating the contribution of a given value of the feature i in the explained instance is as follows. First, we determine m, the number of random samples that we will draw (with replacement). For each drawn sample, we generate two new instances by randomly permuting the features ($O \in \pi(N)$) and replacing each feature's value that appears before the i-th feature (the value of which we are explaining) in order O with the feature's value in the original instance. For one of the two new instances, we replace the i-th feature's value as well. In the end, the two new instances differ only in the value of the i-th feature. Then we compute the prediction difference between the two new instances in each iteration (m-times) and the average difference represents the contribution for the selected feature's value.

While the entire procedure is linear in the number of features a (Štrumbelj & Kononenko, 2010), in practice the convergence is slow and the method gets prohibitively slow with the number of attributes exceeding a hundred. The slowness is due to many samples required before the approximation of Shapley value converges. The required number of samples to draw is related to the desired approximation error, which depends on the population variance. We estimate the required number of samples to achieve the desired error during the sampling process by computing the variance of the samples we have already drawn. The minimal number of samples for the entire explanation is defined as:

$$m_{min}(\langle 1-\alpha, e\rangle) = n \cdot \frac{Z^2_{1-\alpha} \cdot \bar{\sigma^2}}{e^2}, \tag{3}$$

where e is the allowed error of approximation, α the probability of the error, $\bar{\sigma^2}$ the estimated variance, and Z the quantile of the normal distribution. Since during the approximation procedure the mean and variance of the final population are not yet known, in our work we use an alternative formulation of the variance which can be computed in an online fashion, during the estimation procedure.

$$Var(x) = E[(X-\mu)^2] = E[(X-E(X))^2] = E[X^2] - E[X]^2, \tag{4}$$

where μ ($E[X]$) is a squared deviation from the mean of X, $E[X^2]$ is the mean of the square of X, and $E[X]^2$ is the square of the mean of X.

In this way, we satisfy the desired error bounds and improve the running times. The number of required samples reflects the complexity of explanation for a given model's prediction. This number might be very different for different instances.

As mentioned above, the problem of this method is that it only works fast enough for up to a hundred features, and is therefore not useful for text classification. In text classification, the features typically represent words. When we sample the instances in the process of Shapley value estimation (as described above), each feature can therefore be assigned any value from the dictionary (typically of the dimension $> 10,000$). This is a serious obstacle for practical use of this method. We attempted to speed up the method with more efficient language model based sampling but current results are not yet completely satisfactory.

2.2.2 Explanation method LIME

LIME (Local Interpretable Model-agnostic Explanations) by Ribeiro et al. (2016) calculates explanations for large data sets relatively efficiently, in terms of a number of instances and number of features. It uses perturbations in the locality of an explained instance to produce explanations (similarly to locally weighted regression) as illustrated in Figure 1. LIME defines explanations as an optimization problem and tries to find a trade-off between local fidelity of explanation and its interpretability. The search space

is over explanations generated by interpretable models $g \in G$, where G is a class of interpretable models (in practice, linear models are used). Interpretability is quantified with the complexity of explanations $\Omega(g)$, where complexity measure Ω can be the depth of tree for decision trees or the number of non-zero weights for linear models. The model f being explained has to return numeric values $f : \Re^d \rightarrow \Re$, for example probability scores in classification. Locality is defined using a proximity measure π between the explained instance x and perturbed points z in its neighborhood. Local infidelity $L(f, g, \pi)$ is a measure of how unfaithful the explanation model g is in approximating the prediction model f in the locality defined by $\pi(x, z)$. The chosen explanation then minimizes the sum of local infidelity L and complexity Ω:

$$e(x) = \arg\min_{g \in G} \; L(f, g, \pi) + \Omega(g) \tag{5}$$

The approach uses sampling around explanation instance x to draw samples z weighted by the distance $\pi(x, z)$. The samples form a training set for a model g from an interpretable model class, e.g., linear model. Due to locality enforced by π, the model g is hopefully a faithful approximation of f. In practice, Ribeiro et al. (2016) use linear models as a class of interpretable models G, the squared loss as a local infidelity measure, number of non-zero weights as complexity measure Ω, and choose sample points in the neighborhood of explanation instance x according to the Gaussian distribution of distance between x and sampled point z.

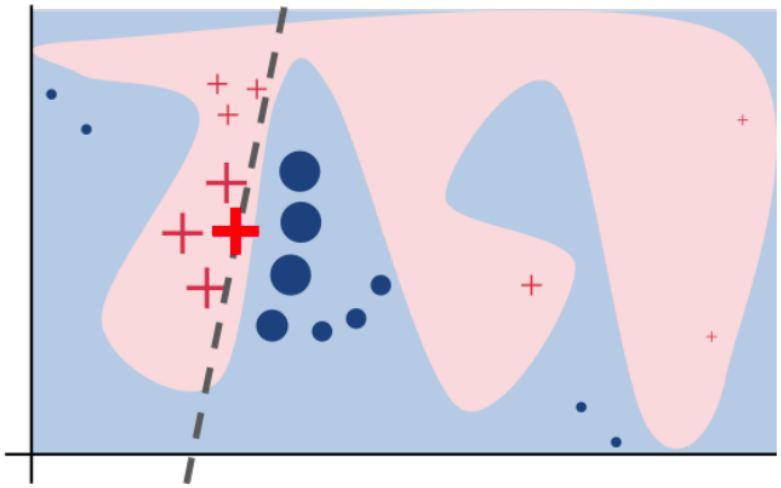

Figure 1: The process of generating explanations with the method LIME. The method is based on the local linear approximation of the model behaviour. For each instance that is being explained (i.e. red cross) LIME samples instances in the neighborhood (i.e. blue/red instances) and assigns weights that correspond to the distance between them. Then it learns a linear model (i.e. dashed line) that is locally (but not necessarily globally) faithful. The global model function is represented with the blue/red background.

To explain text classification tasks, LIME uses a bag-of-words representation to output and visualize a limited number of the most locally influential words. By presenting the explanation as an optimization problem, LIME avoids the exponential search space of all feature combinations which is solved by game-theory based sampling in IME. However, LIME offers no guarantees that the explanations are faithful and stable. By using the neighborhood around the explained instance, it may fall into a curse of dimensionality trap, which is fatal for neighborhood-based methods like kNN in high dimensional spaces, i.e. the local prediction methods become unreliable in high dimensional spaces. The problem of feature interactions is seemingly avoided by using an approximating function from a class of interpretable explanation but the problem is just swept under the carpet, as the interpretable explanation class may not be able to detect interactions. In the current settings the method assumes at least locally linear decision boundaries, but the assumption is not true for categorical variables, for instance.

2.2.3 Explanation method SHAP

The method SHAP (SHapley Additive exPlanations) is another perturbation based Shapley value approximating explanation method that assigns the importance value to each feature in an individual prediction (Lundberg & Lee, 2017).

In recent years various explanation methods for complex machine learning models have been proposed. As complex models are incomprehensible for humans, some methods try to explain the model's predictions by using simpler explanation models (i.e. interpretable approximations of the original model). These models are designed to explain a prediction of the original model f for an individual input x. They use simplified inputs x' that map to original inputs through a specific mapping function h_x. Local methods generate instances z' in the vicinity of x' and try to ensure that $g(z') \approx f(h_x(z'))$ whenever $z' \approx x'$, where g represents the explanation model.

The authors of SHAP have noted that several existing methods use the same explanation principle and have defined a class of additive feature attribution methods that generalizes them. Additive feature attribution methods have an explanation model that is defined with the following equation:

$$g(z') = \phi_0 + \sum_{i=1}^{M} (\phi_i z_i'), \tag{6}$$

where g is the explanation model, z' are instances generated in the vicinity of x', M is the number of simplified input features, ϕ_0 is the effect if no feature values are given, i.e. $\phi_0 = f(h_x(\emptyset))$, and ϕ_i is the effect of each feature. The sum of all the effects approximates the output $f(x)$ of the original model.

Lundberg & Lee (2017) have shown that several existing methods match this equation, including LIME (Ribeiro et al., 2016) and IME (Štrumbelj & Kononenko, 2010). The class of additive feature attribution methods introduces three desirable properties of explanation, and a single unique solution to Eq. (6). The first property is local accuracy, which states that when approximating the original model f for a specific input x, the output should match the explanation model g for the simplified input z'. The second property is missingness, which means that the features that are missing in the input should have no impact. The third property is consistency, which states that if a model changes in a way that some simplified input's contribution increases or stays the same, then that input's impact should not decrease.

The unique solution for Eq. (6) that satisfies all three desired properties are Shapley values ϕ_i:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)], \tag{7}$$

where f is the original model, x is the original input, M is the number of features, $|z'|$ is the number of non-zero entries in z', $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x', and f_x corresponds to a score returned by the explained prediction model.

To improve possible violation of local accuracy and consistency in previous methods such as LIME and improve efficiency compared to IME, Lundberg & Lee (2017) introduce SHAP values as approximation to Shapley values. They proposed a model-agnostic approximation method called Kernel SHAP as well as several faster model-specific adaptations (e.g., Deep SHAP for a specific class of neural networks).

In our work we use Kernel SHAP, which combines weighted linear regression from LIME with Shapley values. LIME uses a linear explanation model to locally approximate the original model, but because its parameters are chosen heuristically, they are not necessarily optimal and the consequence is a violation of local accuracy and/or consistency, which makes the method's explanations unintuitive in certain circumstances. However, by choosing the loss function, the weighting kernel and the regularization term in a way that they respect the three Shapley properties, we get better sample efficiency than directly

using the classical Shapley equations. Conceptually, non-continuous boundaries could be a problem for linear regression used in SHAP, e.g., transitions between categorical values could be poorly approximated by linear regression models. In practice, SHAP produces sensible results for bag-of-words text representation, while more thorough analysis of its limitations in discrete spaces is still an open research question.

2.3 Attention Visualisation for Text

Visualization of the attention mechanism for text has recently emerged as an active research area due to an increased popularity of attention based methods in natural language processing. Recent deep neural network language models such as BERT (Devlin et al., 2019), XLNet (Z. Yang et al., 2019), and RoBERTa (Y. Liu et al., 2019) are comprised of multiple *attention heads*—separate weight spaces each associated with the input sequence in *a unique way*. Language models consist of multiple attention matrices, all contributing to the final prediction. Visualising the attention weights from each attention matrix is thus an important component in understanding and interpreting these models.

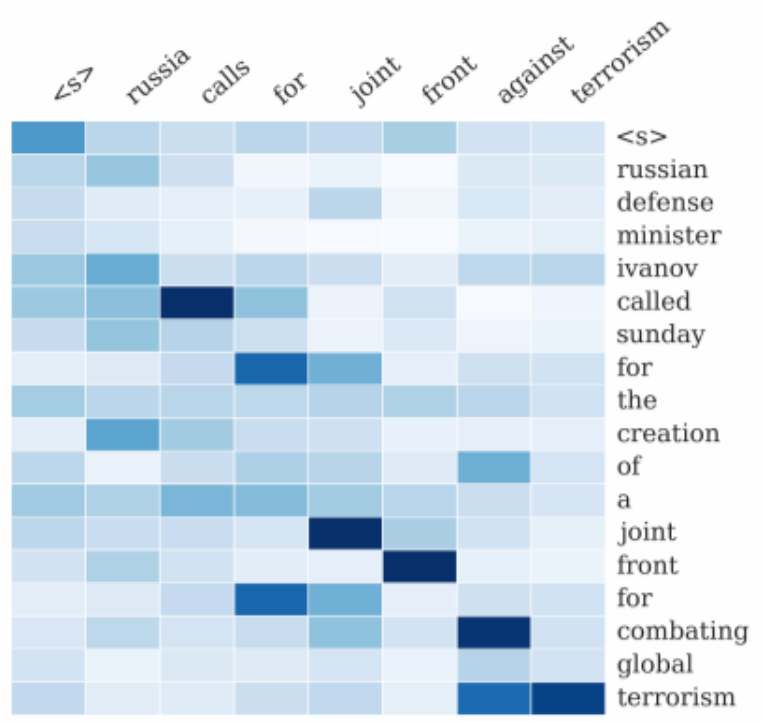

Figure 2: Figure from Rush et al. (2015). Example output of the attention-based summarization (ABS) system. The heatmap represents a soft alignment between the input (right) and the generated summary (top). The columns represent the distribution over the input after generating each word.

The attention mechanism which originated in the work on neural machine translation lends itself naturally to visualisation. Bahdanau et al. (2014) used *heat maps* to display the attention weights between input and output text. This visualisation technique was first applied to the task of translation but found use in many other tasks such as visualising an input sentence and output summarization (Rush et al., 2015) and visualizing an input document and textual entailment hypothesis (Rocktäschel et al., 2015). In these heat map visualisations, a matrix or a vector is used to represent the learned alignments and colour intensity illustrates attention weights. This provides a summary of the attention patterns describing how they map the input to the output. In Figure 2, showing an example from Rush et al. (2015), the attention patterns between a source sentence and generated sentence summary are displayed using the heatmap technique. For classification tasks, a similar visualisation approach can be used to display

the attention weights between the classified document and the predicted label (Z. Yang et al., 2016; Tsaptsinos, 2017). Here, the visualisation of attention often displays the input document with the attention weights *superimposed* onto individual words. The superimposed attention weights are represented similarly to heat map visualisations using a colour saturation to encode attention value. The neat-vision tool[1] encodes attention weights associated with input text in this manner; Figure 3 shows sections from an input file, and highlighting, via colour intensity, the words with high attention score. Similarly, the Text Attention Heatmap Visualization (TAHV[2]) which is included in the NCRF++ toolkit (J. Yang & Zhang, 2018) can be used to generate weighed sequences which are visualised using superimposed attention scores.

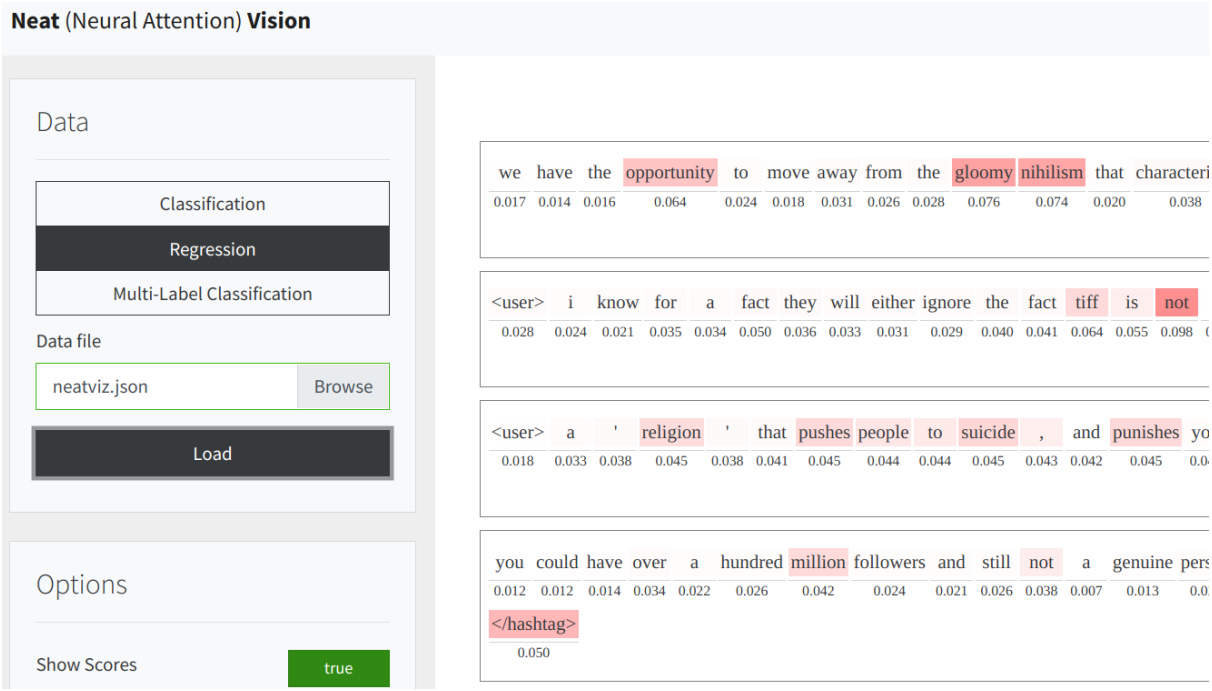

Figure 3: The neat-vision tool displaying attention weights superimposed onto sections of text from an input file.

An alternative visual encoding of attention weights is a bipartite graph visualisation. Here attention weights are represented by edge weights or thickness between two lists of words. This technique has been used to help interpret model output in neural machine translation (Lee et al., 2017), in natural language inference (S. Liu et al., 2018a), and for model debugging (Strobelt et al., 2018). A version of this visualisation which was designed specifically for multi-head self-attention (Vaswani et al., 2017) uses colour hue to encode the attention head associated with each weight. The visualisation, shown in Figure 4, has colour applied to the edges and superimposed as a colour strip over the node words. The intensity of the colours in the strips at each word position summarises the distribution of attention weight for that word across the heads.

This multi-head self visualisation technique was recently extended by the BertViz[3] tool (Vig, 2019). BertViz provides a bipartite graph visualisation and two extensions of the technique. The first, called "Model View", is a visualisation of the bipartite graphs for each layer and head in a model arranged in a matrix. This follows the visualisation design principle of overview and detail on demand by presenting token to token attention patterns across all attention heads for a layer, and enabling further investigation of each single bipartite graph individually. The second visualization, "Neuron View", drills down to the computation of the attention score associated with each weighed edge in the bipartite graph. The

[1] https://github.com/cbaziotis/neat-vision
[2] https://github.com/jiesutd/Text-Attention-Heatmap-Visualization
[3] https://github.com/jessevig/bertviz

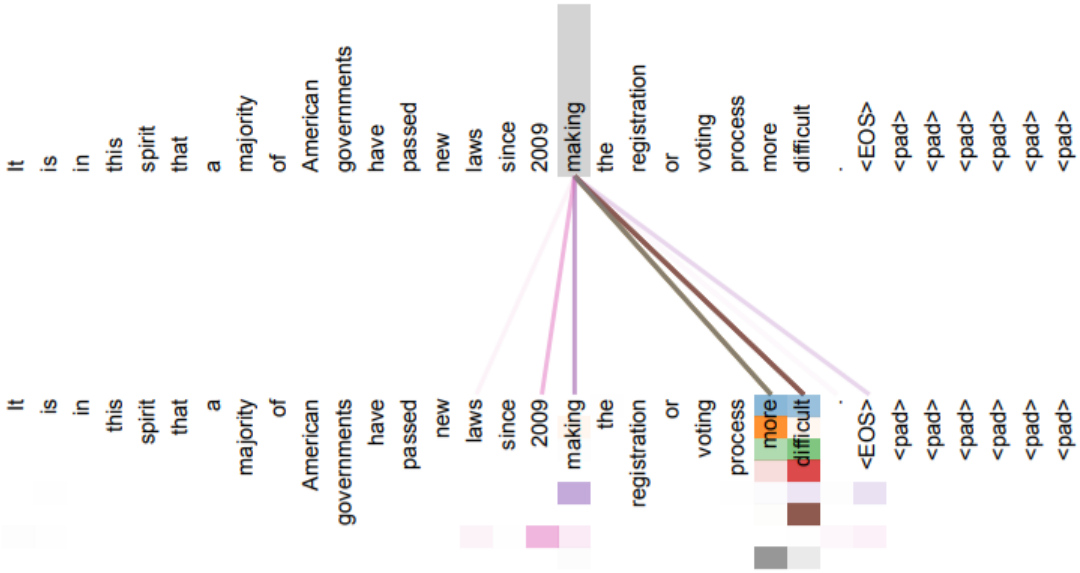

Figure 4: Figure from Vaswani et al. (2017). An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of a 6 layer model. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colours represent different self-attention heads.

element-wise product, dot product, and softmax values are all visualised using coloured elements with saturation representing the magnitude of the value. This view provides a visual summary of the query and key vectors and shows how they are used to compute attention. Figure 5 shows the query vector for the word "on" selected, the visual representation of the the dot product of the query vector and the key vectors of each word in the text segment provide the attention score bettween each word and "on", displayed using color intensity. This visualisation provides some insight into how each attention weight was computed while still providing the overview of attention weights.

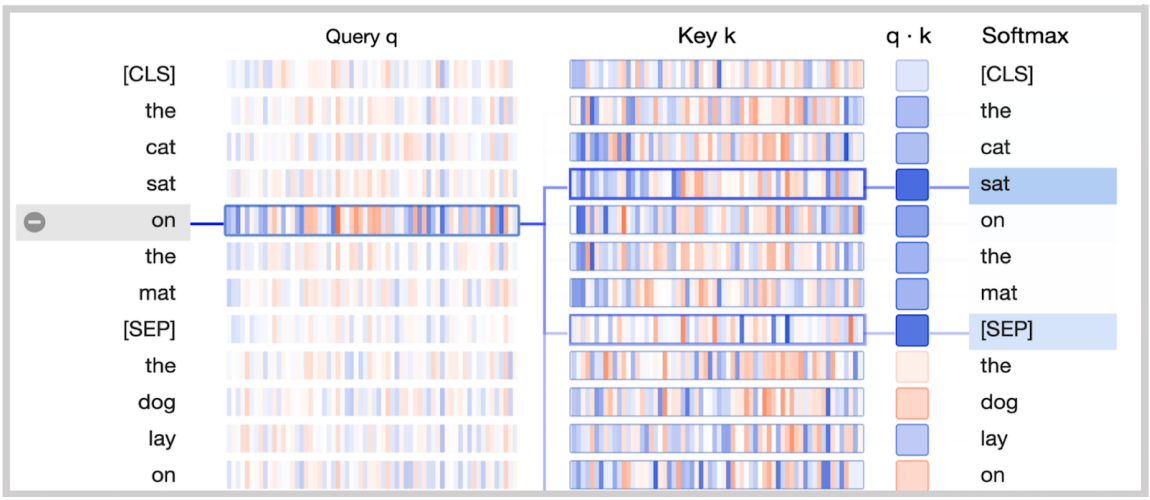

Figure 5: Figure from Vig (2019). "Neuron View" visualisation from BertViz tool. The visualisation shows the query and key vectors and shows how they are used to compute attention scores between the highlighted word "on" and the other words in the text.

3 Explanation methods adapted for text classification

Many modern deep neural networks including transformer networks (Vaswani et al., 2017) (e.g., BERT-like models) split the input text into subword tokens. This is very convenient for morphologically rich languages such as the less-resourced EMBEDDIA languages. However, perturbation-based explanation methods (such as IME, LIME, and SHAP) have problems with text input and in particular subword input, as the credit for a given output cannot be simply assigned to syntactic units such as words, phrases, or sentences.

In Section 3.1, we describe the adaptations we have introduced to these explanation methods to allow them to work with state-of-the-art text models such as BERT. In Section 3.2, we describe our initial work on making explanation methods more robust and prevent possible adversarial attacks which can mislead users applying the existing explanation methods.

3.1 IME, LIME and SHAP adapted for BERT

In this section we describe the implementation details required to make explanation methods LIME, SHAP, and IME, to work with BERT-like neural networks.

The implementation of the method LIME[4] consists of different modules that depend on the type of data. There are modules for interpreting image classification, for classification of tabular data, and for explaining text classifiers, which is the one we adapted. The LimeTextExplainer module generates artificial data in the neighborhood of the instance to be explained by randomly masking feature values and then locally training weighted linear models on the generated data to get explanations for each of the target classes. The method requires a function for splitting the instances (i.e. sentences) and a classifier function that returns probabilities. The LIME library already provides some examples, but there is no support for BERT-like models that use subword input. We implemented custom functions for preprocessing the input data for LIME, to get the predictions from the BERT model, and to prepare the data for the visualization.

The input preprocessing of different input texts is specific to their characteristics. For example, for tweets we split input instances using the TweetTokenizer function from NLTK library[5], and omitted some of the punctuation marks. We removed periods, commas, colons, semicolons, apostrophes and quotation marks, but included exclamation and question marks, and also all of the emojis. The result is a list of sentence fragments which serves as a basis for the word masking. The sentence fragments consist of words, punctuation and emojis. This preprocessed sentence is further processed by the classifier prediction function, which we modified to use the BERT tokenizer that converts the sentence fragments to sub-word tokens and then to indexes. The two custom functions are fed to the LimeTextExplainer along with the text data we want to interpret. With this modification, LIME gets the desired predictions from the BERT model for the new locally generated instances. This enables it to compute the features' impact on the prediction.

The implementation of the SHAP method[6] includes several explanation algorithms that are either model-agnostic (i.e. Kernel SHAP) or adjusted to specific machine learning approaches, i.e. a specific type of tree ensembles and a specific architecture of deep neural networks. In our work, we adapted model-agnostic SHAP, which is similar to LIME concerning the required functions. With minor modifications

[4] https://github.com/marcotcr/lime
[5] https://www.nltk.org
[6] https://github.com/slundberg/shap

of functions implemented for LIME, we constructed a data preprocessing pipeline for word perturbations that are needed in the SHAP explanation method. The classifier prediction function needed no modifications.

Since we use our own implementation of the IME method, we did not have the same restrictions as with LIME and SHAP. We reconstructed the algorithm to allow adjustment in the sampling process for perturbed feature values. The functions for the preprocessing step in which we split the instances to sentence fragments and prepare inputs for the BERT model are the same as the ones used for LIME. As mentioned in Section 2.2, one of the main implementation differences between LIME and IME is the way they generate artificial data around the explained instance. While LIME modifies the instance by randomly hiding feature values, IME replaces them with values randomly sampled from the feature space. For textual data, this is the word space, i.e. vocabulary in bag-of-words representation. The vocabulary we used for the sampling process was constructed from the dataset of English tweets which was initially used to fine-tune the BERT model. The dataset was tokenized with the same TweetTokenizer function from the NLTK library as in the LIME method.

While currently our adaptation[7] only supports random sampling from the word space, we intend to improve it by taking into account specific properties of text data and apply language models in sampling. As random sampling generates mostly unintelligible and grammatically wrong sentences, the resulting explanations based on such artificial sentences with little meaning are likely to fail in capturing the true impact of the individual words on the prediction. We plan to restrict the sampling candidates for each word only to the ones that are appropriate considering the part of speech and general context of the sentence. Better sampling will also improve the speed of explanations as it will decrease the variance of explanations.

3.2 Robustness of explanations and malicious attacks

Recent research has shown that existing explanation methods that internally generate additional samples, such as IME, LIME and SHAP, are susceptible to adversarial attacks which can trick users into believing that irrelevant attributes are responsible for the given prediction (Slack et al., 2020). We analyzed the robustness and adequacy of sampling in these approaches and proposed a more robust explanation approach that uses our recent Monte Carlo dropout autoencoder-based data generators (Miok et al., 2019) (included in Appendix A) and our previously developed generators based on RBF networks (Robnik-Šikonja, 2016). This work is still in progress and we will fully report on it in the final deliverable of this task (D1.9).

4 Contributions to visualization techniques for text classification

In this section we explain our contributions in visualisation techniques for text classification with deep neural networks. First, in Section 4.1 we present the adaptations of existing visualizations for individual predictions of text classification models. In Section 4.2 we propose a novel visualization for explanations of predictions produced by text classification models, named TextExplainViz. In Section 4.3 we present the proposed AttViz method for visualisation of attention-based neural networks.

4.1 Visualisation of explanations adapted for text classification

To visualize the explanation of a prediction, we show relevant textual, graphical, or numerical interpretable data representations that provide insight into the inner mechanisms of the model and allow for

[7] https://github.com/enjakokalj/interpret_BERT

qualitative understanding of the relationship between the instance's features and the model's prediction. It is important to distinguish between actual input features to the prediction model and the so-called interpretable versions of the features that are meaningful and understandable to humans. For example, in text classification, the prediction models use word embeddings as the actual features, while the interpretable features are usually composed of binary vectors that indicate whether a specific word is present in the input or not. A typical explanation consists of a list of the most relevant interpretable features (e.g., words) that impacted the prediction, either supporting a given class or opposing it. The direction of the impact is usually represented with different colours and is supplemented with numerical values that correspond to the magnitude of the impact.

In this section we present visualizations of explanations adapted for BERT-like prediction models, as presented in Section 3.1. We first briefly describe the dataset used and training of the BERT model used in explanations, followed by the original visualizations used in LIME and SHAP.

4.1.1 Datasets and models

To demonstrate explanations of BERT model, we used our trilingual CroSloEngual BERT model, pretrained on large corpora of English, Croatian, and Slovene texts in Task 1.2 of WP1 (Ulčar & Robnik-Šikonja, 2020). This BERT model is aimed at good model transfer between the three languages involved. We demonstrate its behavior on the sentiment prediction problem. We fine-tuned the CroSloEngual BERT model on a dataset of sentiment annotated English tweets (Mozetič et al., 2016). The dataset contains roughly 88,000 English tweets with sentiment labels that were assigned by human annotators. The model achieved an accuracy of 66.60%. The macro average of F1 score was 66.38%, while F1 scores per class were 65.56% (positive), 67.92% (neutral) and 65.66% (negative).

4.1.2 LIME visualizations adapted for BERT

The LIME method uses bar charts to visualize the explanations of predictions. For text classification, the bars represent the impact on prediction of the most relevant words in descending order of impact. The most relevant words are highlighted in the accompanying text. The impact bars in the chart are shown with two different colours, which correspond to the colour of the target class an individual word contributes to. Figures 6 and 7 show an example of explanation visualization with LIME for positive and negative sentiment, respectively. We used multi-class sentiment analysis classification using the BERT model described in Section 4.1.1. On the left-hand side, the prediction probabilities are presented; in the middle, the horizontal bar chart shows the words sorted by relevance, and on the right-hand side we can see an input text with the words highlighted according to their importance (i.e. weights obtained from the weighted linear regression). For example, in Figure 7 green words indicate negative sentiment, while blue words indicate any of the other two classes. The weights correspond to probabilities. For example, in model explained in Figure 6 if we remove the word "kiss", the prediction probability for the positive class of the explained sentence drops by 0.28.

4.1.3 SHAP visualizations adapted for BERT

The SHAP method visualizes explanations by plotting the features' contributions to the prediction as the difference between the default classifier and the actually predicted class probability. The default value represents the predicted value of a model if all features are ignored (i.e. the average model output for the training set). The features that contribute to the predicted class label are shown on the right-hand side from the baseline in red, and the ones that oppose the predicted class are shown in blue left from the baseline. Figures 8 and 9 show an example of explanation visualization with SHAP for positive and negative sentiment, respectively. We use the same sentences shown above in Figures 6 and 7 and predict with the same BERT model as above with the LIME explanation method. For example, in Figure 8, the features with strongest impact on the prediction correspond to longer arrows that point

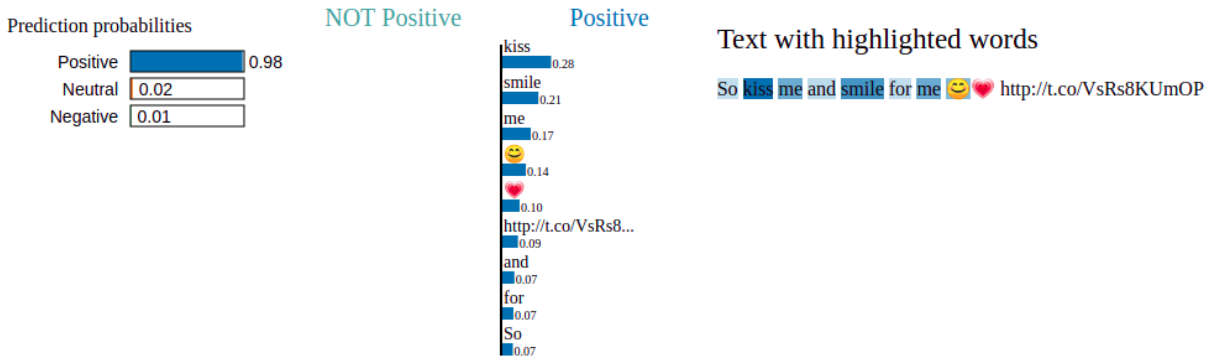

Figure 6: Visualization of prediction explanations with LIME for positive sentiment.

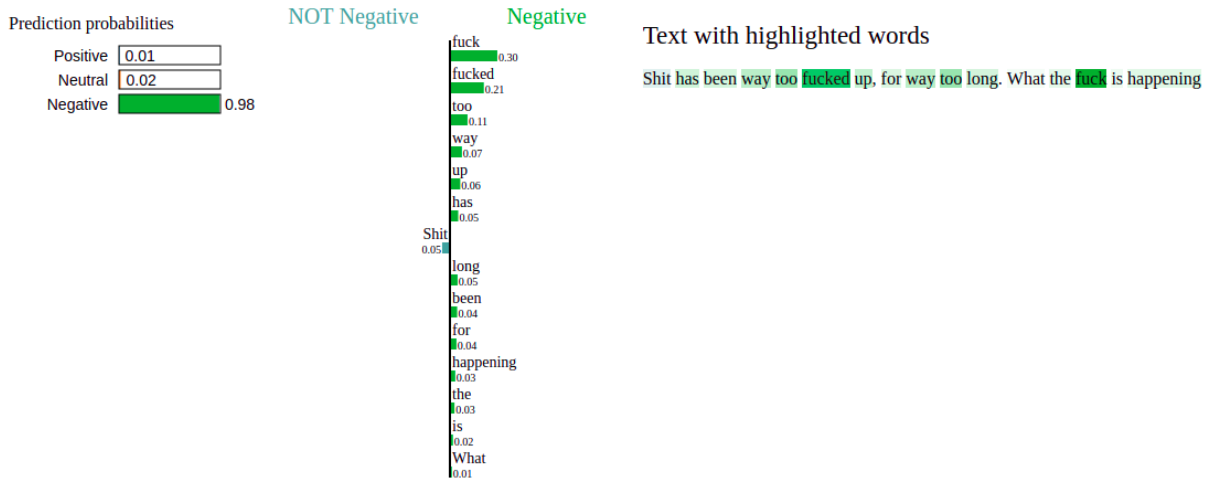

Figure 7: Visualization of prediction explanations with LIME for negative sentiment.

in the direction of the predicted class (shown in red on the right-hand side from the base value). The contributions of all features sum up to the difference between the default prediction and the output for the specific instance. If there were any features that significantly opposed the predicted class, they would be shown in blue on the left-hand side from the base value.

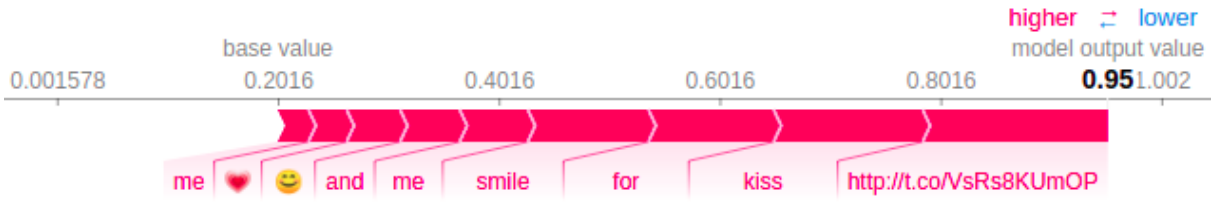

Figure 8: Visualization of prediction explanations with SHAP for positive sentiment.

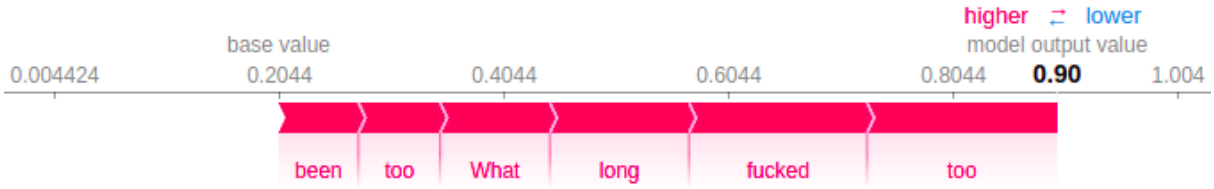

Figure 9: Visualization of prediction explanations with SHAP for negative sentiment.

4.2 TextExplainViz visualization of explanations for text classification

The newly proposed vizualisation method TextExplainViz[8] is general and applicable to all above mentioned explanation techniques (IME, LIME, and SHAP); nevertheless, we demonstrate its potential for visualizations of the state of the art BERT model.

To make visualization of predictions better adapted to texts, we modified the visualizations used in LIME and SHAP (shown in Figures 6-9). Figure 10 is an example of our visualization for explaining text classifications. It was inspired by the visualization used by the LIME method shown in Figures 6 and 7. We made some modifications to make the explanation more intuitive. Instead of the horizontal bar chart of features' impact on the prediction sorted in descending order, we used vertical bar chart and presented the features (i.e. words) in the order they appear in the original sentence. In this way, the graph allows the user to compare the direction of the impact (positive/negative) and also the magnitude of impact for individual words. The bottom text box representation of the sentence shows the words coloured green if they significantly contributed to the prediction and red if they significantly opposed it. If the contribution was insignificant, the colour is absent. The significant impacts are discretized into two intervals according to the predetermined thresholds, where darker colour represents more influential words and lighter colour less influential ones.

While the LIME visualization includes adjustments for text data, the SHAP visualization shown in Figures 8 and 9 is primarily designed for explanations of tabular data and images. As explanations for text data are represented in the same way, they are unintuitive and sometimes hard to understand. The feature contributions are shown with arrows corresponding to the direction of the impact and are labeled with the feature name and value. The features (i.e. words) are ordered by contribution and only the most important ones are shown. Since the graph is not supplemented with the original sentence, it is hard to make sense of the random words the algorithm recognized as important.

In the future, we intend to merge the two graphs (bar chart and text box representations) by supplementing each feature's text box with a corresponding vertical bar that represents whether the word had a positive or negative impact on the prediction and also how big the impact is. We think this would be more intuitive for textual data.

4.3 AttViz: Online toolkit for visualization of self-attention

Modern NLP techniques make use of *transfer learning* via large pre-trained models—deep neural network architectures that have gone through extensive unsupervised pre-training in order to capture context-dependent meaning of individual tokens (Devlin et al., 2019; Y. Liu et al., 2019; Z. Yang et al., 2019). Even though pre-training of such multi-million parameter neural networks can be expensive (Radford et al., 2019), many pre-trained models have been made freely available to the wider research community, unveiling the opportunity for the exploration of how, and why such large models perform well. One of the main problems with neural language models is their lack of *interpretability*.

[8] https://github.com/enjakokalj/interpret_BERT

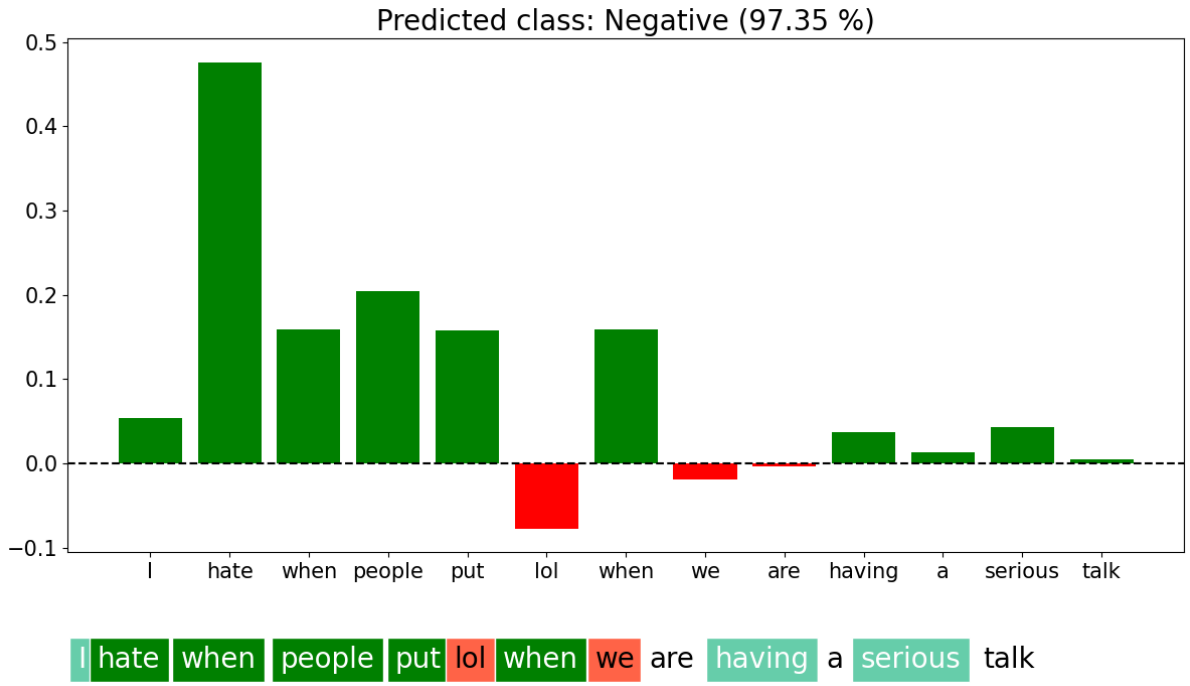

Figure 10: An example of our approach to visualization of prediction explanations for negative sentiment. We obtained the features' contribution values with the LIME method. It is evident that the word "hate" strongly contributed to the negative sentiment classification, while the word "lol" (laughing out loud) slightly opposed it.

A potential way of extracting the token relevance is the attention mechanism (Bahdanau et al., 2014; Luong et al., 2015). The attention mechanism learns token pair-value mappings, potentially encoding *relations* between token pairs. When inspected as self-relations, the attention of a token w.r.t. itself (the diagonal element of the token attention matrix) potentially offers some insight into the importance of that token. Similar findings were also recently discussed when considering tabular data (Arik & Pfister, 2019).

However, analytically, exploration of attention can be a time consuming task requiring the investigation of attention across a large number of attention heads and layers. This difficulty in the analysis of attention has resulted in the rise of approaches aimed at *attention visualization*. Visualization of (latent) embedding spaces is becoming ubiquitous in contemporary machine learning. For example, Google's Embedding Projector[9] has offered numerous visualizations for non-savvy users, making embedding projections to low dimensional (human-understandable) vector spaces simple and available online. Even though visualization of simple embedding spaces is already accessible, visualization of complex neural network models' interior representations distributed across multiple embeddings (e.g., attention vectors), however, can be a challenging task. The works of S. Liu et al. (2018a) and Yanagimto et al. (2018) are examples of attempts at unveiling the workings of black-box attention layers and offering an interface for human researchers to learn and inspect their models. Further, Yanagimto et al. (2018) visualized self-attention with examples in sentiment analysis.

We built AttViz, an online solution that can be coupled with existing language models from the PyTorch-transformers library[10]—one of the most widely used resources for language modeling. The idea behind AttViz is that it is *lightweight*, as it does not offer (online) neural model training, but facilitates the exploration of *trained* models. Along with AttViz, we provide a set of Python scripts that take as an input a trained neural language model and output a JSON file to be used by AttViz visualisation tool.

[9] https://projector.tensorflow.org/
[10] https://github.com/huggingface/transformers

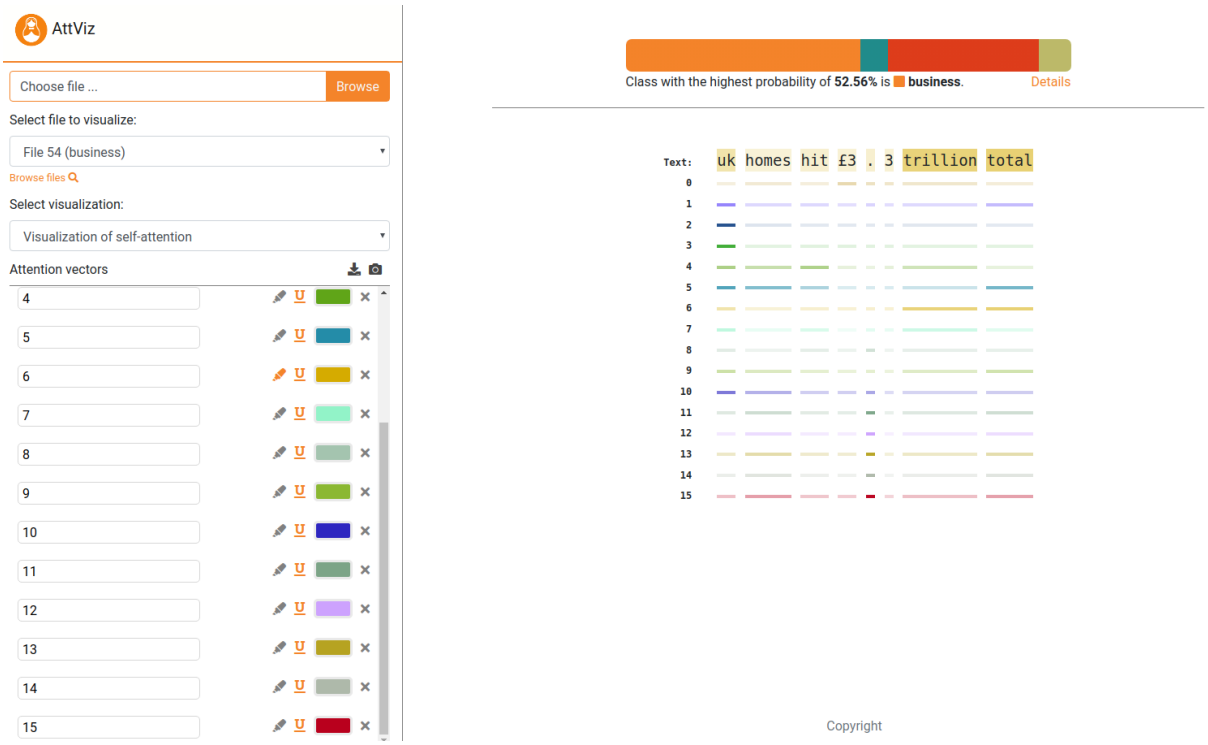

Figure 11: Visualization of all attention heads. The sixth heads's self attention is also used to highlight the text. The document was classified as a business-related, which can be linked to high self attention at the "trillion" and "uk" tokens. Note also that, the network is not *certain* about the class label – the business and politics classes were predicted with similar probabilities (orange and red parts of the bar above visualized text).

AttViz focuses exclusively on self-attention and introduces two novel ways of visualizing this property while being available online and accessible to a wider audience. AttViz can interactively aggregate the attention vectors and offers simultaneous exploration of the output probability space, as well as the attention space. A common pipeline for using AttViz is as follows. First, a transformer-based trained neural network model is used to obtain predictions on a desired set of instances (texts). The predictions are converted into the JSON format, suitable for AttViz, along with the attention space of the language model. The JSON file is loaded into AttViz (on the user's machine client side), where its visualization and exploration is possible. We next discuss the proposed visualization of the self-attention.

4.3.1 Visualization of self-attention

The initial AttViz view offers sequence-level visualization, where each (byte-pair encoded) token is equipped with a self-attention value based on a given attention head (see Figure 11; central text space). Following the first row that represents the input text, consequent rows correspond to attention values that represent the importance of a given token with respect to a given attention head.

The rationale for this display is that commonly, only a certain number of attention heads are activated (coloured fields), thus visualization must encompass both the whole attention space, as well as emphasize individual heads (and tokens). In the example in Figure 11, we visualize a short segment related to UK homes and spending. Note that the text is shown after the preprocessing consisting of byte-pair encoding and lower-casing. The segment was correctly classified as business-related. Tokens such as "trillion", "uk" and "total" are all associated with high attention. The example shows how different attention heads detect different aspects of the sentence, even at the single token (self-attention) level. The user can observe that the next most probable category for this topic was politics (red colour), which

is indeed a more sensible classification than e.g., sports. The example shows how interpretation of the attention can be coupled with the model's output for increased interpretability.

To examine of the effectiveness of the visual encoding, we can use a ranking of visual variables (Bertin, 1983) per data type, as proposed by Mackinlay (1986). In this example the quantitative values of self-attention are being rendered using the visual variable colour intensity. This is quite a low ranking variable for rendering quantitative information, this makes it difficult to estimate or compare the quantity of self attention across the heads or tokens. If visual explanations require the quick comparisons of the "amount" of self attention a different rendering may be better, using the variable's position or length to represent the quantity. However, this interface is designed to be able to render a large number of attention heads at once which would be compromised by using a more space filling visual encoding. By using the position variable to align the sentence with the self-attention quantities at each head, quick comparisons across the heads is achieved but trades off the precision of the comparisons for a more general overview.

4.3.2 Aggregation of self-attention

The visualisation of self-attention seen in Figure 11 was extended to allow for the exploration of token-based self-attention using a variety of aggregation schemes. In Figure 12, this new rendering can be seen. The leftmost part shows (by id) individual self-attention vectors, along with options to select the desired visualization, aggregation and file. The file selection indexes all examples contained in the input (JSON) file. Attention vectors can be coloured with custom colours, as shown in the central (token-value) view. The user can observe that, for example, the violet aggregation vector is active, and emphasizes tokens such as "development" and "next" under an element-wise maximum aggregation. Further, the upmost visualization in the right part of the view shows probabilities (obtained via softmax normalization of the output layer weights) of the considered document belonging to a given class. This functionality was added to help human explorers investigate the correspondence between the actual classification and the classified text. Using the "Details" section below the probability legend, the distributions across the class space can be further inspected.

In order to achieve this visualisation we apply several aggregation schemes across the space of individual tokens. Consider a matrix $A \in \mathbb{R}^{h \times t}$, where h is the number of attention vectors and t the number of tokens. We consider various aggregations across the second dimension of the attention matrix A (index j). In entropy based calculation, we denote with P_{ij} the probability of observing A_{ij} in the j-th column. The m_j corresponds to the number of unique values in that column. The proposed schemes are summarized in Table 1.

Table 1: Aggregation schemes used in AttViz.

Aggregate name	Definition
Mean(j) (mean)	$\frac{1}{h}\sum_i A_{ij}$
Entropy(j) (ent)	$-\frac{1}{m_j}\sum_{i=0}^{h} P_{ij} \log P_{ij}$
Standard deviation(j) (std)	$\sqrt{\frac{1}{h-1}\sum_i (A_{ij} - \overline{A_{ij}})^2}$
Elementwise Max(j) (max)	$\max_i(A_{ij})$
Elementwise Min(j) (min)	$\min_i(A_{ij})$

A further rendering of self-attention which makes use of these aggregation schemes is shown in Figure 13. The visualization displays the overall distribution of attention values across the whole token space. Resembling a time series, for each consecutive token, the attention values are plotted separately. This visualization offers an insight into *self-attention peaks*, i.e. parts of the attention space focused around certain tokens that potentially impact the performance and the decision making process of a given neural network. This view also emphasizes different aggregations of the attention vector space for a single token (e.g., mean, entropy, and maximum). The visualization, apart from the mean self-attention (per token), also offers the information on maximum and minimum attention values (red

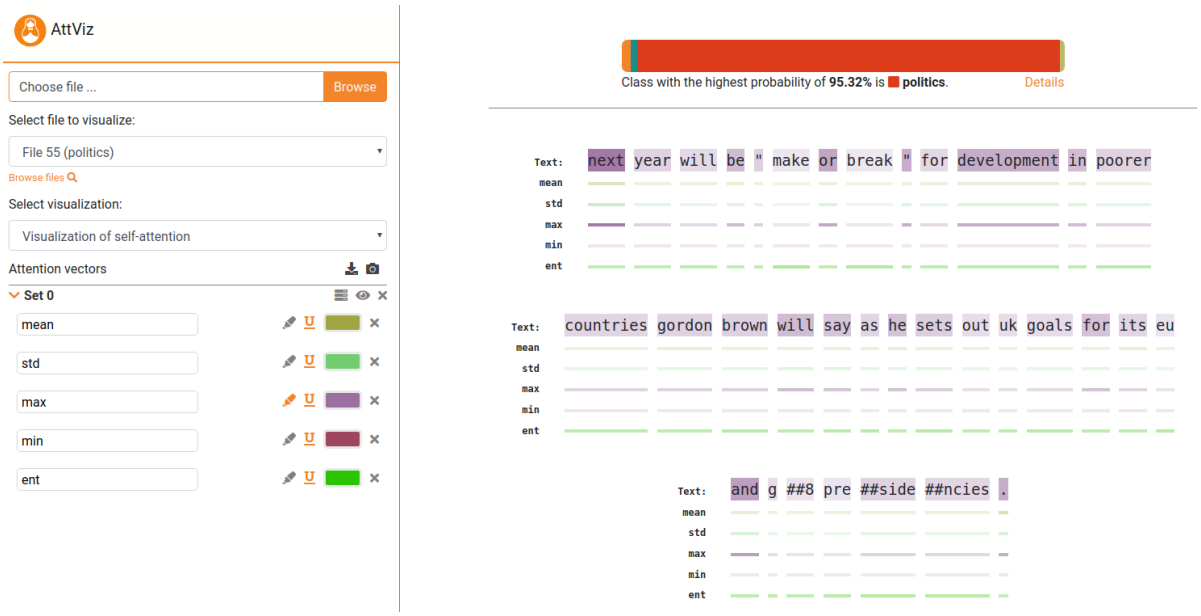

Figure 12: Visualization of aggregations. The document was classified as a politics-related topic, it can be observed that aggregations emphasize tokens such as “development”,“uk” and “poorer”. The user can also highlight desired head information – in this example the maximum attention (purple) is highlighted.

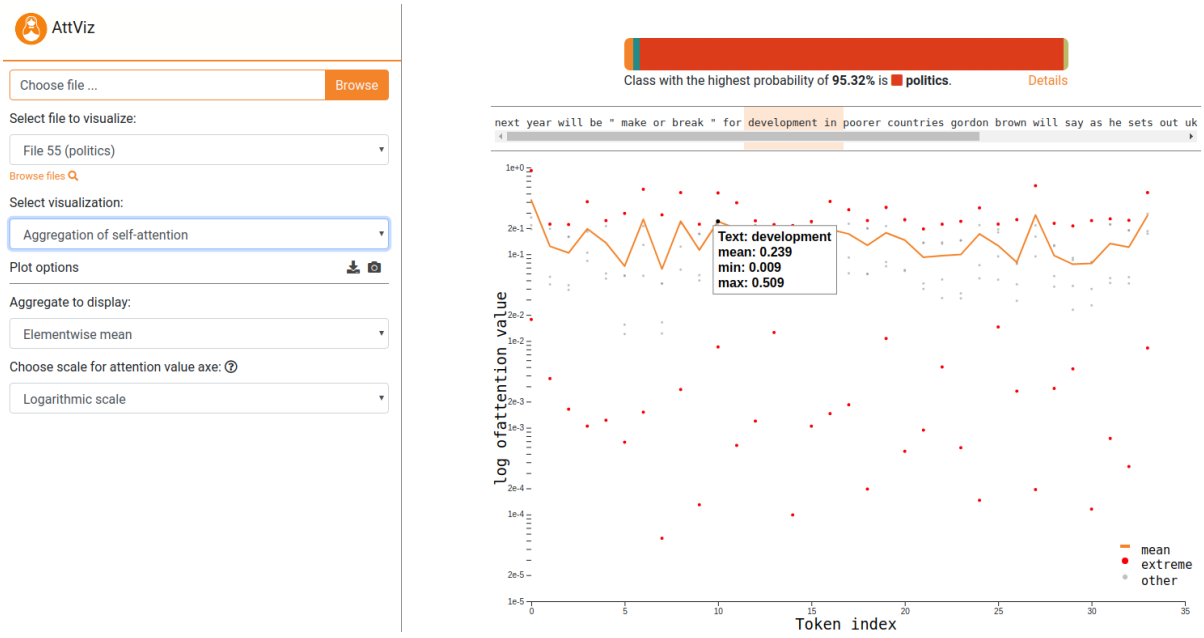

Figure 13: The interactive series view. The user can, by hoovering over the desired part of the sequence, inspect the attention values and their aggregations. The text above the visualization is also highlighted automatically.

dots), as well as the remainder of the self-attention values (gray dots). The user can this way explore both the self-attention peaks, as well as the overall attention spread. In this view the visual encoding enables easier quantitative comparisons of self-attention at the token level. The detail is greater, but the overview is reduced.

4.3.3 Comparison with state-of-the-art

An outline of similarities and differences of AttViz with other state-of-the-art visualization approaches is shown in Table 2.

Table 2: Comparison of different aspects of the attention visualization approaches.

Approach	AttViz (this work)	BertViz	neat-vision	NCRF++
Visualization types	sequence, aggregates	head, model, neuron	sequence	sequence
Open source	✓	✓	✓	✓
Language	Python + Node.js	Python	Python + Node.js	Python
Accessibility	Online	Jupyter notebooks	Online	script-based
Sequence view	✓	✓	✓	✓
Interactive	✓	✓	✓	✗
Aggregated view	✓	✗	✗	✗
Target probabilities	✓	✗	✓	✗
Compatible with PyTorch Transformers	✓	✓	✗	✗
Token-to-token attention	✗	✓	✗	✓

The main novelties introduced as part of AttViz are the capability to aggregate the attention vectors with four different aggregation schemes, offering insights both into the average attention and its variance.

The neat-vision project is the closest to AttViz's functionality, neat-vision provides a similar sequence visualisation to that found in NCRF++ (J. Yang & Zhang, 2018) but has additional interactive features. Comparing neat-vision with AttViz the following differences were observed, neat-vision is not directly bound to the PyTorch transformers library, requiring additional pre-processing on the user-side. Similarly, the fast switching between the sequence and aggregate view are more emphasized in AttViz, as they offer more general overview of the attention space. The class probabilities are to our knowledge available in both tools, offering simultaneous exploration of both input and output space at the same time.

AttViz is focused on the exploration of *self-attention* this a major difference compared with the functionality provided in the BertViz tool (Vig, 2019), which focuses on token-to-token attention patterns. We realize that the self-attention is not necessarily the only important aspect of a neural network that needs to be inspected, but it is possibly the one where visualisation techniques have been the least explored. Similarly to the work of S. Liu et al. (2018b), we plan to further explore potentially interesting *relations* emerging from the attention matrices.

The links to source code and data used for the examples, tutorial of AttViz system and live demo are contained in Section 5. The associated paper is attached as Appendix B.

5 Associated outputs

Description	URL	Availability
AttViz tutorials and code	`github.com/SkBlaz/attviz`	Public (GPL3)
AttViz visualization server	`attviz.ijs.si`	Public
MCD AE and VAE data generators	`github.com/KristianMiok/MCD-VAE`	Public (MIT)
TextExplainViz vizualization of explanations	`github.com/enjakokalj/interpret_BERT`	To become public*

* Resources marked here as "To become public" are available only within the consortium while under development and/or associated with work yet to be published. They will be released publicly when the associated work is completed and published.

Parts of this work are also described in detail in the following publications.

Citation	Status	Appendix
Miok, Kristian, Dong Nguyen-Doan, Daniela Zaharie, and Marko Robnik-Šikonja. "Generating Data using Monte Carlo Dropout". In: 2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP). IEEE. 2019, pp. 509–515.	Published	Appendix A
Škrlj, Blaž, Shane Sheehan, Nika Eržen, Marko Robnik-Šikonja, Saturnino Luz, and Senja Pollak. AttViz: Online exploration of self-attention for transparent neural language modeling. 2020.	Submitted	Appendix B

6 Conclusions and further work

We presented our initial work on explanation and visualization of prediction models for text. As currently BERT models produce state-of-the-art performance, we have modified three popular explanation methods, LIME, SHAP, and IME, to work with BERT. We describe the necessary adaptations in the explanations and visualize them with the proposed TextExplainViz approach. We show the outcome using the Twitter sentiment prediction problem, classified with our recently introduced CroSloEngual BERT produced in task T1.2 of WP1. As understanding of BERT workings is an open research question, we contribute to BERT's understanding by presenting AttViz system that visualizes BERT's self-attention heads.

In the future, we intend to improve the explanations and visualizations for text prediction by better assigning credit for predictions to larger textual units, such as n-grams and sentences. To keep the complexity of the method low, we intend to use dependency parsing which will produce sensible candidate textual units for explanation. Our aim is to detect and visualize interactions on the level of subtrees returned by the dependency parsing. As recently shown, the perturbation based explanation methods are susceptible to adversarial attacks. To prevent the attacks, we intend to improve the sampling used internally by these methods. Besides improved robustness, sampling also seems to be the key to improve the efficiency of existing explanation methods. For text classifiers, we are going to approach the problem by using pretrained language models such as BERT and ELMo.

We intend to further develop the work started with AttViz by creating further interactions and visualisations to explore potentially interesting *relations* emerging from the attention matrices. We believe AttViz could be further extended with a larger database of popular models and a back-end functionality, enabling it to e.g., fine-tune models.

Some ideas which will be further explored include:

- visualizing embeddings and visual embedding comparison,
- parallel visualization of (cross-lingual) embeddings,
- explanation and visualisations based on sentence/paragraph level features.

The produced explanation and visualization technologies will contribute to better interpretation of deep learning models developed in T1.3 that will be applied in WP3, WP4, and WP5. The software implementations will be integrated into the Media Assistant developed in WP6.

References

Arik, S. O., & Pfister, T. (2019). TabNet: Attentive Interpretable Tabular Learning. *arXiv preprint arXiv:1908.07442*.

Arras, L., Horn, F., Montavon, G., Müller, K.-R., & Samek, W. (2017). What is relevant in a text document?: An interpretable machine learning approach. *PloS ONE*, *12*(8), e0181142.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bertin, J. (1983). *Semiology of Graphics*. University of Wisconsin Press.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lee, J., Shin, J.-H., & Kim, J.-S. (2017). Interactive Visualization and Manipulation of Attention-based Neural Machine Translation. In *Proceedings of the 2017 conference on empirical methods in natural language processing: System demonstrations* (pp. 121–126).

Lemaire, V., Féraud, R., & Voisine, N. (2008). Contact Personalization using a Score Understanding Method. In *Proceedings of International Joint Conference on Neural Networks (IJCNN)*.

Liu, S., Li, T., Li, Z., Srikumar, V., Pascucci, V., & Bremer, P.-T. (2018a). Visual Interrogation of Attention-Based Models for Natural Language Inference and Machine Comprehension. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 36–41).

Liu, S., Li, T., Li, Z., Srikumar, V., Pascucci, V., & Bremer, P.-T. (2018b). *Visual interrogation of attention-based models for natural language inference and machine comprehension* (Tech. Rep.). Livermore, CA (United States): Lawrence Livermore National Lab.(LLNL).

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems* (pp. 4768–4777).

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, *5*(2), 110–141.

Meyer, D., Leisch, F., & Hornik, K. (2003). The support vector machine under test. *Neurocomputing*, *55*, 169-186.

Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *arXiv preprint 1309.4168*.

Miok, K., Nguyen-Doan, D., Zaharie, D., & Robnik-Šikonja, M. (2019). Generating Data using Monte Carlo Dropout. In *2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP)* (pp. 509–515).

Mozetič, I., Grčar, M., & Smailović, J. (2016). Multilingual Twitter sentiment classification: The role of human annotators. *PLoS ONE*, *11*(5), e0155036.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (pp. 2227–2237).

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8).

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of ACM SIGKDD* (pp. 1135–1144).

Robnik-Šikonja, M. (2016). Data generators for learning systems based on RBF networks. *IEEE transactions on neural networks and learning systems*, *27*(5), 926–938.

Robnik-Šikonja, M., & Kononenko, I. (2008). Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, *20*(5), 589-600.

Rocktäschel, T., Grefenstette, E., Hermann, K. M., Kociský, T., & Blunsom, P. (2015). Reasoning about Entailment with Neural Attention. *CoRR*, *abs/1509.06664*.

Rush, A. M., Chopra, S., & Weston, J. (2015). A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 379–389).

Shapley, L. S. (1953). A value for n-person games. In H. Kuhn & K. Tucker (Eds.), *Contributions to the Theory of Games, Vol. II* (pp. 307–317). Princeton University Press.

Slack, D., Hilgard, S., Jia, E., Singh, S., & Lakkaraju, H. (2020). Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (pp. 180–186).

Strobelt, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H., & Rush, A. M. (2018). Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. *CoRR*, *abs/1804.09299*.

Tsaptsinos, A. (2017). Lyrics-Based Music Genre Classification Using a Hierarchical Attention Network. *CoRR*, *abs/1707.04678*.

Ulčar, M., & Robnik-Šikonja, M. (2020). FinEst BERT and CroSloEngual BERT: less is more in multilingual models. In *Proceedings of Text, Speech, and Dialogue, TSD 2020.* (accepted)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vig, J. (2019). Visualizing Attention in Transformer-Based Language Representation Models. *CoRR*, *abs/1904.02679*.

Škrlj, B., Sheehan, S., Eržen, N., Robnik-Šikonja, M., Luz, S., & Pollak, S. (2020). *AttViz: Online exploration of self-attention for transparent neural language modeling.* (submitted)

Štrumbelj, E., & Kononenko, I. (2010). An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, *11*(Jan), 1–18.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2018). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (pp. 353–355).

Yanagimto, H., Hashimoto, K., & Okada, M. (2018). Attention Visualization of Gated Convolutional Neural Networks with Self Attention in Sentiment Analysis. In *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)* (p. 77-82).

Yang, J., & Zhang, Y. (2018). NCRF++: An Open-source Neural Sequence Labeling Toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics.*

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems* (pp. 5754–5764).

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1480–1489). Association for Computational Linguistics.

Appendix A: Generating Data using Monte Carlo Dropout

Generating Data using Monte Carlo Dropout

Kristian Miok
West University of Timisoara
Computer Science Department
Romania
Email: kristian.miok@e-uvt.ro

Dong Nguyen-Doan
West University of Timisoara
Computer Science Department
Romania
Email: dong.nguyen10@e-uvt.ro

Daniela Zaharie
West University of Timisoara
Computer Science Department
Romania
Email: daniela.zaharie@e-uvt.ro

Marko Robnik-Šikonja
University of Ljubljana
Faculty of Computer and Information Science
Slovenia
Email: marko.robnik@fri.uni-lj.si

***Abstract*—For many analytical problems the challenge is to handle huge amounts of available data. However, there are data science application areas where collecting information is difficult and costly, e.g., in the study of geological phenomena, rare diseases, faults in complex systems, insurance frauds, etc. In many such cases, generators of synthetic data with the same statistical and predictive properties as the actual data allow efficient simulations and development of tools and applications. In this work, we propose the incorporation of Monte Carlo Dropout method within Autoencoder (MCD-AE) and Variational Autoencoder (MCD-VAE) as efficient generators of synthetic data sets. As the Variational Autoencoder (VAE) is one of the most popular generator techniques, we explore its similarities and differences to the proposed methods. We compare the generated data sets with the original data based on statistical properties, structural similarity, and predictive similarity. The results obtained show a strong similarity between the results of VAE, MCD-VAE and MCD-AE; however, the proposed methods are faster and can generate values similar to specific selected initial instances.**

I. Introduction

We live in times of big data; yet, there are many application areas that lack sufficient data for analyses, simulations, and development of analytical approaches. For example, many studies within bio-medical domain require strict and expensive experimental conditions and can produce only small samples within the allocated budget. Similar examples are domains for which data is difficult to obtain, such are rare diseases, private records, or rare grammatical structures [1]. Thus, there is a need for machine learning methods that can generate new data preserving the statistical and predictive characteristics of the original data set.

Since its introduction by Diederik et al. [2], Variational autoencoders (VAE) become one of the most used unsupervised learning methods within the family of autoencoder (AE) techniques [3]. They are used in various problems: predicting dense trajectories of pixels in computer vision [4], anomaly detection [5], and conversion of molecular discrete representations to and from multidimensional continuous representations [6]. A short description of VAEs is provided in Section 3. Our interest in VAEs is due to their ability to generate new data [7, 8].

978-1-7281-4914-1/19/$31.00 ©2019 IEEE

The main goal of this work is to introduce Monte Carlo dropout into (variational) autoencoder-based data generating methods that can provide comparable results to existing VAE generators in a shorter time. To show favorable properties of the new generators, we conduct comparisons among three groups of data sets:

1) original data sets,
2) data sets produced by the VAE generator,
3) data sets generated using the newly introduced MCD-VAE and MCD-AE approaches.

We compare statistics of individual attributes in each of the data sets, structures of the data sets as determined by clustering algorithms, predictive performance of machine learning algorithms trained and tested on data sets from each group, and times required for generation of new instances.

The outline of the paper is as follows. In Section 2, we shortly discuss related work. In Section 3, we introduce the methodology and architecture of our methods. Section 4 describes how the VAE, MCD-VAE and MCD-AE generators were compared followed by the results obtained in Section 5. We compare the computational performance of the three generators in Section 6 and derive conclusions in Section 7.

II. Related Work

Methods that learn the distribution from existing data in order to generate new instances are of recent interest to scientific community. Till recently, generative methods were based on models that provide a parametric specification of a probability distribution function and models that can estimate kernel density [9]. For example, [10] and Yang et al. [11] used kernel density estimation to generate new virtual instances. However, those methods work only for data sets

with low dimensionality. An interesting method that generates new records using an evolutionary algorithm was proposed in [12]. This method does not take dependencies between attributes into account. The generator based on Radial Basis Function (RBF) networks [1] corrects this shortcoming but is less suitable for really high dimensional data sets (such as images and text). Two popular generators for images are VAEs [13] and Generative Adversarial Networks (GAN) [9]. Interesting combinations of those two methods were proposed by Larsen et al. [14] and Rosca et al. [15] suggesting that a GAN discriminator can be used in place of a VAEs decoder.

As the GAN generated data that can be very different from the original data set its outputs cannot be used to simulate the original data. On the other hand, the shortcoming of VAE is that the newly generated values strongly depend on the distribution of the whole training set. Hence, in case we want to generate instances similar to specific instances, e.g., outliers, this is impossible. The proposed method addresses the mentioned shortcomings of VAEs and improves upon it in terms of flexibility of the generated instances and speed of generation.

III. Methods

We first present the background information on AE, VAE and Monte Carlo Dropout method and then explain how we can harness the power of both to produce flexible and efficient data generators. Finally, we visually demonstrate the differences between different generators on a digit recognition data set.

A. (Variational) Autoencoders

A typical AE is made of two neural networks called an encoder and a decoder. The encoder compresses the data into an internal representation and the decoder tries to decompress from this compressed representation (or latent vector) back into the original data using a reconstruction loss function [16]. VAEs inherit the architecture of classical AEs introduced by Rumelhart at al. [17]; however, their learning process uses the data to explicitly estimate *the distribution* from which the latent space is sampled [3]. Hence, VAEs store the latent variables in the form of probability distributions. As depicted in Fig. 1, VAEs resample latent values z from the generated distribution that are further transformed using the decoder network. From the Bayesian perspective the encoder is doing an approximation of the posterior distribution $p(z|x)$:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)},$$

where z denotes the hidden variable values and x the input data. As this distribution usually does not have analytical closed form solution, we have to approximate it. In order to avoid computationally expensive sampling procedure like Markov Chain Monte Carlo (MCMC) sampling, the Variational Inference (VI) method is applied. The VI method [18] samples from the distribution for which the Kullback-Leibler divergence to the posterior distribution is minimal.

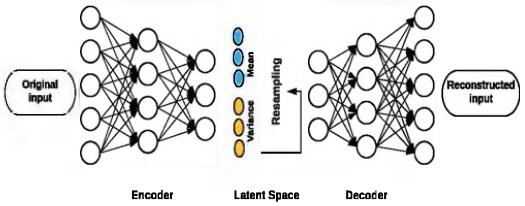

Fig. 1: Variational Autoencoder Diagram.

B. Monte Carlo Dropout Method

Deep learning is the state-of-the-art approach for many problems where machine learning is applied. However, standard deep neural networks do not provide information on reliability of predictions. Bayesian neural networks (BNN) can overcome this issue by probabilistic interpretation of model parameters. Apart from prediction uncertainty estimation, BNNs offer robustness to overfitting and can be efficiently trained even on small data sets [19]. While there exist several BNN variants and implementations, our work is based on Monte Carlo Dropout (MCD) method proposed by Gal and Ghahramani [20]. The idea of this approach is to capture prediction uncertainty using the dropout as a regularization technique. Authors prove that the use of dropout in NNs can be seen as a Bayesian approximation of the Gaussian process probabilistic models. Generating new values can be seen as the uncertainty estimation process of predicti the original instance for which generation is done [21]. The generated values shall reflect the distributional properties of the original instances.

The bias in the prediction accuracy can come from different sources. Based on where uncertainty is coming from, we distinguish: model uncertainty, data uncertainty, and distributional uncertainty. Model uncertainty describes how well the model fits the data and it can be reduced using larger training set. The data uncertainty is caused by the nature of the data set used and is irreducible by current techniques. Distributional uncertainty arises from the distributional incompatibility between the training and testing data sets. In case of the Bayesian inference, the overall uncertainty is captured with the data and model uncertainty [22]. The prediction uncertainties within the Bayesian framework can be summarized with the posterior predictive distribution (PPD) [23]. Once the posterior distribution is estimated, the PPD can be calculated using the formula:

$$p(y^*|x^*, X, Y) = \int p\big(y^*|f^\omega(x^*)\big)\; p(\omega|X, Y)d\omega$$

where the $p\big(y^*|f^\omega(x^*)\big)$ is the likelihood function that contains the data uncertainty while the $p(\omega|X, Y)$ is the posterior distribution of the model parameters ω presenting uncertainty of the model.

The idea of MCD method is to replace the complex Bayesian process of seizing those uncertainties during the

regularization using dropout. Practically, the dropout is equivalent to several forward passes through the network and recalculation of the results. At each backward pass, the model ends-up with new optimization results of the model weights. Keeping all this information, the method mimics the Bayesian inference and is equivalent to the Bayesian posterior distribution estimation [24].

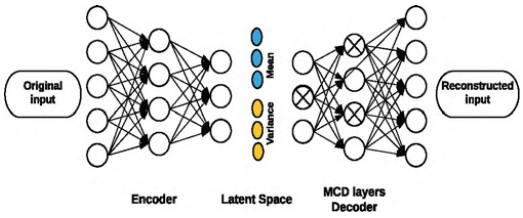

Fig. 2: Variational Autoencoder with MCD Decoder. Note the difference to Figure 1.

C. VAEs for Data Generation

For the VAE architecture (Figure 1), we use two intermediate layers (fully-connected layers) with size of M (e.g. 512) and N (e.g. 256) in the encoder. Similarly, the decoder contains two fully-connected layers with N and M neurons. To take into account various types of data sets used in our experiments, we choose the number of latent variables L to be equal to one-half of attributes present in each data set. This value is chosen in order to keep an important part of the information from which the new data can be generated.

There are two approaches to generate the data from the VAE, once the model is trained. The first approach is to generate the sampled latent vectors from the estimated normal distribution (μ, Σ) where the Σ is a diagonal covariance matrix. The sampled values are then sent through the decoder part to get the final generated instances. The second approach is to send existing instances through the trained encoder and decoder layers. In this paper, we are interested to generate new values similar to existing values present in the training set, therefore we focus on the second approach.

The process of generating data using the VAE method can be described as follows.

1) Obtain the distribution of latent vectors (μ_i, σ_i) with $i = 1, \ldots, L$ from each value in the seeding data set by using the encoder.
2) Resample t times from the obtained latent space distribution, where t is the number of new instances we want to generate for a single seeding instance as in following equation:

$$z_i = \mu_i + \sigma_i \cdot \epsilon, \text{ where } \epsilon \sim N(0, 1).$$

3) Decode the resampled values by the decoder.

D. MCD-VAE for Data Generation

The MCD-VAE architecture (Figure 2) has a similar structure to the VAE generator, with the exception that the MCD regularization is used within the decoder layers.

The process of generating data with MCD-VAE can be described as follows.

1) Obtain the distribution of latent vectors (μ_i, σ_i) with $i = 1, \ldots, L$ from each value in the seeding data set.
2) Send the means $\mu_1, \ldots, \mu_L$ through the MCD decoder t times, where t is the number of new instances we want to generate for a single seeding instance.

As evident from the above description, MCD-VAE utilizes MCD within the decoder part to get additional fine grained control over the generated instances. Namely, once the MCD-VAE is prepared for a single seeding instance, due to dropout, it can produce many different outputs by going forward through the network. This increases the speed of generation and gives the user of the generator much finer control on the generated instances.

E. MCD-AE for Data Generation

We can apply the MC dropout method also in the decoder part of AE and get the generator called MCD-AE. The structure of MCD-AE in our experiments is similar to VAE and MCD-VAE described before. The process to generate data in MCD-AE is outlined below.

1) For each value in the seeding data set, obtain latent vectors of size L.
2) Send the latent vectors through the MCD decoder. The decoder samples a new dropout mask in each of the t forward passes through the network and generates t values for a single input.

The decoder part of the MCD-AE generator is identical to the decoder in MCD-VAE. The difference between the two generators is that MCD-AE does not assume any distributional constraints for the latent space representation.

F. Visual Comparison of the Generators

We visually demonstrate the differences between the three generators (VAE, MCD-VAE, and MCD-AE). For this we have chosen a well-known MNIST data set of hand-written digits[1] and used it to train the three generators. The architecture for VAE and MCD-VAE generators contains a fully-connected layer with 1024 units and a latent layer with the size 10. The generated images are presented in Figure 3. The original seeding image is always given in the first column. In this experiment, we investigated generation of digits 9, 5, and 1 (see the three blocks of images). Digits 9 and 5 were generated from seeding instances that are written in nonstandard way, with the shape that differs from the rest of digits in their class. The digit 1 that was used as a seeding instance is written in a standard way.

The five images generated for the digit 9 using VAE (top group, first row) have the same structures as the seeding digit

[1] http://yann.lecun.com/exdb/mnist/

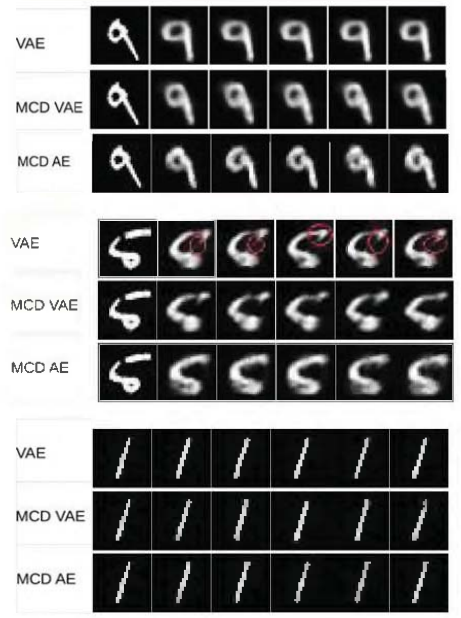

Fig. 3: The generated numbers 9, 5, and 1 are grouped in the top, middle and bottom, respectively. Each block of images contains the original seeding image (in the first column) and five generated images using VAE (the first row), MCD VAE (the second row), MCD AE (the third row).

9 but do not reflect much specifics of the seeding image. Contrarily, the images generated using MCD-VAE and MCD-AE (top group, second and third row) tend to better reflect the actual structure of the seeding images. The digit 5, used as a seeding instance in the middle group of images is a complete outlier - on the first sight one can not be sure if it is 5 or 6. The five generated images for digit 5 using VAE (middle group, first row) reflect all the training instances and do not take specifics of the seeding instance into account; hence, VAE generates images a bit similar to the digit 8. On the other hand, the images generated using MCD-VAE and MCD-AE better mimic the seeding image. The images generated from the seeding digit 1, written in the standard way, do not seem to differ much between the three generators (bottom group).

IV. Experimental setting

In this section, we first describe the methodology used to compare original and generated data in Section IV-A. We compare statistical, structural, and prediction properties of two data sets presented in Section IV-B. In Section IV-C, we present the data sets which served as original data in our evaluation.

A. Data Generation Experiment

To prepare a training data set for generators, the original data set is randomly split into two equal parts as shown in Figure 4. The first part is further split into the equal-sized training and generator seeding parts, while the second part of the original data set is left for evaluation. The training part is used to train the generators, while the generator seeding part is used in data generation. From each instance in the generator seeding set, two new instances were generated. Thus, the newly generated data sets are of the same size as the evaluation data sets.

Fig. 4: Splits of each original data set used in the experimental evaluation: the generator training set (25%), generator seeding set (25%), and evaluation set (50%).

In order to deal with multi-valued categorical attributes, we encode them with several binary substitute attributes, where the presence of a given categorical value in the original attribute sets the substitute variable corresponding to that value to 1. For example, for a multi-valued attribute X with three values $\{red, green, blue\}$ we form three substitute binary variables $X_{red}, X_{green}, X_{blue}$. If the original attribute contains value $X = blue$, the values of the substitute attributes are $X_{red} = 0, X_{green} = 0, X_{blue} = 1$. After the data is generated, we perform the reverse operation and decode the substitute variables into one multi-valued attribute.

B. Data Set Comparison

In evaluation, presented in Section V, we take an existing data set and based on it we generate three synthetic data sets, using VAE, MCD-VAE, and MCD-AE. The original and the three generated data sets are compared using a general data set comparison framework [25] which consist of three components, statistical evaluation of differences between attributes, structural comparison of data sets based on clustering, and predictive comparison based on classification models. We describe the three components below.

1) *Statistical evaluation of attributes.* test the mean, standard deviation, and differences in distributions between matching attributes in two compared data sets. In order to make comparison sensible for all statistics, the attributes are normalized to $[0, 1]$ scale. The value that summarizes the difference between the two data sets is calculated as the median value of pairwise attribute differences. For example, to compare mean across the whole data set, we compute the differences in means for each of the attributes and then average these values and report it as the final measure. We therefore report Δmean and Δstd.
2) *Clustering performance evaluation* is performed based on the structured based distance comparing two data

sets using the adjusted Rand index (ARI) [26]. The ARI value is in range of $[0, 1]$ having 0 in the case of random distributions of clusters and 1 for ideally matching clusters. The clusters of two data sets are separately computed and the process obtains the medoids for each of the clusterings. The instances in the second data set are assigned to the nearest clusters in the first data set based on the medoids computed for the first data set. The same assignment is repeated with the first data set, as instances of the first data set are assigned to clusters computed on the second data set based on the medoids from these clusters. In this way, we obtain two clusterings that contain instances from both data sets. Finally, we use ARI to summarize the clustering similarity between the two clusterings and report it as the data sets topological similarity value.

3) *Classification performance* based evaluation measures the predictive similarity of two data sets by comparing random forest classification accuracies on the two data sets. Let us assume that the original data set is denoted as d_1 and the generated data sets are labeled with d_2. Both d_1 and d_2 are split into two parts, where the first parts are used to train the random forest models m_1 and m_2, while the second parts are used for testing. Four accuracy values are computed: m_1d_1 - model computed on the first data set and evaluated on the first data set; m_1d_2 - model computed on the first data set and evaluated on the second data set; m_2d_1 - model computed on the second data set and evaluated on the first data set; and m_2d_2 - model computed on the second data set and evaluated on the second data set. If those four values are similar (in particular if accuracies on the original data set are close, i.e. accuracies of m_1d_1 and m_2d_1), one can conclude that the first and the second data set have similar predictive performance. We report only the difference of $m_2d_1 - m_1d_1$ as the predictive similarity Δacc.

C. Data Sets

To evaluate the difference between results of the three generators, we use data sets from UCI (University of California Irvine) repository [27]. The R package readMLDATA [28] was used for data manipulation. We selected classification data sets with between 500 and 1000 instances. The characteristics of the used data sets are provided in Table I.

TABLE I: The characteristics of the used data sets. The columns are: n - number of instances, a - number of attributes, num - number of numeric attributes, disc - number of discrete attributes, v/a - average number of values per discrete attribute, C - number of class values, majority % - proportion of majority class in percentages, missing % - percentage of missing values.

Data set	n	a	num	disc	v/a	C	majority (%)	missing (%)
Brest-WDBC	569	30	30	0	0.0	2	62.7	0.00
Brest-WISC	699	9	9	0	0.0	2	65.5	0.25
Credit-screening	690	15	6	9	4.4	2	55.5	0.64
PIMA-diabetes	768	8	8	0	0.0	2	65.1	0.00
Statlog-German	1000	20	7	13	4.2	2	70.0	0.00
Tic-tac-toe	958	9	0	9	3.0	2	65.3	0.00

V. Evaluation and results

Using the above described data sets we evaluated the quality of data generators. In Table II we compare the original data set with the generated data set using VAE architecture. The results comparing the original data set with the MCD-VAE and MCD-AE generators are presented in Tables III and IV, respectively. For comparison we use the statistical, structural, and predictive criteria, described in Section IV-B, i.e. the average difference in means (Δmean) and standard deviation (Δstd), similarity of produced clusters expressed with Adjusted Rand Index (ARI), and differences in predictive accuracy Δacc ($m_2d_1 - m_1d_1$).

TABLE II: Comparison between the original data and VAE generator.

Data set	Δ mean	Δ std	ARI	Δacc
Breast-WDBC	-0.161	-0.089	0.909	-0.024
Breast-WISC	-0.069	0.001	0.970	-0.045
Credit-screening	-0.078	-0.041	0.474	-0.068
PIMA-diabetes	-0.171	-0.047	0.446	-0.015
Statlog-German	-0.040	0.040	0.167	-0.000
Tic-tac-toe	-	-	0.133	-0.092

TABLE III: Comparison between the original data and MCD-VAE generator.

Data set	Δ mean	Δ std	ARI	Δacc
Breast-WDBC	-0.045	-0.044	0.876	-0.008
Breast-WISC	0.011	0.013	0.916	-0.011
Credit-screening	-0.028	-0.038	0.447	-0.061
PIMA-diabetes	-0.022	-0.028	0.715	-0.007
Statlog-German	-0.016	0.028	0.243	-0.001
Tic-tac-toe	-	-	0.122	-0.017

TABLE IV: Comparison between the original data and MCD-AE generator.

Data set	Δ mean	Δ std	ARI	Δacc
Breast-WDBC	-0.059	-0.048	0.746	-0.014
Breast-WISC	0.004	0.021	0.994	-0.020
Credit-screening	-0.046	-0.036	0.393	-0.072
PIMA-diabetes	-0.077	-0.037	0.551	-0.012
Statlog-German	-0.030	0.021	0.235	0.000
Tic-tac-toe	-	-	0.224	-0.158

Comparing the results in Tables II, III, and IV, we can see that differences between the original and generated data are small. There is no clear pattern which of the three generators is better. We can conclude that all of them are useful, while minor differences in the quality of the generated data may depend on the structure of a data set. However, it can be observed that MCD-VAE provide slightly better classification performance than VAE and MCD-AE based on the compared Δacc value. On the other hand, for *Breast-WDBC*, *Breast-WISC* and *Credit-screening* datasets VAE generator has the better clustering performance than the two newly introduced generators.

VI. Comparing efficiency of generators

In order to compare the data generation time (in seconds) of VAE, MCD-VAE, and MCD-AE, we measure the time for 100 repetitions of the data generating process using the above

described data sets. To get reliable measurements, we resample each seeding instance 1000 times (instead of 2 times as in the previous experiments). Table V reports the mean and standard deviation of the measured times. We generate data sets as described in Section III For VAE, the instances in seeding data sets are encoded to obtain the latent values, then the latent values are resampled and decoded. For MCD-VAE and MCD-AE, we obtain the mean values with the seeding instances and obtain the generated data using the MCD decoder.

TABLE V: Comparison of time required for data generation in seconds.

Datasets/Models	VAE [s.d.]	MCD-VAE [s.d.]	MCD-AE [s.d.]
Breast-WDBC	1.04 [0.018]	0.89 [0.022]	0.89 [0.020]
Breast-WISC	0.90 [0.019]	0.85 [0.037]	0.89 [0.011]
Credit-screening	1.00 [0.030]	0.93 [0.025]	0.94 [0.010]
PIMA-diabetes	0.91 [0.034]	0.85 [0.021]	0.85 [0.016]
Statlog-German	1.07 [0.018]	1.03 [0.018]	1.10 [0.045]
Tic-tac-toe	0.99 [0.026]	0.93 [0.039]	0.94 [0.012]

The MCD-VAE and MCD-AE generators are consistently slightly faster than the VAE generator (between 5-10%). Although the MCD-AE generator is architecturally simpler, it is not faster then the MCD-VAE generator. The datasets used are relatively small, hence, for the larger datasets, we expect larger differences.

VII. Conclusions and Further Work

We constructed and compared three generators of semi-artificial data. The VAE generator is based on the variational autoencoder architecture while the MCD-AE and MCD-VAE employ Monte Carlo dropout within autoencoders and variational autoencoders. The comparison of the generated data sets based on statistical, structural, and predictive properties shows that the three generators produce similar data sets which are highly similar to the original data.

The advantages of the proposed Monte Carlo dropout employed within VAE and AE over the existing VAE method can be summarized with the following two points:

- Improved speed. Based on the results presented in Table V we can conclude that generating data using MCD-VAE and MCD-AE is slightly faster than using the VAE generator.
- Greater flexibility. The MCD-VAE and MCD-AE methods generates data similar to specific selected seeding instances. This can be very useful if the provided seeding instances are outliers or instances of special interest. For example, in image generation, the newly generated images will be closer to the original one even when the original image is different from the rest of the images in the training set.

The advantage of the MCD-AE method over MCD-VAE method is that does not make any distributional assumptions during the latent space generation. The information received from the encoder part is directly introduced into the MCD decoder. The time required for data generation using MCD-AE is similar to MCD-VAE. The more detailed differences between these generators are left for further investigation.

With methodological development of deep learning, the models that can estimate the distributions, e.g., the variational autoencoders, are becoming increasingly important. Hence, our further work will focus on investigating new and improving existing architectures that can generate new data efficiently and reliably. Further, we aim to test those architectures within different application contexts. As biomedical imaging is expensive and limited by the budget, our goal is to investigate data generation within this field. The Python code of the proposed generators is publicly available[2].

Acknowledgement

The work was supported by the Slovenian Research Agency (ARRS) core research programme P6-0411 (Marko Robnik-Šikonja). The research was carried out in the frame of the project Bioeconomic approach to antimicrobial agents - use and resistance financed by UEFISCDI by contract no. 7PC-CDI / 2018, cod PN-III-P1-1.2-PCCDI-2017-0361 (Kristian Miok and Daniela Zaharie). This project has also received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 825153 (EMBEDDIA) (Kristian Miok and Marko Robnik-Šikonja).

References

[1] Marko Robnik-Šikonja. Data generators for learning systems based on RBF networks. *IEEE transactions on neural networks and learning systems*, 27(5):926–938, 2015.

[2] P Kingma Diederik, Max Welling, et al. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[3] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[4] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016.

[5] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2:1–18, 2015.

[6] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[7] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016.

[8] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled

[2] https://github.com/KristianMiok/MCD-VAE

generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[10] Der-Chang Li and Yao-San Lin. Using virtual sample generation to build up management knowledge in the early manufacturing stages. *European Journal of Operational Research*, 175(1):413–434, 2006.

[11] Jing Yang, Xu Yu, Zhi-Qiang Xie, and Jian-Pei Zhang. A novel virtual sample generation method based on gaussian distribution. *Knowledge-Based Systems*, 24(6): 740–748, 2011.

[12] Cinzia Meraviglia, Giulia Massini, Daria Croce, and Massimo Buscema. Gend an evolutionary system for resampling in survey research. *Quality and Quantity*, 40 (5):825–859, 2006.

[13] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[14] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[15] Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, and Shakir Mohamed. Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*, 2017.

[16] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[18] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[19] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning*, pages 1050–1059, 2016.

[21] Kristian Miok. Estimation of prediction intervals in neural network-based regression models. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 463–468. IEEE, 2018.

[22] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pages 7047–7058, 2018.

[23] Pavel Myshkov and Simon Julier. Posterior distribution analysis for bayesian inference in neural networks. In *Workshop on Bayesian Deep Learning, NIPS*, 2016.

[24] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

[25] Marko Robnik-Šikonja. Dataset comparison workflows. *International Journal of Data Science*, 3(2):126–145, 2018.

[26] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[27] Kevin Bache and Moshe Lichman. Uci machine learning repository, 2013. *URL http://archive. ics. uci. edu/ml*, 5, 2013.

[28] Petr Savicky. readMLData: Reading machine learning benchmark data sets from different sources in their original format, 2012.

Appendix B: AttViz: Online exploration of self-attention for transparent neural language modeling

AttViz: Online exploration of self-attention for transparent neural language modeling

Blaž Škrlj
Jožef Stefan Institute
Jožef Stefan International Postgraduate School

Nika Eržen
Jožef Stefan Institute

Saturnino Luz
University of Edinburgh

Shane Sheehan
University of Edinburgh

Marko Robnik-Šikonja
University of Ljubljana

Senja Pollak
Jožef Stefan Institute

Abstract

Neural language models are becoming the prevailing methodology for the tasks of query answering, text classification, disambiguation, completion and translation. Commonly comprised of hundreds of millions of parameters, these neural network models offer state-of-the-art performance at the cost of interpretability; humans are no longer capable of tracing and understanding how decisions are being made. The attention mechanism, introduced initially for the task of translation, has been successfully adopted for other language-related tasks. We propose AttViz, an online toolkit for exploration of self-attention—real values associated with individual text tokens. We show how existing deep learning pipelines can produce outputs suitable for AttViz, offering novel visualizations of the attention heads and their aggregations with minimal effort, online. We show on examples of news segments how the proposed system can be used to inspect and potentially better understand what a model has learned (or emphasized).

1 Introduction

Contemporary machine learning that addresses text-related tasks adheres to the use of large *language models*—deep neural network architectures that have gone through extensive unsupervised pre-training in order to capture context-dependent meaning of individual tokens (Devlin et al., 2019; Liu et al., 2019; Yang et al., 2019). Even though pre-training of such multi-million parameter neural networks can be expensive (Radford et al., 2019), many pre-trained models have been made freely available to the wider research community, unveiling the opportunity for the exploration of how, and why such large models perform well. One of the main problems with neural language models is their *interpretability*. Even though the models learn the task well (even at super-human levels), understanding the reasons for predictions and inspection of whether the models picked up irrelevant biases or spurious correlations can be a non-trivial task.

Approaches to understanding black-box (non-interpretable) neural network models often resort to *post-hoc* approximations, e.g., SHAP (Lundberg and Lee, 2017), and similar are not necessary internal to the model itself. A potential way of extracting the token relevance is the attention mechanism (Bahdanau et al., 2014; Luong et al., 2015). The attention mechanism learns token pair-value mappings, potentially encoding *relations* between token pairs. When inspected as self-relations, the attention of a token w.r.t. itself (the diagonal element of the token attention matrix) potentially offers some insight into the importance of that token. Similar findings were also recently discussed when considering tabular data (Arik and Pfister, 2019). However, analytically, as well as numerically, exploration of attention can be a cumbersome task, resulting in the rise of approaches aimed at *attention visualization*. Visualization of (latent) embedding spaces is becoming ubiquitous in contemporary machine learning. For example, the Google's Embedding Projector[1] has offered numerous visualizations for non-savvy users, making embedding projections to low dimensional (human-understandable) vector spaces simple and available *online*. Even though visualization of simple embedding spaces is already accessible, visualization of

[1] https://projector.tensorflow.org/

complex neural network models' interior representations distributed across multiple embeddings (e.g., attention vectors), however, can be a challenging task. The works of (Liu et al., 2018a) and (Yanagimto et al., 2018) are examples of attempts at unveiling the workings of black-box attention layers and offering an interface for human researches to learn and inspect their models.(Liu et al., 2018a) visualize , as well as offer possible coloring of the attention space. Further, (Yanagimto et al., 2018) visualized self-attention with examples in sentiment analysis. The main contributions of AttViz are multi-fold, and can be stated as follows. AttViz focuses exclusively on self-attention and introduces two novel ways of visualizing this property while being available online and accessible to a wider audience. AttViz can interactively aggregate the attention vectors and offers simultaneous exploration of the output probability space, as well as the attention space.

The remainder of this work is structured as follows. In Section 2, we discuss the works, related to the proposed AttViz approach. In Section 3, we present the key ideas and technical implementation of AttViz, followed by our use case – a study of news segments in Section 5. Finally, we discuss (in Section 6) the overall capabilities of AttViz.

2 Attention visualization

Visualization of the attention mechanism for text has recently emerged as an active research area due to an increased popularity of attention based methods in natural language processing. Recent deep neural network language models such as BERT (Devlin et al., 2019), XLNet (Yang et al., 2019), and RoBERTa (Liu et al., 2019) are comprised of multiple *attention heads*—separate weight spaces each associated with the input sequence in *a unique way*. Language models consist of multiple attention matrices, all contributing to the final prediction. Visualising the attention weights from each of attention matrix is thus an important component in understanding and interpreting these models.

The attention mechanism which originated in the work on neural machine translation lends itself naturally to visualisation. (Bahdanau et al., 2014) used *heat maps* to display the attention weights between input and output text. This visualisation technique was first applied to the task of translation but found use in many other tasks such as visualising an input sentence and output summarization (Rush et al., 2015) and visualizing an input document and textual entailment hypothesis (Rocktäschel et al., 2015). In these heat map visualisations, a matrix or a vector is used to represent the learned alignments and color intensity illustrates attention weights. This provides a summary of the attention patterns describing how they map the input to the output. For classification tasks, a similar visualisation approach can be used to display the attention weights between the classified document and the predicted label (Yang et al., 2016; Tsaptsinos, 2017). Here, the visualisation of attention often displays the input document with the attention weights *superimposed* onto individual words. The superimposed attention weights are represented similarly to heat map visualisations using a color saturation to encode attention value.

An alternative visual encoding of attention weights is a bipartite graph visualisation. Here attention weights are represented by edge weights or thickness between two lists of words. This technique has been used to help interpret model output in neural machine translation (Lee et al., 2017), in natural language inference (Liu et al., 2018b), and for model debugging (Strobelt et al., 2018). A version of this visualisation which was designed specifically for multi-head self-attention (Vaswani et al., 2017) uses color hue to encode the attention head associated with each weight. The color is applied to the edges and is superimposed as a color strip over the node words. The intensity of the colors in the strips at each word position summarises the distribution of attention weight for that word across the heads. This multi-head self visualisation technique was recently extended (Vig, 2019) with two visualisations. The first, called "Model View", is a visualisation of the bipartite graphs for each layer and head in the system. The second visualization, "Neuron View", drills down to the computation of the attention score associated with each weighed edge in the bipartite graph. The element wise product, dot product, and softmax values are all visualised using coloured elements with saturation representing the magnitude of the value. This visualisation provides some insight into how each attention weight was computed while still providing the overview of attention weights.

The purpose of the proposed AttViz is to unveil the attention layer space to human explorers in an

intuitive manner. The tool emphasizes *self-attention*, that is, the diagonal of the token-token attention matrix which possibly corresponds to *relevance* of individual tokens. By making use of alternative encoding techniques, the attention weights across the layers and heads can be explored dynamically to investigate the interactions between the model and the input data. The proposed AttViz differentiates from existing visualization tools as follows. The focus of the tool is, as stated, self-attention, implying visualization of (attention-annotated) input token sequences can be carried out directly. We developed a novel visualization technique, where self-attention values are on per-token basis visualized across the input sequence for each self-attention vector. Further, AttViz offers visualization of the distribution of the attention values across the token sequence along with relevant aggregations, such as the min/max and similar. Finally, the tool simultaneously shows both the prediction probabilities, making interpretation of the self-attention space even more transparent, and with it the information on potential alternative classifications.

3 AttViz: An online toolkit for visualization of self-attention

We built AttViz, an online solution that can be coupled with existing language models from the PyTorch-transformers library[2]—one of the most widely used resources for language modeling. The idea behind AttViz is that it is *lightweight*, as it does not offer (online) neural model training, but facilitates the exploration of *trained* models. Along with AttViz, we provide a set of Python scripts that take as an input a trained neural language model and output a JSON file to be used by AttViz visualisation tool. A common pipeline for using AttViz is as follows. First, a transformer-based trained neural network model is used to obtain predictions on a desired set of instances (texts). The predictions are converted into the JSON format, suitable for AttViz, along with the attention space of the language model. The JSON file is loaded into AttViz (on the user's machine client side), where its visualization and exploration is possible. We next discuss the proposed visualization of the self-attention.

3.1 Visualization of self-attention

In this section, we discuss the proposed visualization schemes that emphasize different aspects of self-attention. The initial AttViz view offers sequence-level visualization, where each (byte-pair encoded) token is equipped with a self-attention value based on a given attention head (see Figure 1; central text space). Following the first row that represents the input text, consequent rows correspond to attention values that represent the importance of a given token with respect to a given attention head. As discussed in the empirical part of this paper (Section 5), the rationale for this display is that commonly, only a certain number of attention heads are activated (colored fields), thus visualization must entail both the whole attention space, as well as emphasize individual heads (and tokens).

The same document can also be viewed in the "aggregation" mode (Figure 2), where the attention sequence is shown across the token space. The user can interactively explore how the self-attention varies for individual input tokens, by changing both the scale, as well as the type of the aggregation used, the visualization can be used to emphasize various aspects of the self-attention space.

The second developed visualization (Figure 2) is the overall distribution of attention values across the whole token space. Resembling a time series, for each consequent token, the attention values are plotted separately. This visualization offers an insight into *self-attention peaks*, i.e. parts of the attention space focused around certain tokens that potentially impact the performance and the decision making process of a given neural network. This view also emphasizes different aggregations of the attention vector space for a single token (e.g., mean, entropy, and maximum). The visualization, apart from the mean self-attention (per token), also offers the information on maximum and minimum attention values (red dots), as well as the remainder of the self-attention values (gray dots). The user can this way explore both the self-attention peaks, as well as the overall spread.

[2] https://github.com/huggingface/transformers

3.2 Aggregation of self-attention

We apply several aggregation schemes across the space of individual tokens. Consider a matrix $A \in \mathbb{R}^{h \times t}$, where h is the number of attention vectors and t the number of tokens. We consider various aggregations across the second dimension of the attention matrix A (index j). In entropy based calculation, we denote with P_{ij} the probability of observing A_{ij} in the j-th column. The m_j corresponds to the number of unique values in that column. The proposed schemes are summarized in Table 1. The

Table 1: Aggregation schemes used in AttViz.

Aggregate name	Definition
Mean(j) (mean)	$\frac{1}{h}\sum_i A_{ij}$
Entropy(j) (ent)	$-\frac{1}{m_j}\sum_{i=0}^{h} P_{ij} \log P_{ij}$
Standard deviation(j) (std)	$\sqrt{\frac{1}{h-1}\sum_i (A_{ij} - \overline{A_{ij}})^2}$
Elementwise Max(j) (max)	$\max_i(A_{ij})$
Elementwise Min(j) (min)	$\min_i(A_{ij})$

attention aggregates can also be visualized as part of the aggregate view (Figure 2), where, for example, the mean attention is plotted as a line along with the attention space for each token, depicting the *dispersion* around certain parts of the input text.

4 Comparison with state-of-the-art

In the following section we discuss in more detail the similarities and differences of AttViz with other state-of-the-art visualization approaches. Comparisons are summarized in Table 2. The neat-vison package is available at[3].

Approach	AttViz (this work)	BertViz (Vig, 2019)	neat-vision	NCRF++ (Yang and Zhang, 2018)
Visualization types	sequence, aggregates	head, model, neuron	sequence	sequence
Open source	✓	✓	✓	✓
Language	Python + Node.js	Python	Python + Node.js	Python
Accessibility	Online	Jupyter notebooks	Online	script-based
Sequence view	✓	✓	✓	✓
Interactive	✓	✓	✓	✗
Aggregated view	✓	✗	✗	✗
Target probabilities	✓	✗	✓	✗
Compatible with PyTorch Transformers? (Wolf et al., 2019)	✓	✓	✗	✗
token-to-token attention	✗	✓	✗	✓

Table 2: Comparison of different aspects of the attention visualization approaches.

The main novelties introduced as part of AttViz are the capability to aggregate the attention vectors with four different aggregation schemes, offering insights both into the average attention but also its dispersity around a given token. The neat-vision project is the closest to AttViz's functionality, with the following differences. It is not directly bound to PyTorch transformers library, requiring additional pre-processing on the user-side. Similarly, the fast switching between the sequence and aggregate view are more emphasized in AttViz, as they offer more general overview of the attention space. The class probabilities are to our knowledge available in both tools, offering simultaneous exploration of both input and output space at the same time.

5 Example usage: News visualization

In this section, we present a step-by-step use of the server along with potential insights the user can obtain. The examples are based on the BBC news data set[4] (Greene and Cunningham, 2006) that contains 2,225 news articles on five different topics (business, entertainment, politics, sport, tech). The documents

[3] https://github.com/cbaziotis/neat-vision

[4] https://github.com/suraj-deshmukh/BBC-Dataset-News-Classification/blob/master/dataset/dataset.csv

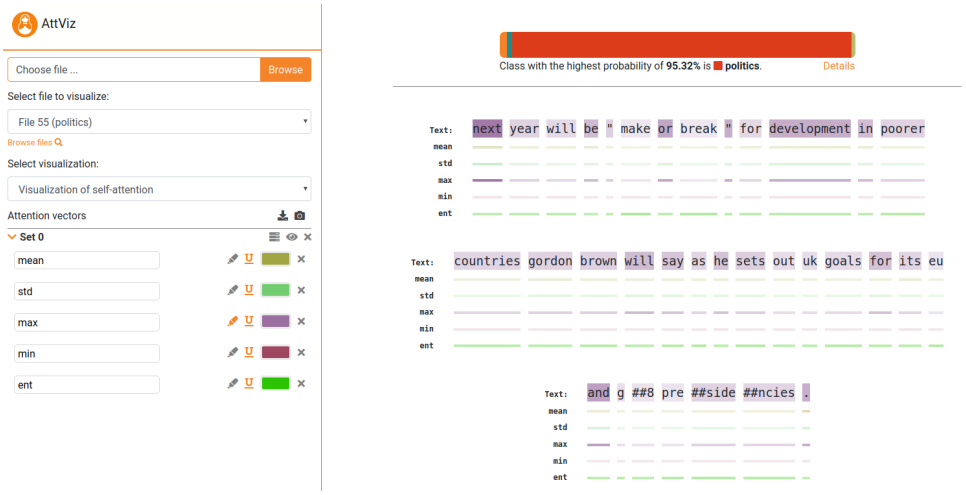

Figure 1: Visualization of aggregations. The document was classified as a politics-related topic, it can be observed that aggregations emphasize tokens such as “development”,“uk” and “poorer”. The user can also highlight desired head information – in this example the maximum attention (purple) is highlighted.

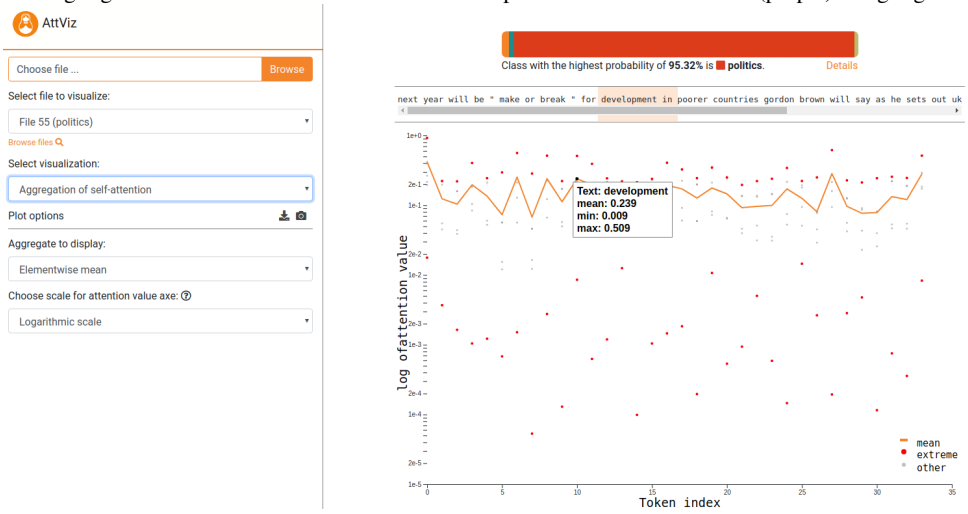

Figure 2: The interactive series view. The user can, by hoovering over the desired part of the sequence, inspect the attention values and their aggregations. The text above the visualization is also highlighted automatically.

from the dataset were split into short *segments*. The splits allow easier training (manageable sequence lengths), as well as easier inspection of the models. We split the dataset into 60% of the documents that were used to train a BERT-base (Devlin et al., 2019) model, 20% for validation and 20% for testing[5].

The fine-tuning of the BERT model was conducted as discussed in the examples of the PyTorch-Transformers library (Wolf et al., 2019). The best-performing hyper parameter combination was using

[5]The obtained model classified the *whole* documents into five categories with 96% Accuracy, which is comparable with the state-of-the-art performance (Trieu et al., 2017); however, note that the train, validation, and test splits were randomly created. For prediction and visualisation, only short segments are used

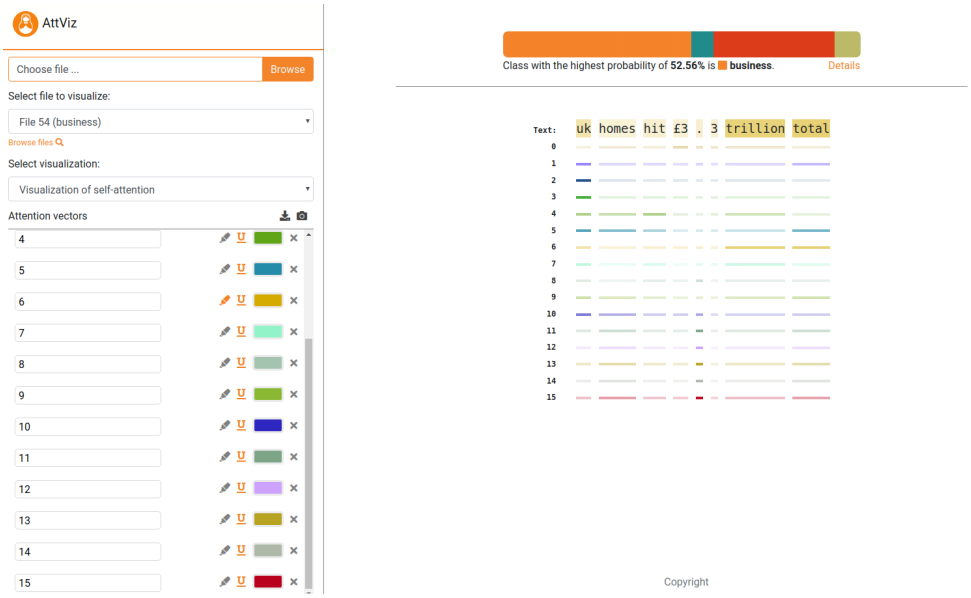

Figure 3: Visualization of all attention heads. The sixth heads's self attention is also used to highlight the text. The document was classified as a business-related, which can be linked to high self attention at the "trillion" and "uk" tokens. Note also that, compared to the first two examples (Figures 1 and 2), the network is less *certain* – the business and politics classes were predicted with similar probabilities (orange and red parts of the bar above visualized text).

3 epochs with the sequence length of 512 (other hyper parameters were left at their default values). We used Nvidia Tesla V100 GPU processor for fine-tuning. While more recent larger language models such as e.g., XLNet (Yang et al., 2019) could produce better accuracy, the idea and the use of AttViz visualizations is the same; hence, we selected the most commonly used model (BERT-base).

The main user interface of AttViz is displayed in Figures 1 and 2 and 3. In the first example (Figure 1), the user can observe the main view that consists of two parts. The leftmost part shows (by id) individual self-attention vectors, along with visualization, aggregation and file selection options. The file selection indexes all examples contained in the input (JSON) file. Attention vectors can be colored with custom colors, as shown in the central (token-value view). The user can observe that, for example, the violet attention head (no. 5) is active, and emphasizes tokens such as "servants" (from civil servants), which indicates a politics-related topic (as correctly classified). Here, the token (byte-pair encoded) space is shown along with self-attention values for each token. The attention vectors are shown below the token space and aligned for direct inspection (and correspondence). Further, the upmost visualization in the right part of the view shows probabilities (obtained via softmax normalization of the output layer weights) of the considered document belonging to a given class. This functionality was added to help human explorers investigate the correspondence between the actual classification and the classified text. Using "Details" section below the probability legend, the distributions across the class space can be further inspected.

In Figure 2, the user can observe the same text segment as an attention series spanning the input token space. Again, note that tokens, such as "trillion" and "uk" correspond to high values in a subset of the attention heads, indicating their potential importance for the obtained classification. However, we observed that only a few attention heads "activate" with respect to individual tokens, indicating that other attention heads are not focusing on the tokens themselves, but possibly on *relations* between them. This is possible and the attention matrices contain such information, yet the study of token relations is not

the focus of this work (see (Vig, 2019) for such a visualization). In this work we focus on self-attention as such information can be directly mapped across token sequences, emphasizing tokens that are of relevance to the classification task at hand. Consequently, we see AttViz as being the most useful when exploring models used for classification of hatespeech or similar news texts, where individual tokens carry key information for classification.

In the example in Figure 3, we visualize a short segment related to uk homes and spending. Note that the text is shown after the preprocessing consisting of byte-pair encoding and lower-casing.

The segment was correctly classified as business-related. Tokens, such as "trillion", "uk" and "total" are all associated with high attention. The example shows how different attention heads detect different aspects of the sentence, even at the single token (self-attention) level. The user can observe that the next most probable category for this topic was politics (red color), which is indeed a more sensible classification than e.g., sports. The example shows how interpretation of the attention can be coupled with the model's output for increased interpretability.

A careful inspection of the remainder of the documents revealed that in the majority of cases, the first token is also emphasized. We believe the following reasons can induce this observed bias. First, as the BERT-base model was used for the classification task, the model was only fine-tuned on the news data set (for a few epochs), after being extensively pre-trained on vast amounts of text. The pre-training phase could introduce the bias, as the model is implicitly forced to learn to predict the next token, indicating that the first token in the classified segment will be of high "relevance". In the second interpretation, when the first token is a content work, it can already carry a lot of meaning for the whole sentence, thus it could be reasonably relevant for the task.

6 Critical overview of AttViz and Conclusions

As AttViz is an online toolkit for facilitated attention exploration, we discuss possible concerns regarding its usefulness. One of the main issues with online methods is privacy. Currently, AttViz does not employ any anonymization strategies, meaning that private processing of the input data is not guaranteed. We believe that this issue can be addressed as a part of further work or with a private installation of the tool. Further, AttViz leverages users' computing capabilities, meaning that too large data sets can cause memory overheads (e.g., several millions of examples). We believe that such situations are difficult to address with AttViz, however, instances can be filtered prior to being used in AttViz. This would enable seamless exploration of a subset of the data (e.g., only (in)correctly predicted instances, or certain time slot of instances). In terms of functionality, AttViz is focused on the exploration of *self-attention*. We realize that the self-attention is not necessarily the only important aspect of a neural network that needs to be inspected, but it is possibly the one, where visualisation techniques have been the least explored. Similarly to the work of (Liu et al., 2018a), we plan to further explore potentially interesting *relations* emerging from the attention matrices.

Finally, we believe AttViz could be further extended with a larger database of popular models and a back-end functionality, enabling it to, e.g., fine-tune models. Such endeavors are out of the scope of this paper—the current version of AttViz is lightweight, can be hosted by anyone (with minimal requirements overhead) and performs fast when considering exploration of self-attention.

7 Availability

The tutorials and other input preparation scripts are available at: `https://github.com/SkBlaz/attviz`. The server is live at: http://attviz.ijs.si.

Acknowledgements

Omitted for anonymity reasons.

References

Sercan O Arik and Tomas Pfister. 2019. Tabnet: Attentive interpretable tabular learning. *arXiv preprint arXiv:1908.07442*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Derek Greene and Pádraig Cunningham. 2006. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proceedings of the 23rd International Conference on Machine learning (ICML'06)*, pages 377–384. ACM Press.

Jaesong Lee, Joong-Hwi Shin, and Jun-Seok Kim. 2017. Interactive visualization and manipulation of attention-based neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 121–126, Copenhagen, Denmark, September. Association for Computational Linguistics.

Shusen Liu, Tao Li, Zhimin Li, Vivek Srikumar, Valerio Pascucci, and Peer-Timo Bremer. 2018a. Visual interrogation of attention-based models for natural language inference and machine comprehension. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

Shusen Liu, Tao Li, Zhimin Li, Vivek Srikumar, Valerio Pascucci, and Peer-Timo Bremer. 2018b. Visual interrogation of attention-based models for natural language inference and machine comprehension. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 36–41, Brussels, Belgium, November. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September. Association for Computational Linguistics.

Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M. Rush. 2018. Seq2seq-vis: A visual debugging tool for sequence-to-sequence models. *CoRR*, abs/1804.09299.

Lap Q. Trieu, Huy Q. Tran, and Minh-Triet Tran. 2017. News classification from social media using twitter-based doc2vec model and automatic query expansion. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*, SoICT 2017, pages 460–467, New York, NY, USA. ACM.

Alexandros Tsaptsinos. 2017. Lyrics-based music genre classification using a hierarchical attention network. *CoRR*, abs/1707.04678.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Jesse Vig. 2019. Visualizing attention in transformer-based language representation models. *CoRR*, abs/1904.02679.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

H. Yanagimto, K. Hashimoto, and M. Okada. 2018. Attention visualization of gated convolutional neural networks with self attention in sentiment analysis. In *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*, pages 77–82, Dec.

Jie Yang and Yue Zhang. 2018. Ncrf++: An open-source neural sequence labeling toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.