Learning by Interpreting

Xuting Tang[1] , **Abdul Rafae Khan** [1] , **Shusen Wang** [2], and **Jia Xu**[1*]
[1]Steven Institute of Technology
[2]Xiaohongshu Inc
{xtang18, akhan4, jxu70}@stevens.edu, shusenwang@xiaohongshu.com

Abstract

This paper introduces a novel way of enhancing NLP prediction accuracy by incorporating model interpretation insights. Conventional efforts often focus on balancing the trade-offs between accuracy and interpretability, for instance, sacrificing model performance to increase the explainability. Here, we take a unique approach and show that model interpretation can ultimately help improve NLP quality. Specifically, we employ our learned interpretability results using attention mechanisms, LIME, and SHAP to train our model. We demonstrate a significant increase in accuracy of up to +3.4 BLEU points on NMT and up to +4.8 points on GLUE tasks, verifying our hypothesis that it is possible to achieve better model learning by incorporating model interpretation knowledge.

1 Introduction

Deep Learning (DL) methods dominate the NLP state-of-the-art. However, their black-box nature hinders our understanding of how the model reaches decisions. Because less information about the importance of model features and components makes it harder to improve model architectures, DL models are often designed to be more complex (by adding parameters) and thus require more training data for quality climbing. Blindly enlarging model sizes on one side causes higher demands for computational and linguistic resources as well as a reduction in model accountability, controllability, and learning generalization.

To address these problems, there has been a plethora of works providing explainability to DL [Du *et al.*, 2019], such as attention mechanisms, LIME, and SHAP. Nonetheless, there are often trade-offs between model performance and explanation fidelity [Du *et al.*, 2019]. Models with intrinsic explainability can provide explanations with high fidelity but often sacrifice prediction accuracy; while analyzing models with post-hoc methods does not hurt the performance of the models, the obtained explanations can be limited. Therefore, one important challenge is to handle the trade-offs between model accuracy and explainability, and we propose our research question from a novel view: *Can interpretability results improve machine learning prediction accuracy?* Our experiments suggest a positive answer, aligning with our goal of reducing the model deficiency and errors with our method of adding the explainable knowledge into decision-making. In contrast to the conventional approach trading off between model performance and explainability, we show that accuracy and interpretability goals can be unified and help each other.

Specifically, we introduce the paradigm of "interpretability enhanced learning in NLP." Our approach learns interpretations of the model learning process and adds this interpretability result in the form of additional input features into the model learning. It seems that when DL knows the important input features to focus on for each learning step, the learning task becomes much easier than when DL knows nothing about the importance of each particular input. Our intuition is explained further with the following three toy examples.

Our first example is an analogy to human learning, when a student first reads a book and highlights the important phrases and statements. The second time the student rereads this book, the highlighted parts reinforce the studying. The second example is the modeling of the XOR function. $\text{XOR}(x_1, x_2)$ is a nonlinear function that cannot be modeled correctly with a linear separator. However, if we add another input feature $x_1 \times x_2$ to x_1 and x_2, then $\text{XOR}(x_1, x_2, x_1 \times x_2)$ becomes linear separable and a much simpler function to learn. In the third example, let us take a new bit representing the computation of a Neural Network (NN). Our new input is formed by adding this new bit to our previous input. Information theoretically, this new bit does not interject any new information, since one can infer it from the NN and the input. If we choose this function in such a way that you cannot compute it (such as a halting problem), then in this specific occasion described, one can never **compute** this bit. This means that while the answer exists, there is no way to compute it. These three examples from different perspectives showcase why adding extracted features into DL can possibly help learning and reduce prediction errors.

Now that we have put our previous question in context, let us describe how we realize our "interpretability enhanced learning" in the NLP framework. First, we train a baseline NLP model and extract the interpretability results for each

*Jia Xu is the corresponding author of this paper.

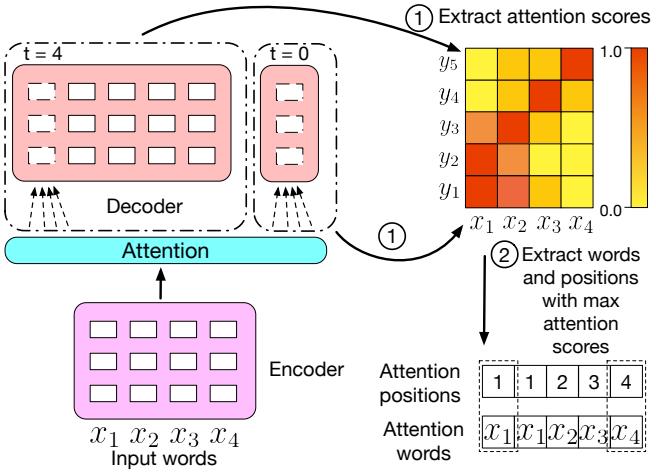

Figure 1: Extract attention words and positions. Figure shows decoder at time step t=0 and t=4.

training sample, for example, we extract the sequence of input features contributing most to generate each output token in a sentence. Second, we add these interpretability results as auxiliary information into the original input features and re-train the NLP model. Last, we decode the test set to get the interpretability information and then add it to re-decode for the final prediction.

We apply three interpretability methods to verify our paradigm: attention mechanisms, LIME, and SHAP. The details of implementation vary across NLP tasks. Figure 1 illustrates an example of extracting attention model explainable features for a Neural Machine Translation (NMT) task. After training a baseline model, we go through every target output token (word/sub-word) for each sample and extract the attention weights of each input token $(x_1, \cdots, x_i, \cdots, x_I)$ to generate its output token. Each column indicates the target token sequence $(y_1, \cdots, y_j, \cdots, y_J)$ to be generated. Afterward, we generate an attention word vector by selecting the input token with the highest attention weight for each output token. In the end, the attention word vector is concatenated with the original source sentence and fed into the NLP models for our model re-training. In the test phase, we run the test on the baseline model to obtain the attention word vector, add it to the source sentences, and decode again.

Our experimental results demonstrate a significant improvement on several NLP tasks, including +1.4 BLEU points on WMT DE-EN and up to +3.4 BLEU points on IWSLT'17 FR-EN over baselines of Transformer and ConvS2S in NMT as well as up to +4.8 points in GLUE tasks.

The major contributions of this work mainly includes:

1. introducing an enhanced learning NLP paradigm by employing the model interpretability results;
2. extracting interpretability information of attention mechanisms, LIME, and SHAP and incorporating them into DL model training;
3. verifying the usefulness of the interpretation methods with NLP improvements;
4. significantly enhancing NLP quality on NLP tasks.

Below, we will first describe our algorithms and model architectures then show experimental results and analysis, and after that, we overview related work and conclude.

2 Interpretability Models

We consider interpretability as the input words and word sequences that are important to predict. We apply three well-known interpretability methods to find these important words, including a model-based popular intrinsic method using attention mechanisms [Serrano and Smith, 2019; Gomez *et al.*, 2021], two model-independent post-hoc methods, LIME [Ribeiro *et al.*, 2016], and SHAP [Lundberg and Lee, 2017].

2.1 Attention Mechanisms

In this paper, we evaluate our approach on two baseline models: ConvS2S [Gehring *et al.*, 2017] and Transformer [Vaswani *et al.*, 2017], where we use different attention mechanisms and positional embeddings.

ConvS2S. At each decoding step j, ConvS2S combines decoder hidden state s_j and the embedding of previous target word g_j to obtain decoder state summary $o_j = Ws_j + b + g_j$, where W is a weight matrix and b is bias. The attention score of state j and source element i can be computed as

$$a_{ij} = \frac{\exp(o_j \cdot h_i)}{\sum_{k=1}^{I} \exp(o_j \cdot h_k)}. \quad (1)$$

Transformer. In Transformer, the attention is calculated by a scaled dot-product of key k_i and query q_j:

$$e_{ij} = \frac{\left(W^K k_i\right)^T \left(W^Q q_j\right)}{\sqrt{d_k}}, \quad (2)$$

where d_k is the dimension of k_i, and K and Q represents the key matrix and query matrix, respectively. $W^K, W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are parameter matrices, and d_{model} is the output dimension of attention and embedding layer. The attention weight is computed by applying a softmax function over e_{ij}:

$$a_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{I} \exp e_{kj}} \quad (3)$$

The transformer encoder and decoder have multiple layers. Within a layer, there are multiple heads. Each head has its independently learned queries and keys, which means each head has different attention. We used the sum of attention of all heads from the last layer for attention information extraction.

2.2 Attention with Positional Embeddings

ConvS2S. A separate embedding layer is learned for positional embeddings. Let us take the absolute positions of input tokens as $\mathbf{p} = (p_1, \cdots, p_I)$, the input representation is computed by taking the sum of word embeddings and positional embeddings: $\mathbf{r} = (\epsilon_1(x_1) + \epsilon_2(p_1), \ldots, \epsilon_1(x_I) + \epsilon_2(p_I))$, where $\epsilon_1(\cdot)$ and $\epsilon_2(\cdot)$ are learned word embedding function and positional embedding function, respectively.

Transformer. Instead of learning positional embeddings, Transformer uses Sine and Cosine functions of different frequencies for positional embedding calculation: $\epsilon_{(p_i,2l)} = \sin(p_i/10000^{2l/d_{\text{model}}})$ and $\epsilon_{(p_i,2l+1)} = \cos(p_i/10000^{2l/d_{\text{model}}})$, where l represents the indices of each dimension of the positional embedding, and d_{model} is the output dimension of attention layer and embedding layer.

2.3 LIME

In addition to intrinsic interpretability methods like attention mechanisms, we also apply post-hoc interpretability methods. Local Interpretable Model-agnostic Explanations (LIME) [Ribeiro *et al.*, 2016] analyzes input features by generating local explanations of black-box models to interpret their behaviors. It approximates the predictions of a black-box model via local surrogate interpretable models, such as linear models and decision trees. To explain an instance, LIME samples nearby observations by drawing nonzero elements of the instance uniformly at random. The sampled nearby observations and model predictions of these observations can fit an interpretable model to create an explanation for this instance. In text classification task, for example, LIME generates an importance score for each word for every class, and we choose the word with the highest importance score for the prediction.

2.4 SHAP

SHapley Additive exPlanations (SHAP) [Lundberg and Lee, 2017] is another black-box interpretability method that utilizes Shapley values derived from game theory. It assigns Shapley values to each feature for a particular prediction. The higher the Shapley value a feature gets, the more important the feature is to the model's prediction. We also output one word most important to the prediction.

3 Extract Explainable Features

Now, we present how to extract the explainable features, namely important word sequences/words from attentional model, and LIME/SHAP, respectively, for decision making.

3.1 Attention Words

We extract two types of information from attention mechanism: (1) the source word sequence with the highest attention score to generate each target word one by one, see Equation 4. We refer to them as "attention words" $\boldsymbol{\alpha}$; (2) the positions of the attention words, which we call "attention positions" $\boldsymbol{\rho}$.

Figure 1 illustrates an example of extracting token sequences with high attention weights. To simplify the notation, we formalize the method of obtaining a sequence of attention words $\boldsymbol{\alpha}$ and a sequence of attention positions $\boldsymbol{\rho}$ as following:

$$\begin{aligned} \rho_j =& \arg \max_{i \in 1,\cdots,I} (a_{ij}) \\ \alpha_j =& x(\rho_j), \end{aligned} \tag{4}$$

where $\boldsymbol{\rho} = (\rho_1, \ldots, j, \cdots, \rho_J)$, $\boldsymbol{\alpha} = (\alpha_1, \cdots, j, \cdots, \alpha_J)$, and $x(\rho_j)$ is the input token (word or subword) at the ρ_j-th position in the input sentence $\mathbf{x}$. This means that each target word is predicted using the sequence of input words; each input word has an attention score, and we choose the word with the highest attention score.

Method		Example
Attention	Source	C'est un stylo .
	Reference	This is a pen .
	Attention Words	C'est C'est un stylo .
	Attention Positions	1, 1, 2, 3, 4
LIME/SHAP	Input Sequence	A great musician
	Important Word	great

Table 1: Example of extracted attention words/positions using attention mechanism and important word using LIME/SHAP.

Algorithm 1 Attention Words/Positions Extraction

Input: Pre-trained model ($\mathcal{M}$), source tokens ($\mathbf{x} = x_1, \cdots, x_I$), special tokens for start (*START*) and end (*END*)
Output: Predicted tokens ($\hat{\mathbf{y}}$), attention words ($\boldsymbol{\alpha}$), attention positions ($\boldsymbol{\rho}$)

```
train = FALSE
Initialize start prediction token ŷ_0 ← START
Initialize lists ŷ, α and ρ
h ← get M's encoder hidden states
j ← 0
while ŷ_j ≠ END do
    s ← get M's decoder hidden states
    Get M's attention scores {a_ij}_{i=1}^I between h and s
    ρ_j ← arg max_i({a_ij}_{i=1}^I)
    Append ρ_j to ρ
    α_j ← token at position ρ_j in x
    Append α_j to α
    ŷ_{j+1} ← get M's next predicted token
    Append ŷ_{j+1} to ŷ
    j ← j + 1
end while
return ŷ, α, ρ
```

Table 1 shows an example of such extraction. For a pair of tokenized parallel French to English sentences, the source is "C'est un stylo .", and the target is "This is a pen ." The extracted attention positions form the sequence of IDs [1, 1, 2, 3, 4]. If we look for the source tokens corresponding to these IDs, we get the sequence of attention words "C'est C'est un stylo ."

Algorithm 1 describes the process of extracting attention words and positions from a trained model ($\mathcal{M}$). For next output word $\hat{y}_{j+1}$, the attention network takes as input the encoder hidden states h and the decoder states s_j of previous predicted word $\hat{y}_j$ and outputs the attention vector. The source word with the maximum attention score and its corresponding position are extracted. The algorithm outputs the prediction, attention word and attention position for each target position.

3.2 LIME and SHAP

We apply LIME and SHAP on text classification tasks. Given a pre-trained model and an input sequence, LIME and SHAP

Algorithm 2 Important Words Extraction with LIME/SHAP

Input: Pre-trained model ($\mathcal{M}$), source tokens ($\mathbf{x} = x_1, x_2, \cdots, x_I$), LIME/SHAP explainer ($\mathcal{E}$)
Output: predicted label (c), important word (ω)

```
train = FALSE
c ← get M's prediction of x
E takes as input M, x and c, outputs importance scores {b_i}_{i=1}^I of all tokens on prediction c
i ← arg max_i({b_i}_{i=1}^I)
ω ← x_i
return c, ω
```

generate each word's importance score to the predicted label. Similar to extracting words with the highest attention scores, the word with the highest importance score, referred to as "important word" is extracted for each input sequence, details are in Algorithm 2. We adopt the implementations of LIME[1] and SHAP.[2]

4 Incorporate Explainable Features

4.1 Combine Word/Word Sequence Interpretation

The explainable features extracted from methods in Section 3 are in the form of word sequences for attention models or words for LIME and SHAP. We incorporate these features into our NLP model training and decoding in the following way. We concatenate the original sentences in training set and test set with their explainable features. We use the new training set to fine-tune our model and the new test set to decode on our fine-tuned model. For the example in Table 1, the concatenated input to NLP model is "C'est un stylo . C'est C'est un stylo ." Figure 2 illustrates examples showing the detail of our concatenation method for attention model, attention model with positional embeddings, LIME, and SHAP. The explainable features are appended to the end of the original sentence. To add positional embeddings, the previous positions "[5, 6, 7, 8, 9]" are replaced by attention positions "[1, 1, 2, 3, 4]".

Let us give a translation example to our concept. We translate from "good morning Girls" to German "Guten Morgen Maedchen", then $\mathbf{x}+\boldsymbol{\alpha}$ = [good, morning, girls, good (weight 0.9 of good for Gute), morning (weight 0.8 of morning for Morgen), girls (weight 0.9 of girls for Maedchen)]. Note that our method should not be confused with data augmentation. The former adds auxiliary information without additional data, while the latter adds manual labeled or synthetic training data to learning.

4.2 Training

After constructing the new dataset by combining the explainable words with the original sentences, we fine-tune our baseline models as in Algorithm 3 and Algorithm 4.

Note that we use the deep learning NLP models in two modes. The train mode is for back-propagation and model parameter updating, and the test mode does not do back-propagation or update but decodes the inputs. The extraction process of attention information or important words is conducted in the test mode, where there is no parameter update and the target sentences or labels are not used. There are two reasons to extract in the test mode: (1) the pre-trained model should not be updated during extraction; (2) we do not have target sentences or labels for test data, and the training mode requires targets. Therefore, the extraction processes should all be done in test mode for training, validation, and test data. The decoding is performed in two steps. The first step is in the test mode to extract the attention information or important words, and the second step is in the training mode because we need to fine-tune the model with the concatenated inputs and update its parameters.

[1]https://github.com/marcotcr/lime.
[2]https://github.com/slundberg/shap.

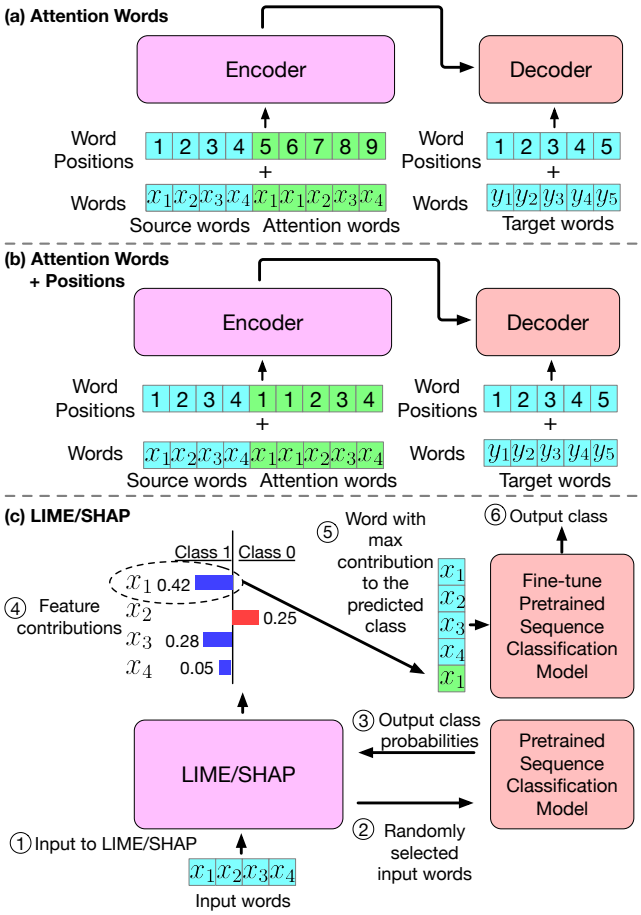

Figure 2: (a) Combine attention Words only. (b) Combine attention words & attention positions. (c) Combine important words from LIME/SHAP.

4.3 Bootstrapping

Since incorporating extracted attention information can aid the training of a pre-trained model, after fine-tuning the model with attention information, we can obtain a model with better performance. We can further extract attention information from the better model, and fine-tune the better model with the newly extracted attention information. This training

Algorithm 3 Train NMT with Attention Information

Input: Pre-trained model ($\mathcal{M}$), source tokens ($\mathbf{x} = x_1, \cdots, x_I$), target tokens ($\mathbf{y} = y_1, \ldots, y_J$), attention words ($\boldsymbol{\alpha} = \alpha_1, \cdots, \alpha_J$), attention positions ($\boldsymbol{\rho} = \rho_1, \cdots, \rho_J$)
Output: Fine-tuned model ($\mathcal{M}$)

$\mathbf{x}' \leftarrow$ concatenate $\mathbf{x}$ and $\boldsymbol{\alpha}$ as $[x_1, \cdots, x_I, \alpha_1, \cdots, \alpha_J]$
$\eta \leftarrow$ "whether to combine attention positions"
if η **then**
  $\mathbf{p}' \leftarrow [1, 2, \cdots, I, \rho_1, \rho_2, \cdots, \rho_J]$
else
  $\mathbf{p}' \leftarrow [1, 2, \cdots, I, \cdots, I + J]$
end if
Fine-tune $\mathcal{M}$ with $\mathbf{x}', \mathbf{p}', \mathbf{y}$ as usual
return $\mathcal{M}$

Algorithm 4 Train GLUE Model with Attention/Importance Information

Input: Pre-trained model ($\mathcal{M}$), input tokens ($\mathbf{x} = x_1, \cdots, x_I$), label ($y$), attention word ($\alpha$), important word ($\omega$)
Output: Fine-tuned model ($\mathcal{M}$)

$\eta \leftarrow$ "whether to combine attention word"
if η **then**
  $\mathbf{x}' \leftarrow$ concatenate $\mathbf{x}$ and $\boldsymbol{\alpha}$ as $[x_1, \cdots, x_I, \alpha]$
else
  $\mathbf{x}' \leftarrow$ concatenate $\mathbf{x}$ and ω as $[x_1, \cdots, x_I, \omega]$
end if
Fine-tune $\mathcal{M}$ with $\mathbf{x}', y$ as usual
return $\mathcal{M}$

strategy is referred to as bootstrapping with attention.

5 NLP Applications

We evaluate our approach on a common NLP task, Neural Machine Translation (NMT), and GLUE, a standard benchmark with various NLP tasks.

5.1 Machine Translation

Datasets and Pre-processing. We conduct our experiments on the large-scale WMT 2014 dataset in the German-English news domain and the medium scale IWSLT 2017 dataset in the French-English TEDTalk domain. The raw datasets contain about 4.5 million and 25.2 thousand pairs of sentences, respectively. For pre-processing, we use Moses [Koehn *et al.*, 2007] tokenizer, filter out the sentences with source and target length ratio greater than 1.5, and apply Moses truecaser. Then, we apply Byte-Pair-Encoding (BPE) [Sennrich *et al.*, 2015] using 32K merge operations for the WMT data and 16K operations for the IWSLT data. Table 2 shows the corpus statistics before and after pre-processing and BPE.

Model training and parameter settings. We train our model using Fairseq toolkit [Ott *et al.*, 2019] for ConvS2S and the Transformer. The hyper-parameters for training the ConvS2S network include: the encoder and decoder embedding dimensions as 768, the learning rate as 0.25, the gradient clip norm as 0.1, the dropout ratio as 0.2, the max tokens in a batch as 4000, the optimizer as NAG. For Transformer, we set the embedding dimension as 768, the learning rate as 5×10^{-4}, the warmup updates as 4000, the dropout as 0.3, the weight decay as 1×10^{-4}, the max tokens in a batch as 5000, the Adam optimizer [Kingma and Ba, 2014] betas as 0.9 and 0.98. Both models use early stopping with a patience of 5 and the max number of epochs is 50. The training is performed on 4 Nvidia Titan V GPUs. For the TEDTalk task, we reduce the encoder and decoder dimension to 256 for ConvS2S and 512 for Transformer. Other hyper-parameters are the same as for the news task. The decoding beam size is 5 for NMT. We evaluate using Sacrebleu [Post, 2018] with the tokenizer parameter set to "13a".

		WMT'14 DE-EN		**IWSLT'17 FR-EN**	
Sents. Raw		4.5M		25.2K	
Pre-processed		3.9M		23.6K	
		DE	EN	FR	EN
Before BPE	R.W.	98M	102M	5.3M	4.9M
	Vocab	1.5M	684K	8.3K	6.2K
After BPE	R.W.	118M	112M	5.9M	5.4M
	Vocab	35K	33K	13K	11K

Table 2: Number of Sentences in WMT 2014 News (DE-EN) and IWSLT 2017 TEDTalk (FR-EN) datasets. Vocabulary statistics in WMT 2014 News (DE-EN) and IWSLT 2017 TEDTalk (FR-EN) datasets before and after Byte-pair encoding. R.W. is the number of running words and Vocab is the vocabulary size.

ConvS2S				**Transformer**	
DE-EN		EN-DE		DE-EN	
Baseline	Attn W	Baseline	Attn W	Baseline	Attn W
29.0	**30.4**	24.4	**25.0**	29.7	**31.1**

Table 3: Translation results in BLEU[%] on WMT'14 News. Baseline is the pre-trained model, model fine-tuned with attention words is denoted as "Attn W".

Results. As shown in Table 3, on WMT 2014 DE-EN, after fine-tuning the baseline ConvS2S model using attention words (Figure 2), the fine-tuned model (Attn W) achieves an improvement of +1.4 BLEU points over the baseline (Baseline) model. Similarly, on the same language direction, the Transformer baseline fine-tuned with attention words also achieve +1.4 BLEU points improvement. On IWSLT 2017 FR-EN, as shown in Table 4, fine-tuning baseline ConvS2S model with attention words achieves a +2.8 BLEU points improvement; fine-tuning the pre-trained Transformer model with attention words achieves an improvement of +3.4 BLEU points. We evaluate fine-tuning on attention words with and without attention positions on IWSLT'17 FR-EN task (Figure 2). The results are shown in Table 4. We observe a slight improvement when using attention position information (Attn W + Attn P). We also apply bootstrapping (Attn W Bootstrap) with a max bootstrapping iteration of 20. The training reaches its best validation BLEU score at iteration 10, and its corresponding test BLEU score is $+0.4$ higher than fine-tuning with one iteration. So bootstrapping explainable feature extraction and training brings slight improvement.

Model	Method	IWSLT'17 FR-EN
ConvS2S	Baseline	30.5
	Attn W	33.3
	Attn W + Attn P	33.5
	Attn W Bootstrap	**33.7**
Transformer	Baseline	32.9
	Attn W	**36.3**

Table 4: Translation results in BLEU[%] on IWSLT'17 FR-EN. "Attn W + Attn P" is fine-tuning with the combination of attention words and attention positions, "Attn W Bootstrap" is fine-tuning with attention words in a bootstrapping manner.

	QQP	CoLA	STS-B	RTE
Baseline	88.8	51.4	82.6	65.8
Attn W	88.9	52.1	83.1	**68.1**
SHAP W	88.8	**56.2**	**84.9**	68.0
LIME W	**89.1**	54.3	84.4	67.5

Table 5: Results on GLUE Benchmark. Baseline: BERT-base fine-tuned on the corresponding dataset. "Attn W": model fine-tuned with attention words. "LIME W"/"SHAP W" : model fine-tuned with important words extracted by LIME/SHAP. The evaluation uses Matthews correlation coefficient for CoLA, Spearman correlations for STS-B, and accuracy for RTE and QQP.

5.2 GLUE Benchmark

Datasets. In addition to the NMT task, we evaluate our approach on QQP, CoLA, STS-B and RTE of the widely used GLUE Benchmark [Wang *et al.*, 2018].

Model training and parameter settings. For each GLUE task, we first fine-tune the Huggingface [Wolf *et al.*, 2020] BERT-base [Devlin *et al.*, 2019] cased model . The hyperparameters for fine-tuning are as following: the encoder dimension is 768, the number of epochs is 3, the learning rate is 5×10^{-5}, the dropout rate is 0.1, and the Adam optimizer betas is 0.9 & 0.999. After the fine-tuning, we use three different methods (i.e. attention, LIME and SHAP) to extract attention/important words. For attention mechanism, we only extract the word with the highest attention score with respect to the special classification token "[CLS]". Each of the extracted words is combined with the corresponding original input (see Figure 2) and the system is fine-tuned once more using the best learning rate on the validation set. The training time with and without our methods are about the same. Our methods need additional time to extract features, where LIME/SHAP feature extraction is more time-consuming than attention feature extraction.

Results. Table 5 shows the GLUE task results. Incorporating attention words into fine-tuning achieves up to +2.3 BLEU points and incorporating important words of LIME/SHAP achieves up to +4.8 points over the baseline. Our experiments demonstrate that incorporating interpretability results can significantly enhances the model quality.

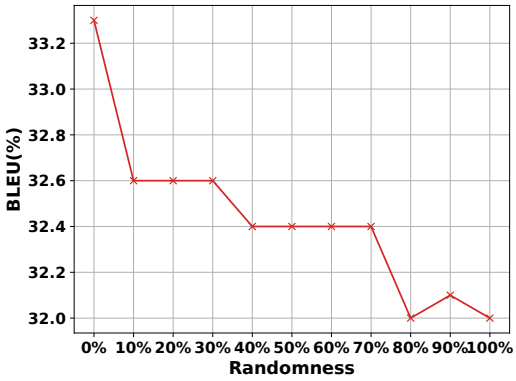

Figure 3: Decrease in performance when adding more random noise to attention words. BLEU scores for IWSLT 2017 FR-EN test data.

5.3 Ablation Test

To understand how the quality of our explainable features impacts the accuracy improvement, we conduct an ablation study. Figure 3 depicts the NMT BLEU score by adding different percentages of random noise into the explainable features incorporated. Specifically, to add $r\%$ of randomness, $r\%$ of the attention words of each sentence are uniformly randomly sampled from the source sentence words. The red curve draws the accuracy of the French-English IWSLT'17 ConvS2S model. We observe that by adding noise into the explainable features, we decrease their quality, thus reducing the BLEU score's improvement over the baseline. Therefore, our ablation analysis indicates that high-quality interpretation of the NLP model helps more on NLP prediction accuracy.

6 Related Work

There have been many works in interpretable deep learning, such as [Murdoch *et al.*, 2019; Li *et al.*, 2021]. There are many works evaluating the interpretability methods such as [Li *et al.*, 2020; Jacovi and Goldberg, 2020]. To our knowledge, we are the first to apply the interpretability results using attention to improve the NLP task accuracy.

There are also work in adding input features into NLP systems, such as adding linguistic features such as [Sennrich and Haddow, 2016] in NMT. None of them used the interpretation results from the same system as features to enhance NMT or NLP in general. The gain of linguistic or BERT features comes from the knowledge extracted from another application, while the accuracy improvement by our approach is to simplify the learning.

7 Conclusion

We show that incorporating interpretability results into deep learning models can effectively reduce learning difficulties, thus enhancing NLP accuracy and generalization. We demonstrate significant accuracy gain on NMT and GLUE. Our introduced paradigm is novel and can be applied with a set of interpretability methods independent of the learning task, serving as a framework that facilitates more possibilities in integrating quality- and explainability-driven research work.

Acknowledgments

We appreciate the National Science Foundation (NSF) Award No. 1747728 to fund this research. We are also thankful for the support of the Google Cloud Research Program.

References

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[Du *et al.*, 2019] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2019.

[Gehring *et al.*, 2017] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR, 2017.

[Gomez *et al.*, 2021] Tristan Gomez, Suiyi Ling, Thomas Fréour, and Harold Mouchère. Improve the interpretability of attention: A fast, accurate, and interpretable high-resolution attention model. *arXiv preprint arXiv:2106.02566*, 2021.

[Jacovi and Goldberg, 2020] Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? *arXiv preprint arXiv:2004.03685*, 2020.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Koehn *et al.*, 2007] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180, 2007.

[Li *et al.*, 2020] Jierui Li, Lemao Liu, Huayang Li, Guanlin Li, Guoping Huang, and Shuming Shi. Evaluating explanation methods for neural machine translation. *arXiv preprint arXiv:2005.01672*, 2020.

[Li *et al.*, 2021] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *arXiv preprint arXiv:2103.10689*, 2021.

[Lundberg and Lee, 2017] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. 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 4765–4774. Curran Associates, Inc., 2017.

[Murdoch *et al.*, 2019] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

[Ott *et al.*, 2019] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[Post, 2018] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[Ribeiro *et al.*, 2016] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[Sennrich and Haddow, 2016] Rico Sennrich and Barry Haddow. Linguistic input features improve neural machine translation. *arXiv preprint arXiv:1606.02892*, 2016.

[Sennrich *et al.*, 2015] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[Serrano and Smith, 2019] Sofia Serrano and Noah A Smith. Is attention interpretable? *arXiv preprint arXiv:1906.03731*, 2019.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[Wang *et al.*, 2018] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[Wolf *et al.*, 2020] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.