AllenNLP Interpret:
A Framework for Explaining Predictions of NLP Models

Eric Wallace[1] **Jens Tuyls**[2] **Junlin Wang**[2] **Sanjay Subramanian**[1]
Matt Gardner[1] **Sameer Singh**[2]
[1]Allen Institute for Artificial Intelligence [2]University of California, Irvine
ericw@allenai.org, sameer@uci.edu

Abstract

Neural NLP models are increasingly accurate but are imperfect and opaque—they break in counterintuitive ways and leave end users puzzled at their behavior. Model interpretation methods ameliorate this opacity by providing explanations for specific model predictions. Unfortunately, existing interpretation codebases make it difficult to apply these methods to new models and tasks, which hinders adoption for practitioners and burdens interpretability researchers. We introduce AllenNLP Interpret, a flexible framework for interpreting NLP models. The toolkit provides interpretation primitives (e.g., input gradients) for any AllenNLP model and task, a suite of built-in interpretation methods, and a library of front-end visualization components. We demonstrate the toolkit's flexibility and utility by implementing live demos for five interpretation methods (e.g., saliency maps and adversarial attacks) on a variety of models and tasks (e.g., masked language modeling using BERT and reading comprehension using BiDAF). These demos, alongside our code and tutorials, are available at https://allennlp.org/interpret.

1 Introduction

Despite constant advances and seemingly superhuman performance on constrained domains, state-of-the-art models for NLP are imperfect: they latch on to superficial patterns (Gururangan et al., 2018), reflect unwanted social biases (Doshi-Velez and Kim, 2017), and significantly underperform humans on a myriad of tasks. These imperfections, coupled with today's advances being driven by (seemingly black-box) neural models, leave researchers and practitioners scratching their heads, asking, "*why did my model make this prediction?*"

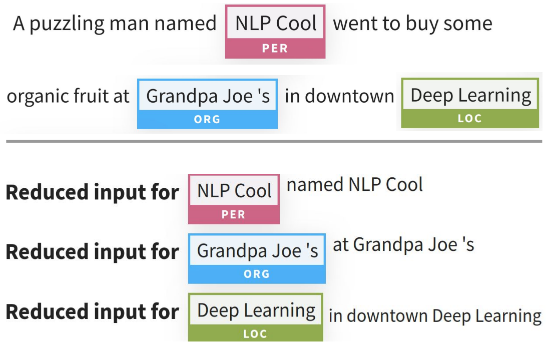


Figure 1: An interpretation generated using AllenNLP Interpret for NER. The model predicts three tags for an input (top). We interpret each tag separately, e.g., input reduction (Feng et al., 2018) (bottom) removes as many words as possible without changing a tag's prediction. Input reduction shows that the words "named", "at", and "in downtown" are sufficient to predict the People, Organization, and Location tags, respectively.

Instance-level interpretation methods help to answer this question by providing explanations for specific model predictions. These explanations come in many flavors, e.g., visualizing a model's local decision boundary (Ribeiro et al., 2016), highlighting the saliency of the input features (Simonyan et al., 2014), or adversarially modifying the input (Ebrahimi et al., 2018). Interpretations are useful to illuminate the strengths and weaknesses of a model (Feng et al., 2018), increase user trust (Ribeiro et al., 2016), and evaluate hard-to-define criteria such as safety or fairness (Doshi-Velez and Kim, 2017).

Many open-source implementations exist for instance-level interpretation methods. However, most codebases focus on computer vision, are model- or task-specific (e.g., sentiment analysis), or contain implementations for a small number of interpretation methods. Thus, it is difficult for practitioners to interpret *their* model. As a re-

Proceedings of the 2019 EMNLP and the 9th IJCNLP (System Demonstrations), pages 7–12
Hong Kong, China, November 3 – 7, 2019. ©2019 Association for Computational Linguistics

sult, model developers rarely leverage interpretations and thus lack a robust understanding of their system. The inflexibility of existing interpretation codebases also burdens interpretability researchers—they cannot easily evaluate their methods on multiple models.

We present AllenNLP Interpret, an open-source, extensible toolkit built on top of AllenNLP (Gardner et al., 2018) for interpreting NLP models. The toolkit makes it easy to apply existing interpretation methods to *new models*, as well as develop *new interpretation methods*. The toolkit consists of three contributions: a suite of interpretation techniques implemented for broad classes of models, model- and task-agnostic APIs for developing new interpretation methods (e.g., APIs to obtain input gradients), and reusable front-end components for interactively visualizing the interpretations.

AllenNLP Interpret has numerous **use cases**. Our external website shows demos of:

- *Uncovering Model Biases:* A SQuAD model relies on lexical overlap between the words in the question and the passage. Alternatively, a textual entailment model infers contradiction on observing the word "politics" in the hypothesis.
- *Finding Decision Rules:* A named entity recognition model predicts the location tag when it sees the phrase "in downtown".
- *Diagnosing Errors:* A sentiment model incorrectly predicts the positive class due to the trigram "tony hawk style".

2 Interpreting Model Predictions

This section introduces an end user's view of our toolkit, i.e., the available interpretations, models, and visualizations.

2.1 What Are Instance-Level Interpretations

AllenNLP Interpret focuses on two types of interpretations: gradient-based saliency maps and adversarial attacks. We choose these methods for their flexibility—gradient-based methods can be applied to any differentiable model.

Saliency maps explain a model's prediction by identifying the importance of the input tokens. Gradient-based methods determine this importance using the gradient of the loss with respect to the tokens (Simonyan et al., 2014).

Adversarial attacks provide a different lens into a model—they elucidate its capabilities by exploiting its weaknesses. We focus on methods that modify tokens in the input (e.g., replace or remove tokens) in order to change the model's output in a desired manner.

2.2 Saliency Map Visualizations

We consider three saliency methods. Since our goal is to interpret why the model made *its* prediction (not the ground-truth answer), we use the model's own output in the loss calculation. For each method, we reduce each token's gradient (which is the same dimension as the token embedding) to a single value by taking the L_2 norm.

Vanilla Gradient This method visualizes the gradient of the loss with respect to each token (Simonyan et al., 2014). Figure 2 shows an example interpretation of BERT (Devlin et al., 2019).

Integrated Gradients Sundararajan et al. (2017) introduce integrated gradients. They define a baseline x', which is an input absent of information (we use a sequence of all zero embeddings). Word importance is determined by integrating the gradient along the path from this baseline to the original input.

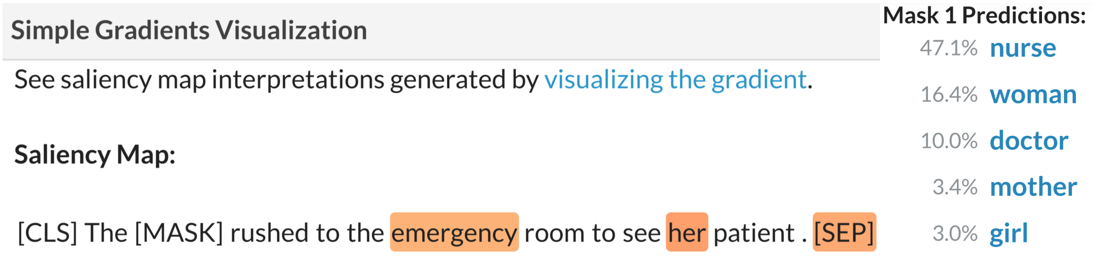
SmoothGrad Smilkov et al. (2017) average the gradient over many noisy versions of the input. For NLP, we add small Gaussian noise to every embedding and take the average gradient value.

2.3 Adversarial Attacks

We consider two adversarial attacks: replacing words to change the model's prediction (HotFlip) and removing words to maintain the model's prediction (Input Reduction).

Untargeted & Targeted HotFlip We consider word-level substitutions using HotFlip (Ebrahimi et al., 2018). HotFlip uses the gradient to swap out words from the input in order to change the model's prediction. It answers a sensitivity question: *how would the prediction change if certain words are replaced?* We also extend HotFlip to a targeted setting, i.e., we substitute words in order to change the model's prediction to a *specific* target prediction. This answers an almost counterfactual question: *what words should be swapped in order to cause a specific prediction?*

We closely follow the original HotFlip algorithm: replace tokens based on a first-order Taylor



Simple Gradients Visualization

See saliency map interpretations generated by visualizing the gradient.

Saliency Map:

[CLS] The [MASK] rushed to the emergency room to see her patient . [SEP]

Mask 1 Predictions:

47.1% nurse

16.4% woman

10.0% doctor

3.4% mother

3.0% girl

Figure 2: A saliency map generated using Vanilla Gradient (Simonyan et al., 2014) for BERT's masked language modeling objective. BERT predicts the [MASK] token given the input sentence; the interpretation shows that BERT uses the gendered pronoun "her" and the hospital-specific "emergency" to predict "nurse".

approximation of the loss around the current token embeddings.[1] Figure 3 shows an example of a HotFlip attack on sentiment analysis.

Input Reduction Feng et al. (2018) introduce input reduction. They remove as many words as possible from the input *without* changing a model's prediction. Input reduction works by iteratively removing the word with the smallest gradient value. We classify input reduction as an "adversarial attack" because the resulting inputs are usually nonsensical but cause high confidence predictions (Feng et al., 2018). Figure 1 shows an example of reducing an NER input.

2.4 Currently Available Models

The toolkit currently interprets six tasks which cover a wide range of input-output formats and model architectures.

- **Reading Comprehension** using the SQuAD (Rajpurkar et al., 2016) and DROP (Dua et al., 2019) datasets. We use NAQANet (Dua et al., 2019) and BiDAF models (Seo et al., 2017).
- **Masked Language Modeling** using the transformer models available in Pytorch Transformers[2], e.g., BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and more.
- **Text Classification** and **Textual Entailment** using BiLSTM and self-attention classifiers.
- **Named Entity Recognition (NER)** and **Coreference Resolution**. These are examples of tasks with complex input-output structure; we can use the same function calls to analyze each predicted tag (e.g., Figure 1) or cluster.

3 AllenNLP Interpret Under the Hood

This section provides implementation details for AllenNLP Interpret: how we compute the token embedding gradient in a model-agnostic way, as well as the available front-end interface. Figure 4 provides an overview of our software implementation and the surrounding AllenNLP ecosystem.

3.1 Model-Agnostic Input Gradients

Existing Classes in AllenNLP Models in AllenNLP are of type Model (a thin wrapper around a PyTorch Module). The Model wrapper includes a forward() function, which runs the model and optionally computes the loss if a label is provided.

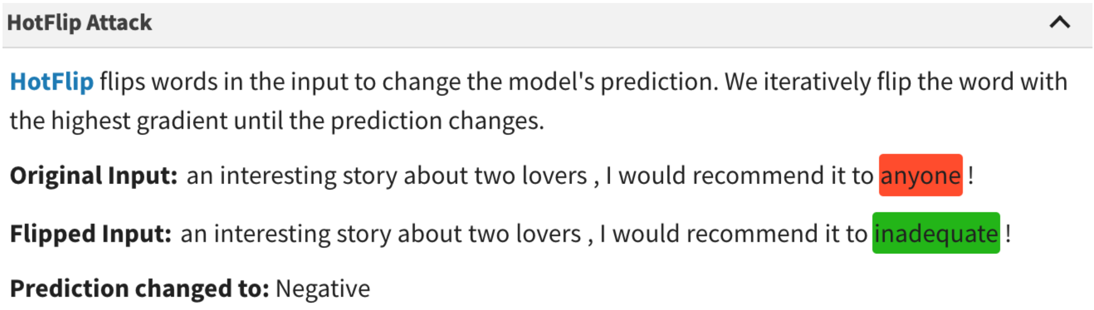
Obtaining predictions from an AllenNLP Model is simplified via the Predictor class. This class provides a model-agnostic way for obtaining predictions: call predict_json() with a JSON containing raw strings and it will return the model's prediction. For example, passing {"input": "this demo is amazing!"} to a sentiment analysis Predictor will receive positive and negative class probabilities in return.

Our AllenNLP Extension The core backbone of our toolkit is an extension to the Predictor class that allows interpretation methods to compute input gradients in a model-agnostic way. Creating this extension has two main implementation challenges: (1) the loss (with the model's *own predictions* as the labels) must be computed for widely varying output formats (e.g., classification, tagging, or language modeling), and (2) the gradient of this loss with respect to the token embeddings must be computed for widely varying embedding types (e.g., word vectors, ELMo (Peters et al., 2018) embeddings, BERT embeddings).

Predictions to Labeled Instances To handle challenge (1), we leverage the fact that all mod-

[1] We also adapt HotFlip to contextual embeddings; details provided in Section 3.2.

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



HotFlip Attack

HotFlip flips words in the input to change the model's prediction. We iteratively flip the word with the highest gradient until the prediction changes.

Original Input: an interesting story about two lovers , I would recommend it to anyone !

Flipped Input: an interesting story about two lovers , I would recommend it to inadequate !

Prediction changed to: Negative

Figure 3: A word-level HotFlip attack on a sentiment analysis model—replacing "anyone" with "inadequate" causes the model's prediction to change from Positive to Negative.

els will return a loss if a label is passed to their forward() function. We first query the model with the input to obtain its prediction. Next, we convert this prediction into a set of "labeled examples" using a function called predictions_to_labeled_instances(). For categorical predictions (e.g., classification, span prediction), this function returns a single instance with the label set to the model's argmax prediction.

For tasks with structured outputs (e.g., NER, coref), this function returns multiple instances, where each instance is used to compute the loss for a different part of the output. For example, there are separate instances for each of the three NER tags predicted in Figure 1. Separating out the instances allows us to have more fine-grained interpretations—we can analyze one part of the overall prediction rather than interpreting the entire tag sequence.

Embedding-Agnostic Gradients To handle difficulty (2)—computing the gradients of varying token embeddings—we rely on the abstractions of AllenNLP. In particular, AllenNLP uses a TokenEmbedder interface to converts token ids into embeddings. We can thus compute the gradient for any embedding method by registering a PyTorch backward gradient hook on the model's TokenEmbedder function.

Our end result is a simple API for computing input gradients for any model: call predictions_to_labeled_instances() and then get_gradients().

3.2 Context-Independent Embedding Matrix for Deep Embeddings

The final implementation difficulty arises from the fact that contextual embeddings such as ELMo and BERT do not have an "embedding matrix" to search over (their embeddings are context-dependent). This raises difficulties for methods such as Hotflip (Section 2.3) that require searching over a discrete embedding matrix. To solve this, we create a context-independent matrix that contains the features from the model's last context-independent layer. For instance, we pass all of the words from a particular task's training set into ELMo and save the features from its context-independent Char-CNN into a "word embedding matrix". This allows us to run HotFlip for contextual embeddings while still capturing context information since the gradient backpropagates through the contextual layers.

3.3 Frontend Visualizations

We interactively visualize the interpretations using the AllenNLP Demo, a web application for running AllenNLP models. We add HTML and JavaScript components that provide visualizations for saliency maps and adversarial attacks. These components are reusable and greatly simplify the process for adding new models and interpretation methods (Section 4). For example, a single line of HTML code can create the visualizations shown in Figures 1–3. Note that visualizing the interpretations is not required—AllenNLP Interpret can be run in an offline, batch manner. This is useful for aggregating interpretation results, e.g., as in Feng et al. (2018) and Wallace et al. (2018).

4 Adding a Model or Interpretation

This section describes the high-level process for adding new analysis methods or AllenNLP models to our toolkit.

New Interpretation We provide a tutorial for adding a new analysis method to our toolkit. In particular, it walks through the three main requirements for adding SmoothGrad:

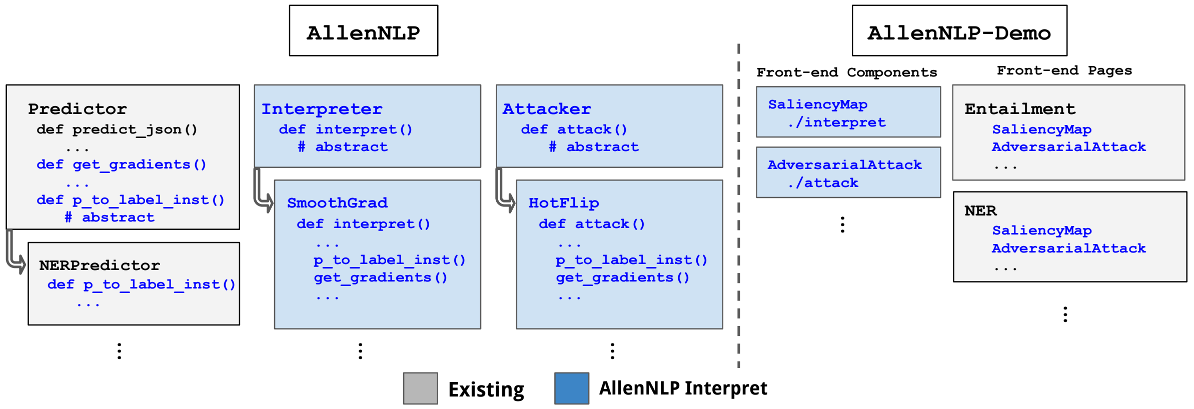


Figure 4: **System Overview:** Our toolkit (in blue) and the surrounding AllenNLP ecosystem. The only model-specific code is a simple function called predictions_to_labeled_instances() (abbreviated as p_to_label_inst()), which is added to the model's Predictor class (e.g., for an NER model's predictor; left of figure). This function allows input gradients to be calculated using get_gradients() in a model-agnostic manner (e.g., for use in SmoothGrad or HotFlip; middle left of Figure). On the front-end (right of Figure), we create reusable visualization components, e.g., for visualizing saliency maps or adversarial attacks.

1. Implementing SmoothGrad in AllenNLP, using predictions_to_labeled_instances() and get_gradients() (requires adding about ten lines of code to the vanilla gradient method).
2. Adding a SmoothGrad Interpreter to the demo back-end (about five lines of code).
3. Adding the HTML/JavaScript for saliency visualization (requires making a one-line call to the reusable front-end components).

New Model We also provide a tutorial for interpreting a new model. If your task is already available in the demos (e.g., text classification), you need to change a *single* line of code to replace the demo model with your model. If your task is not present in the demos, you will need to:

1. Write the predictions_to_labeled_instances() function for your model (consists of three lines for classification).
2. Create a path to your model in the demo's back-end (about 5-10 lines of code).
3. Add a front-end page to visualize the model and interpretation output. This is simplified by the reusable front-end components (consists of copy-pasting code templates).

5 Related Work

Alternative Interpretation Methods We focus on gradient-based methods (saliency maps and adversarial attacks) but numerous other instance-level model interpretation methods exist. For example, a common practice in NLP is to visualize attention weights (Bahdanau et al., 2015) or to isolate the effect of individual neurons (Karpathy et al., 2016). We focus on gradient-based methods because they are applicable to many models.

Existing Interpretation Toolkits In computer vision, various open-source toolkits exist for explaining and attacking models (e.g., Papernot et al. (2016); Ozbulak (2019), inter alia); some toolkits also include interactive demos (Norton and Qi, 2017). Similar toolkits for NLP are significantly scarcer, and most toolkits focus on specific models or tasks. For instance, Liu et al. (2018), Strobelt et al. (2019), and Vig (2019) visualize attention weights for specific NLP models, while Lee et al. (2019) apply adversarial attacks to reading comprehension systems. Our toolkit differs because it is flexible and diverse; we can interpret and attack any AllenNLP model.

6 Conclusion

We presented AllenNLP Interpret, an open-source toolkit that facilitates the interpretation of NLP models. The toolkit is flexible—it enables the development and evaluation of interpretation methods across a wide range of NLP models and tasks.

The toolkit is continually evolving—we will continue to implement new interpretation methods and models as they become available. We welcome open-source contributions, and we hope the toolkit is useful for model developers and interpretability researchers alike.

Acknowledgements

The authors thank Shi Feng, the members of UCI NLP, and the anonymous reviewers for their valuable feedback. We also thank the developers of AllenNLP for their help with constructing our toolkit, especially Joel Grus. This work is supported in part by NSF Grant IIS-1756023.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *ACL*.

Shi Feng, Eric Wallace, Alvin Grissom II, Mohit Iyyer, Pedro Rodriguez, and Jordan Boyd-Graber. 2018. Pathologies of neural models make interpretations difficult. In *EMNLP*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *ACL Workshop for NLP Open Source Software*.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *NAACL*.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. In *ICLR Workshop Track*.

Gyeongbok Lee, Sungdong Kim, and Seung-won Hwang. 2019. QADiver: Interactive framework for diagnosing QA models. In *AAAI Demonstrations*.

Shusen Liu, Tao Li, Zhimin Li, Vivek Srikumar, Valerio Pascucci, and Peer-Timo Bremer. 2018. Visual interrogation of attention-based models for natural language inference and machine comprehension. In *EMNLP*.

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:1907.11692*.

Andrew P Norton and Yanjun Qi. 2017. Adversarial-Playground: A visualization suite showing how adversarial examples fool deep learning. In *2017 IEEE VizSec Symposium*.

Utku Ozbulak. 2019. Pytorch CNN visualizations. https://github.com/utkuozbulak/pytorch-cnn-visualizations.

Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, et al. 2016. Technical report on the CleverHans v2.1.0 adversarial examples library. *arXiv:1610.00768*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should I trust you?: Explaining the predictions of any classifier. In *KDD*.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. In *ICLR*.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR*.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. 2017. SmoothGrad: removing noise by adding noise. In *ICML Workshop on Visualization for Deep Learning*.

Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M Rush. 2019. Seq2Seq-Vis: A visual debugging tool for sequence-to-sequence models. *IEEE TVCG*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *ICML*.

Jesse Vig. 2019. Visualizing attention in transformer-based language models. *arXiv:1904.02679*.

Eric Wallace, Shi Feng, and Jordan Boyd-Graber. 2018. Interpreting neural networks with nearest neighbors. In *EMNLP 2018 Blackbox NLP Workshop*.