Explainable AI

Intro AI Project Proposal

Adam Noack

1 Problem/Motivation

AI is everywhere. Facial recognition software on our phones' cameras, natural language processing capabilities in personal assistants such as Siri, Alexa, and Google Assistant, self driving cars almost necessarily have a large number of different machine learning algorithms to help in the navigation task, Netflix, Spotify, and Amazon all use AI to deliver the content to its users that is most engaging, the list goes on and on.

AI's explosion over the past decade or so has in large part been because of a particular machine learning method called deep learning. Deep learning is a powerful method of learning that trains large neural networks, networks that can have millions of nodes and connections, to approximate all sorts of highly complex functions.

As compute and data become cheaper and more accessible, ever bigger networks are able to be trained. When approaching many problems, the motto seems to be at least to some degree, "the bigger the better." The issue though is that as deep neural networks become deeper, they become progressively more impossible for humans to understand without some sort of simplifying apparatus.

As ever more tasks are offloaded onto machines controlled by extremely complex black-box algorithms, the more we will come to rely on predictions and approximations that we can in no way verify unless we devise methods by which we can interpret the output of a given model.

The approaches described in the following section explain some of the more recent and successful attempts at making these black boxes a bit whiter.

2 Approaches

2.1 Learning with Interpretable Structure from RNN

Recurrent neural networks (RNNs) are very difficult for humans to understand – arguably even more difficult than standard feed-forward neural networks. Their general incomprehensibility is due in large part to the fact that their hidden unit values are represented numerically rather than semantically. Interestingly though, the task that RNNs are so good at performing, interpreting sequences of data, can also be performed by finite state automata (FSA). This is particularly interesting because states and transitions in FSA are represented semantically and are therefore much easier for humans to understand than RNN states and transitions.

It was previously known that the values of the hidden states of classical RNNs tend to clump together. This fact led Bo-Jian Hou to believe that the hidden states of classical RNNs could be represented as semantically separate states in an FSA. I.e. each clumping of hidden unit values could be interpreted as a separate state in an FSA.

However, classical RNN units suffer from the vanishing or exploding gradient issue, so they have in large part been deprecated in favor of gated recurrent units. It was not known if the values of gated RNN units such as the long short-term memory (LSTM) unit, gated recurrent units (GRUs), or the minimal gated unit (MGU) clumped together. After training several separate gated unit RNNs in [2], Hou determined that the values of gated units also clump together. With this information at hand, Hou set to work converting trained gated RNNs into equivalent FSA.

Hou did this in the following manner: He first trained gated RNNs on data. He then used k-means to find clusters of semantically similar hidden state values. These clusters were then made into the states of the FSA. The FSA's vocabulary was then created by simply enumerating all of the possible inputs to the RNN. A transition matrix was then learned

that defined how one should transition given the current state and the next input.

Hou obtained interesting results. He found that of the three different types of gated unit RNNs tested, MGU RNNs could be represented by an FSA with the fewest number of states. This implies that MGUs are much easier to interpret via FSA than LSTMs and GRUs. Given the fact that MGUs tend to be only slightly outperformed by LSTMs and GRUs, MGUs might be the better option if one desires performance *and* interpretability.

2.2 Anchors: High-Precision Model-Agnostic Explanations

Ribeiro, the creator of the LIME approach [4], introduces anchors in [6], a new method by which black box machine learning models can be explained.

Take a rule, a set of predicates for a given dataset, and for each data sample determine which of those predicates are present in the sample. The rule is an anchor for a particular sample if each predicate is present in the sample and the presence of all the predicates in the rule is highly correlated with a particular output class. For example, if the set of words $A = \{`not', `bad'\}$ is a rule for a set of text data, $A(x_i)$ will return 1 and be an anchor for x_i if all of the feature predicates are present in a sample x_i. Assuming that $A(x_i) = 1$ predicts the sentiment "positive" for this dataset with sufficiently high probability, A is an *anchor* for the sample $x_1 =$ "*This movie is not bad.*" because $A(x_1) = 1$. The anchor can then be interpreted as an explanation for the class prediction.

How are anchors computed? One method by which Ribeiro computes anchors for a given feature space begins with an empty set of predicates $B = \{\}$ and iteratively adds new features to B. In other words, each time through the search loop, the most explanatory, unused feature predicate is selected and added to B. The search loop terminates when the class for a given sample x_i can be predicted with high probability given only the fact that $B(x_i) = 1$.

Anchors are useful because they are extremely easy and intuitive for humans to interpret. Simple rules such as the presence of a few words, in the case of textual data, or the presence of a few superpixels, in the case of image data, are easily communicated to and understood by the user.

2.3 Learning to Explain: An Information-Theoretic Perspective on Model Interpretation

Learning to Explain [3], or L2X as it's called, functions at a high level in the following manner. For a given input sample x existing in $\mathbb{R}^d$, L2X seeks to find a vector x_s existing in $\mathbb{R}^k$ where $k < d$ and the exact value for k is defined by the user of the model. In other words, like many of the other approaches to model explanation, L2X produces explanations by rank-ordering the features for a particular sample that are most influential in that sample's prediction.

L2X is a bit different, though, in the way it decides on the most important features for a sample. It finds the subset of features x_s from the original sample that maximizes the mutual information between x_s and the correct output value for the sample, y.

Mutual information can be thought of as the amount of information gained about one event given the observation of a different event. If the mutual information between x_s and y is high, conceptually, that means that observing x_s gives us a lot of insight into y's value.

A direct method for finding the subset of features that maximizes mutual information is not possible, so in this paper the solution is approximated using a variational approach.

Chen, et al. derive a lower bound on the mutual information. They do this by recognizing the fact that the mutual information between x_s and y can be expressed in terms of the conditional distribution of y given x_s.

2.4 Extracting Rules from Artificial Neural Networks with Distributed Representations

In [5], Sebastian Thrun uses validity interval analysis to generate easily understood rules for a neural network that predicts whether or not a robotic arm is in a safe position or not.

Thrun starts off by providing the motivation for this line of research. He explains that many common methods for generating rules for neural networks are not applicable in cases where the internal representation of the neural network is distributed in nature, i.e., the network is not sparse, and the activations are continuous as opposed to binary.

Validity interval analysis works by attaching intervals to every neuron's activation value. For example, for neuron x_i, $\sigma(x_i) \in [a_i, b_i]$. For the robotic arm example, there are two classes of output, "safe" and "unsafe". A single output neuron can be used to cover the output space: the value for x_{output} will be predicted "safe" if $\sigma(x_{output}) \geq .5$ and "unsafe" if $\sigma(x_{output}) < .5$. It's true then that any combination of input features that satisfies the constraint that the output neuron's activation value must lie in the interval $[0, .5)$ to will always be classified as "unsafe".

In practice, constructing rules for neural networks using validity interval analysis is usually done in one of two ways, specific to general or general to specific. Thrun took a specific to general approach. He would start with a single data sample, initially assigning each neuron x_i's interval to be $[\sigma(x_i), \sigma(x_i)]$. From there, the intervals for each neuron were gradually widened until the output prediction class changed. The range of input values that "worked" in that range became a rule for that output class. For example, if the input to the net is a five element array, a, and the ith feature of a can be accessed by a_i, then a learned rule for the network might be written as follows: *if* $a_2 < 50$ *and* $a_4 > 75$ *then "safe"*.

3 My Demonstration

I plan to train a neural network on a text corpus and implement the Anchors algorithm to explain the model's predictions.

3.1 Data

I plan to use the Rotten Tomatoes dataset. The training set contains 156,060 phrases from movie reviews posted on the Rotten Tomatoes website. The test set contains 66,292 phrases. Each phrase is attached to a sentiment label. The sentiment labels are: 0 - negative, 1 - somewhat negative, 2 - neutral, 3 - somewhat positive, 4 - positive.

3.2 Model

I will most likely use Google's Tensorflow [1] library to build an RNN for classifying the phrases using Python 3.

3.3 Generating Anchors

Of the two approaches to generating anchors that Ribeiro mentions in [6], I am going to use the "Bottom-up" approach.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens,

Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Zhi-Hua Zhou Bo-Jian Hou. Learning with interpretable structure from rnn. *National Key Laboratory for Novel Software Technology*, 2018.

[3] Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. *CoRR*, abs/1802.07814, 2018.

[4] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[5] Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, pages 505–512, Cambridge, MA, USA, 1994. MIT Press.

[6] T. Yang, X. Zhang, Z. Li, W. Zhang, and J. Sun. MetaAnchor: Learning to Detect Objects with Customized Anchors. *ArXiv e-prints*, July 2018.