Fabrice Harel-Canada
CS 239 Winter 2019
Professor Miryung Kim

MODE Project Report

March 10, 2019

ABSTRACT

The source code for MODE debugging has not been open sourced and the authors have, as of yet, not responded to requests to release the related repositories. This effort sought to replicate the various components of MODE so that it may be used for debugging. Beyond the baseline goal of replication, this effort also sought 1) to explore the parameterization of similarity metrics for input selection and 2) determine if the relatively time-intensive layer selection process actually adds to model performance.

Unfortunately, various ambiguities in the original paper and other challenges made it difficult to create a fully successful implementation. This report will describe these challenges and the decisions made to overcome them. The results of comparing 9 different distance / similarity metrics indicated that the original choice of using the dot product is suboptimal and debugging via differential state analysis would (rather unintuitively) be improved by using the “earth mover” (EM) / wasserstein distance instead. Performance is also not significantly aided by selecting the output from an intermediate layer of the model, suggesting that additional time savings could be realized by forgoing this step and always building heat maps from the predictions of the models normal output layer. [1]

[1] Project code can be found here: https://github.com/fabriceyhc/mode_nn_debugging

INTRODUCTION

In *MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection*, Shiqing Ma et al. present an approach to identify training bugs in neural networks (NN) by generating heat maps of pixel areas that contribute most to correct or incorrect class predictions. These heat maps are then used as a criteria for selecting the next batch of input samples depending on the type of errors observed. For overfitting errors, more dissimilar inputs are provided so that the model can generalize it's notion of the difficult class cases. For underfitting errors, more similar inputs are provided so that the model learns a class's structure more concisely.

IMPLEMENTATION

The primary functions were organized into separate .py files to group related functions and hide many hundreds of lines of code that would have otherwise cluttered the main notebook where the experiments took place. This effort was implemented in Python and and the neural networks built in TensorFlow / Keras.

A simple three layer model consisting of 3 fully connected layers of 28 neurons each, except for the final layer which contained 10 for the softmax outputs. 28 neurons were selected to decrease training time and for the purpose of visualizing the weights of the layer in a dimension compatible with the input images.

The baseline replication effort relied on the descriptions contained within the original paper. Perhaps unsurprisingly, the form of an academic research paper differs significantly from a proper technical specification and many ambiguities or otherwise omitted details resulted the forcing several design decisions that will be covered in the following Challenges section.

The experiments conducted as extensions to the MODE paper include the following:

1. **Exploration of alternative distance metrics.**

   In the pursuit of the hypothesis that the original dot product used in the paper is suboptimal for image tasks, 9 metrics were used to select the next batch of inputs. These metrics include:

   - dot product
   - cosine similarity
   - manhattan (L1) distance

- euclidean (L2) distance
- minkowski (Lp) distance
- chebyshev (L-inf) distance
- earth mover (wasserstein) distance
- canberra distance
- bray curtis dissimilarity

In order to determine which metric was able to leverage the information contained within the differential state heat maps, two steps were taken - simple visual inspection and comparison of overall improvement in model performance after retraining with that metric. Visual inspection was used as an informal way of building an intuition for whether a given metric is well suited the task of identifying helpful data from the bug-fixing data set. It was also used to form a further hypothesis that the most normal looking set of most similar images - as judged by this author - would make for the best metric to use. This resulted in selection of the the bray curtis dissimilarity as the most likely to improve model performance.

However, it would not be sufficient to suspect improved performance on visual inspection alone. Therefore, each metric was used in the retraining process for a model to see if it could be used to generate a customized batch for underfitting/overfitting issues and improve performance more than the control case of using an equal amount random bug-fixing data on a cloned version of the original model.

2. **Analysis of whether target layer selection is really helpful.**

   A second project extension that arose in response to the relatively high training times associated with layer selection is seeing if we can forgo this step and exclusively use the differential state heat maps generated from the final output layer to improve performance. The setup for this experiment was exactly the same as for the previous extension with the exception that the layer selection step was skipped and therefore, instead of generating the heatmaps from some internal layer's output performance, we exclusively used the model's final layer. [2]

Other implementation details related to hyperparameter selections for overfitting, underfitting, sufficient bhattacharyya similarity (never covered in the paper), and target label vs. random input ratio can be seen in the second cell of the notebook referenced above.

---

[2] It is worth noting that this approach yield the same results as the layer selection process picking the final layer.

CHALLENGES

Target Layer Selection

There were a number of ambiguities surrounding the target layer selection in MODE. The following comprises most of the issues that came up during the implementation:

1. **Do we prioritize layer selection for underfitting or overfitting first?**

   If we only have overfitting or underfitting, then this issue is not a problem. However, when we have underfitting on some labels and overfitting on others, it was not clear how we should go about prioritizing which to pursue first. I decided that it would be best to address the *underfitting* issues first because our primary objective was increasing accuracy rather than addressing overfitting, which is primarily concerned with the difference between train and test metrics, not the actual level of performance.

   The general approach was to address all buggy labels for underfitting issues for a maximum number of iterations (set to 5 for this experiment), then proceed to address any overfitting bugs, if any, for the same number of iterations. After that, it could be argued that we should again check for underfitting that might have resulted from the overfitting adjustments. This was considered and omitted in the current implementation because of limitations in the number of samples for a target for a particular class label.

2. **How often do you select a faulty target layer?**

   The first target layer is selected after initial training based on whether labels have an underfitting or overfitting issue. After selecting the target layer and generating heat maps to support the selection of the next batch of inputs, should we again evaluate which layer is now our target or should we assume that layer selection happens only once for our retraining? I decided that it made the most sense to check for a layer after training on our selected input batches.

   Supporting evidence for the correctness of this design choice can be seen in the occasional selection of different target layers in different iterations of training. However, this approach entails a large training overhead that is at odds with the time savings reported in the original paper. For this experiment, I used the same size batches of 2,000

samples each and limited the iterations to 5 so that a max of 10,000 samples were used from the bug-fixing data set. The average time was 26 minutes compared to 6 minutes in the paper, an approximately 4.3x difference on half the amount of data allotted in the paper for retraining (up to 20,000 data points).

3. **How do you handle the wide variety of layers when creating feature submodels?**

    a. **Do dropout, pooling, flattening layers count as layers?**

    One of the shortcoming of generating multiple models and coming the Bhattacharyya distance of their outputs is that some steps distort the output predictions at regular intervals and makes it difficult to compare layer to layer performance meaningfully. This is the case with dropout, pooling, and flattening layers. Dropout and pooling layers caused performance issues when they were not followed by more Dense (fully-connected) layers, but aided performance when they were, which created a noisy zigzag pattern in the comparisons from layer to layer. Flattening had no effect on the performance relative to a submodel that lacked it and so it always forced the algorithm to consider it as the saturation point even though performance could be improved beyond that, as evidenced by a complete model.

    In order to overcome these challenges, I simply did not include any of these layers in the experiments, although developing some rules for handling these types of layers might be a fruitful direction for future work.

    b. **If a layer is not fully connected, how do we associate it's outputs to the final layer?**

    Convolutional layers are very common in machine learning models today. One of the reasons they're so popular is because they can drastically cut down on the number of parameters in a model by sharing weights between nearby neurons. However, this means that each neuron is not fully connected to the neurons in subsequent layers. This poses a problem for the generation of submodels and connecting them to the output layer and the paper did not cover how they would propose handling them.

    For the sake of simplicity, the experiment did not include models with such layers.

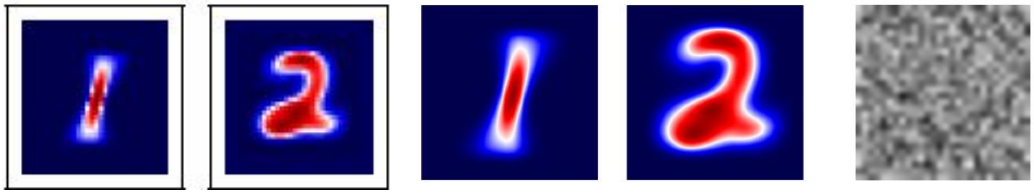
Heatmaps

Another important ambiguity in the paper centers around how heatmaps are actually created, with the main uncertainty focusing on whether they are generated from model weights or from some process applied directly to a subset of the training images.

The authors state that the heat map is

> *an image whose size equals to the number of neurons and the color of a pixel represents the importance of a neuron.*

So it seems pretty clear that the heat maps are based on neuron weights. However, throughout the paper, images are presented of the heat maps as pictures that more or less look like the input images (Figure 1 Left). In general, model weights tend to not look anything like their inputs, but are rather abstractions of different parts of an image that are used as identifying them. It is rather surprising that they looked so similar to the inputs while the visualizations of the weights I was able to come up (Figure 1 Right) with look more or less like noise despite achieving a comparable model performance. When I instead used an approach that averaged the image of all the target training points, I arrive at heat maps that look very similar to the ones in the paper (Figure 1 Center).



Left Center Right

Figure 1. (Left) Heat maps from the paper - generated from model weights. (Center) Heat maps from my experiments based on the average of all the targeted inputs, NOT the model weights. NOTE: I used a gaussian interpolation to smooth out my images whereas the original left them in their normal pixelated state. (Right) Example of a heatmap generated from weights in my experiments.

Furthermore, if the heat maps were actually generated from model weights, it seems like it would be a limiting constraint on the models because the dimensions of the heat map have to be compatible with the input images for the important step of assessing similarity and therefore input selection. For this purpose, I deliberately chose neuron counts per layer to be 28 x 28 so that I could actually form an image of comparable size. It is not at all clear how the models they picked up from various open source repositories, all of which have layers of different sizes, could have produced heat maps that could be used for comparison since at least one of the dimensions has to be 28 or 784 if flattened. Ultimately, because the results seemed so at odds with the paper, I decided to proceed with heat maps based on the average of images. This results in three

different sets of heat maps - one for the correctly classified images (DHCI) and two more for the false positives (DHMI) and false negatives (DHWI), respectively.

RESULTS

Table 1 shows the results of testing for accuracy and training loss compared across three points - the initial values after training on the training set alone, the final values after 10,000 training images with input selection based on differential state analysis, and a control of 10,000 training images selected from the bug-fixing data set at random. It is worth noting that MODE was not particularly effective at resolving buggy labels as it was only successful at raising the per-label accuracy above the pre-set threshold of 0.92 in only one case. It also degraded performance relative to the control model more frequently than it helped.

All of the results are fairly close to one another, with the strongest performer being, rather unintuitively, the earth mover distance. However, with an alpha of 0.05 and p-value of 0.41, none of these results are statistically significant and this may be due to the fact that the model had reached its limit of performance and could not be improved further. In the original paper, the authors had models with initial accuracies in the low 0.80s and they were able to improve them up to 0.94 with their approach, so it looks like the experimental model was already too performant to benefit much from this approach.

Metric	Initial Acc	Final Acc	Control Acc	Finished Early	Diff I-F	Diff F-C
dot product	0.9412	0.9451	0.9458	0	0.0039	-0.0007
cosine	0.9473	0.9472	0.9464	0	-0.0001	0.0008
manhattan	0.9406	0.9415	0.9466	0	0.0009	-0.0051
euclidean	0.9437	0.9411	0.9500	0	-0.0026	-0.0089
minkowski	0.9466	0.9373	0.9394	0	-0.0093	-0.0021
chebyshev	0.9479	0.9447	0.9470	0	-0.0032	-0.0023
earth mover	0.9465	0.9485	0.9425	0	0.0020	0.0060
canberra	0.9416	0.9434	0.9438	0	0.0018	-0.0004
bray curtis	0.9502	0.9447	0.9508	1	-0.0055	-0.0061

Table 1. Results of different distance metrics on performance. Finished early indicates that all labels exceeded the underfitting threshold. I, C, and F in the final two columns stands for Initial (I), Final (F), and Control (C) accuracy comparisons..

Figure 2 provides a visual inspection of the best (most similar) bug-fixing samples for different distance metrics used for input selection. The original hypothesis as to which metric would perform the best in improving model performance was based on reviewing this figure and

selecting the metric that looked to produce the most normal looking numbers that were similar to the heat maps. To me, this meant the bray curtis metric. Counter to intuition, it was one that produced probably the worst looking examples that ended up doing the best overall - the earth mover distance. However, this is based on a single test of the metrics. Additional iterations of the experiment were not conducted because it took about 4 hours to complete this one pass on my single CPU set-up.

Figure 3 provides the worst examples from the bug-fixing data set as measured by each of the distance metrics evaluated in this project. As we can see from the results, all metrics are reasonably good at identifying unusually written numbers that might be hard for a human to classify.

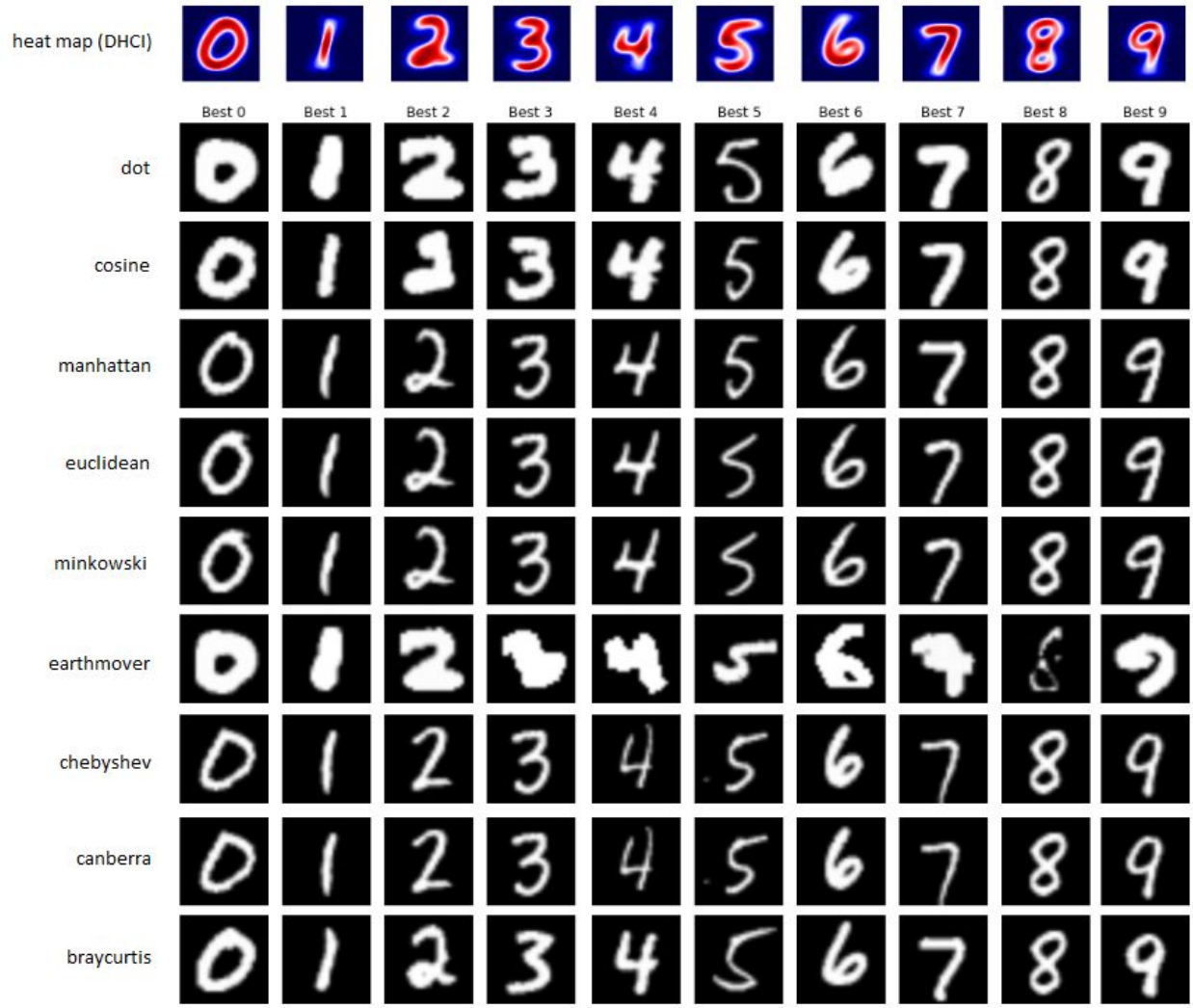


Figure 2. Bug fixing data samples judged *most* similar to the heat map per metric.

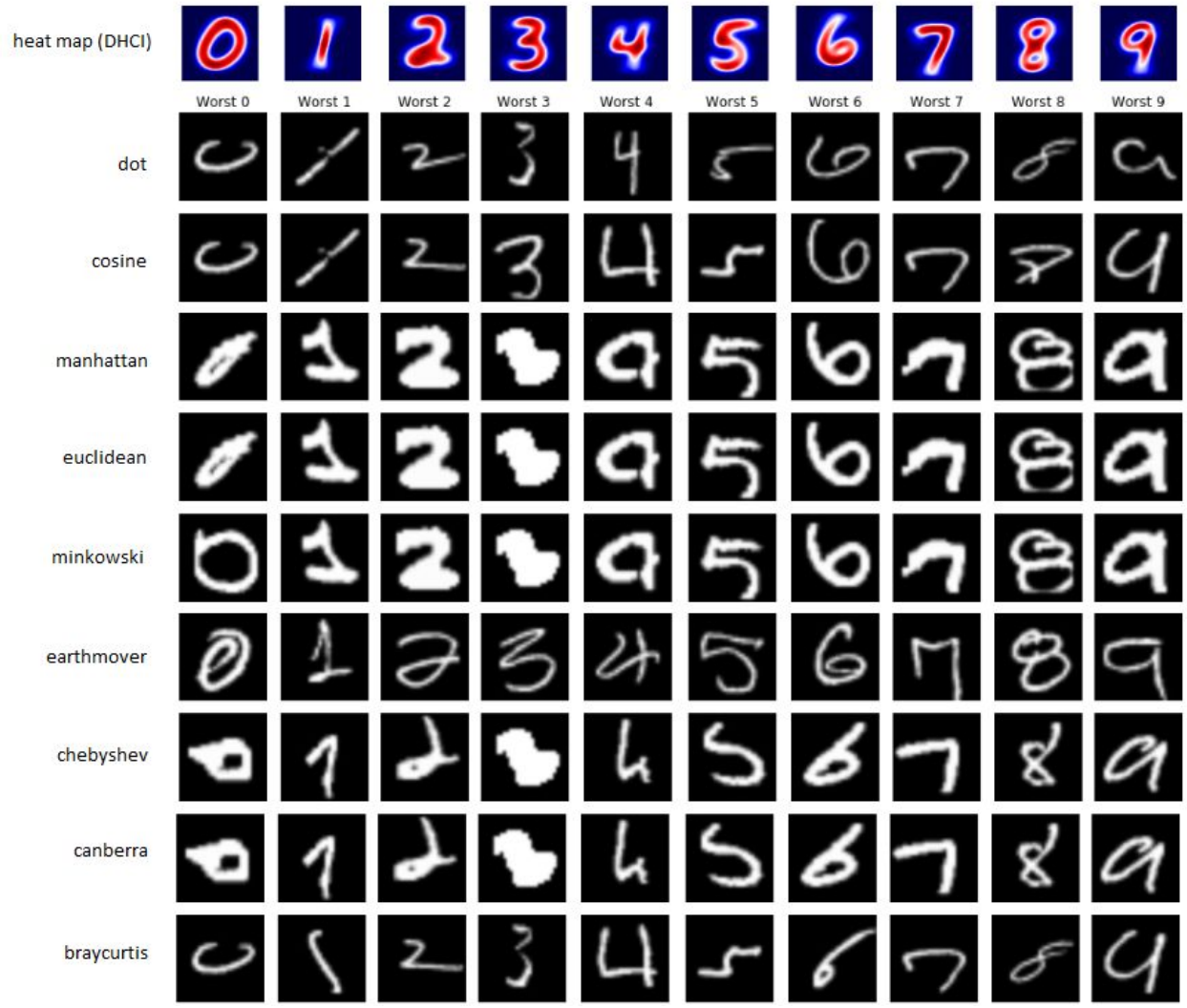


Figure 3. Bug fixing data samples judged *least* similar to the heat map per metric.

Lastly, the results of the experiment to see if model selection made a significant impact on performance gains showed that it did not. With a alpha of 0.05, the difference in the final accuracies yielded a p-value of 0.87. Layer selection is a time consuming step in the MODE algorithm and forgoing it in favor of simply generating heat maps from the final layer output predictions could make the approach faster and therefore more appealing to prospective users.

Metric	Initial Acc	Final Acc	Control Acc	Finished Early	Diff I-F	Diff F-C
dot product	0.9436	0.9438	0.9483	0	0.0002	-0.0045
cosine	0.9481	0.952	0.9481	0	0.0039	0.0039
manhattan	0.9452	0.9356	0.9439	0	-0.0096	-0.0083
euclidean	0.9445	0.9458	0.9438	0	0.0013	0.0020
minkowski	0.945	0.9405	0.9438	0	-0.0045	-0.0033
chebyshev	0.9439	0.9426	0.9439	0	-0.0013	-0.0013
earth mover	0.9468	0.9435	0.9485	1	-0.0033	-0.0050
canberra	0.9502	0.9435	0.9490	0	-0.0067	-0.0055
bray curtis	0.9462	0.9435	0.9434	1	-0.0027	0.0001

Table 2. Results of different distance metrics on performance *without layer selection*.

FUTURE RESEARCH

Future research directions include:

1. Extending layer selection to handling the nuances of different kinds of common layers (and their combinations) used in neural networks when constructing feature submodels. [3]
2. Investigating how to the use of the least similar DHCI images from Figure 3 would compare to using the false positive DHMI and false negative DHWI images for suppression of overfitting. In the original paper, the authors only discuss using images that are the most similar to the heat maps.
3. Investigating the relationship between the number of classes and the optimal selected vs random inputs for retraining. As of right now, this is an empirically set hyper-parameter, but it would be helpful to automatically set this ratio as a function of the number of classes so that prospective users have one less dimension in their hyperopt space.

REFERENCES

[1] Ma, Shiqing & Liu, Yingqi & Lee, Wen-Chuan & Zhang, Xiangyu & Grama, Ananth. (2018). MODE: automated neural network model debugging via state differential analysis and input selection. 175-186. 10.1145/3236024.3236082.

[2] Deza MM, Deza E. Encyclopedia of Distances. Berlin: Springer; 2009

[3] This effort has shown that layer selection is probably not very valuable to the retraining process, but if it can be shown that my implementation is flawed in some way and it actually is important, then this would be an important step for the viability of MODE in production projects.