CS230

Topological Data Analysis and Deep Learning

Anjan Dwaraknath
Institute for Computational and Mathematical Engineering
Stanford University
anjandn@stanford.edu

Abstract

Convolutional Neural Networks have proven to be very effective in solving image classification tasks such as digit recognition, however they have also been shown to be vulnerable to adversarial attacks. Thus in order to create a model that captures the essence of numerical digits better, we use algorithms from topological data analysis to create features which then can be used to train a deep neural network to classify MNIST digits. We show that we can successfully train a model only on topological features to obtain an accuracy of 94 % on the test dataset and that this model can generalize on much more diverse set of inputs and sizes thus demonstrating that it has captured better the essence of an image of a digit.

1 Introduction

Topological data analysis is a field gaining popularity. Deep learning is shown to be successful in many different applications, thus there is interest in the potential of combining the algorithms from both these fields. In this project we tackle a simple problem of recognizing MNIST digits using methods from both these fields.

The input to our algorithm is a grayscale image of a numerical digit and the output is from 0-9 specifying which digit it is. The image is first fed through the topological algorithms which will be described below to produce a set of features which is input to a deep neural network. Since the topological algorithms do not have any restriction on image sizes, any square image or arbitrary size is a valid input, however for training we use the MNIST dataset so we use fixed size images for training.

2 Related work

There has been some work done before, looking at the combination of neural networks and topological data analysis. The paper by Hofer, Christoph Karl et al [1], shows some of the current work in this field. The paper also describes some of the other related work in this area. They describe different methods for a neural network to take as input topological features. The paper by A. Adcock et al [2] talks about the featurization we use in this project. They however do not use neural networks but use other machine learning algorithms.

3 What is Persistence Homology

Persistence homology is a technique which can be used to find the dominant topological features given a point cloud dataset. The input to persistence homology is an abstract simplicial complex which is a generalization of a graph and a particular filtration which can be thought of as a real valued function on the various simplices of the simplicial complex. The output of the algorithm is sequence of pairs (b,d) referring to the birth and death times of a particular topological features. We will be only considering H0 and H1 features which corresponds to components and loops in our simplicial complex.

The above definition uses the terminology of topological data analysis, for those unfamiliar with these terms the following will be a more simpler explanation but does not capture all the aspects precisely. The input to the algorithm consists of a graph of nodes and edges and a filtration function. The filtration function can be thought of as a real valued function on the nodes, thus every node has a real number associated to it. The idea being we will consider sub graphs induced by the nodes of filtration value less than some threshold value. As we sweep this threshold values from 0 to infinity, the graph goes from empty to full. During this process, various topological features are created and destroyed. Here, by topological features we just mean connected components and loops in a graph. Thus we can imagine as we sweep the filtration value, new nodes are added to the graph and initially they appear as separate components. As more nodes are added, when we increase the filtration value, some of these components merge, thus some of them are destroyed. Similarly connecting up some nodes may not merge components but can create loops which are another kind of topological feature. Eventually even the loops may be filled in and at this point the loop is considered destroyed. Thus the output is basically a sequence of pairs (b,d) which correspond to the birth and death of particular features. Thus we have a sequence for components called H0 and a sequence for loops which are called H1.

Thus for example if the pair (2,5) is part of the H0 sequence then a new component was born at filtration value 2 and it was merged with another component at filtration value 5. Similarly if we have the pair (1,Inf) for H1, then we have a loop that was born at filtration value 1 and it was never filled in thus its death is at infinity. The robustness of the algorithm comes from the fact that the really important features have long lifetimes and can be distinguished from the more transient features which are born and die quickly.

There is another variant of the persistence homology algorithm called zig-zag persistence which sweeps a window of filtration value instead of all the values less then a particular value. Everything else is similar but the barcodes now have a different interpretation as some of topological features can now be removed as the window leaves it and no longer covers it.

4 Application of Persistence Homology to MNIST digits

Our goal is to use the algorithms of persistence homology to look at the digits of MNIST in a topological manner. So we take a particular MSNIST image which is 28 by 28 in size and construct a graph of non-zero pixels. By this we mean that for every non-zero pixel in the image we add a node to the graph and there is an edge between nodes corresponding to neighbouring pixels. We include diagonal pixels as neighbours, thus every pixel can have at most 8 neighbours. This graph now represents the topology of the digits, thus 6 and 9 would not be distinguishable, thus to include such orientation information, we use the filtration. Below is an example of the graph for the digit eight. We can choose as the filtration value the height of the pixel in the image. In fact we can

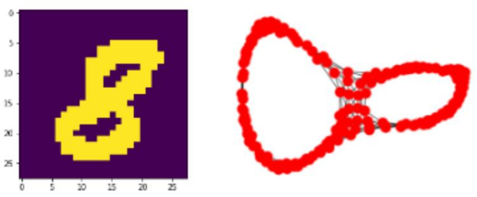


Figure 1: A MNIST digit and the graph implied by its pixels.

choose multiple filtration values, to make sure we capture all the orientation information we need to recognize each digit. Thus we settled on a final total of 8 filtrations corresponding to 8 directions we can sweep an image. The first four are top to bottom. left to right and vice versa. The other four are similar but in the diagonal directions.

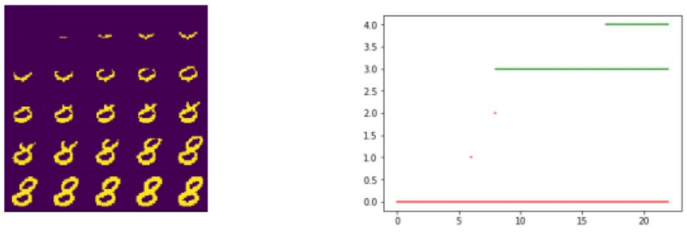


Figure 2: The sequence of images in the filtration and the corresponding barcode

Above, we have an example of an image of 8 being swept from the bottom to the top. Thus for every filtration we can produce a sequence of births and deaths for H0 and H1, we can represent these pairs in what is called a barcode plot. Here we draw a line in red for every pair in H0, the line goes from the birth to the death. Thus the X-axis represents the filtration value. The pairs from H1 are represented in green.

We can now interpret this barcode plot for the digit 8 as follows, since there is overall one connected component, there is one long red bar. The digit 8 has two loops so we have two green bars. Since we are sweeping from the bottom, we close one loop first and close the second loop later, thus one of the green bars is born a little later than the other.

Thus in this way we can create the barcodes for all the eight different directions and this should capture the topological essence of the digit in the image.

5 Featurization

Since we wish to train a neural network on these topological features, we need to convert them to a form that a neural network can process. The main problem here being that the number of pairs of births and death can vary depending on the input, but we need a finite representation to feed to a neural network. We tried the following finite featurization.

$$F(m,n) = \sum_{i=1}^{N_f} (b_i + d_i)^m (d_i - b_i)^n$$

We basically construct a matrix of numbers of size N_f by N_f. The (m,n) element of this matrix is given by the formula above. It is essentially a polynomial function of the births and deaths summed over each of the features, thus giving a finite representation. We repeat this for H0 and H1 and also for each direction. We use N_f = 5, thus we have 25*2 = 50 features for each barcode plot and since we have 8 directions, we have a total of 400 features. Since the powers of the filtration values can be very large we apply the function log(1+x) on each value of the feature. We also subtract the mean and divide by the standard deviation computed over the entire training dataset.

6 Methods

Now that we have a finite size input which is 400 for each image, we can build a simple 9-layer neural network with the following number of hidden units : [400,600,600,100,100,100,100,100,10]. We use cross-entropy loss with a softmax classifier.

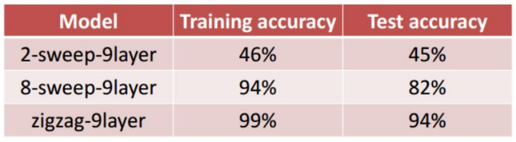
Different architectures were tried but this was the one with the best performance. Training proved to be very hard, as most runs stopped improving after reaching 50% to 60% dev accuracy (training accuracy was similar). To improve the performance various tricks were employed which proved to be very effective in improving performance.

First one of the tricks was to use triangular learning rates schedules. If we use only small learning rates, the loss always gets stuck in bad local minima, however large learning rates also performed poorly, what was found to perform the best was to use small learning rates initially then quickly change to large learning rate then wait until plateau and then reduce the learning rate.

Second trick was to train on the failing cases only. Multiple initializations were analyzed and it was found that the examples they fail on were essentially the same set. Thus we first trained on the failing set only, then after getting reasonable performance we switched to the full dataset. At the end we again alternated between failing set and full set to squeeze out the last bit of performance.

7 Experiments/Results/Discussion

Many different architectures and hyper parameters were tried. The performance was very poor when fewer than 9 layers were used. We initially tried only 2 sweep directions, left to right and top to bottom, but the performance was poor. With persistence homology features, the tricks discussed in the methods section had to be used to get the best performance. However the best performance was obtained when we used the zig zag variation of the persistence homology features. The below table displays some of the representative results for the various different experiments performed.



Model	Training accuracy	Test accuracy
2-sweep-9layer	46%	45%
8-sweep-9layer	94%	82%
zigzag-9layer	99%	94%

Figure 3: Table summarizing the results

We will discuss some of the details of the best model. Zig zag persistance homology features numbering 400 were created for 55 thousand training examples from MNIST. The test set had 5 thousand examples. We used a batch size of 1024. Simple stochastic gradient descent was used as ADAM did not perform very well. The learning rate was initially low at 0.001 , increased to 0.01 for 2000 iterations and decreased back to 0.001. All initializations were not successful. Many of the initializations were stuck in local minima where two digits such as 6 and 8 or 0 and 6 were classified to the same class, and no amount of extra training would move it out of this local minima. Thus such runs were abandoned and restarted with a new initialization. Below is the loss graph and confusion matrix for a successful run with the best performance.

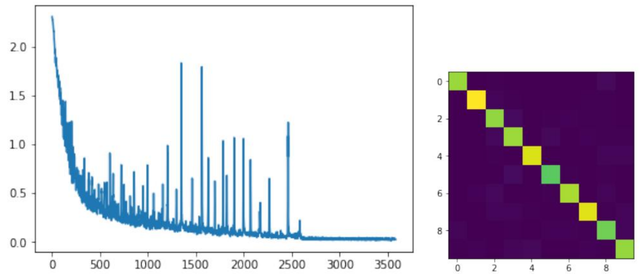


Figure 4: Loss graph and confusion matrix of the best model

Since the goal of project was to obtain a model which purely relied on topological information, the model was tested with some hand created images of digits which a human would classify easily but a convolutional neural network might have trouble with. A small dataset of size around 50, consisting of very non-standard images of digits were created. The performance of the model was 70% on this dataset without any retraining. Below are some of the examples we could classify correctly

As we can see, the model does not have trouble classifying correctly when the line width is varied and other distracting features are added. The model also works on any sized images so no resizing had to be performed. It is to be noted that the model is not perfect, as it did not get 30% of these non-standard digits correctly and more work needs to be done. However for a model that was just trained only on MNIST it is surprising to see the extent of generalization we can achieve without additional training.

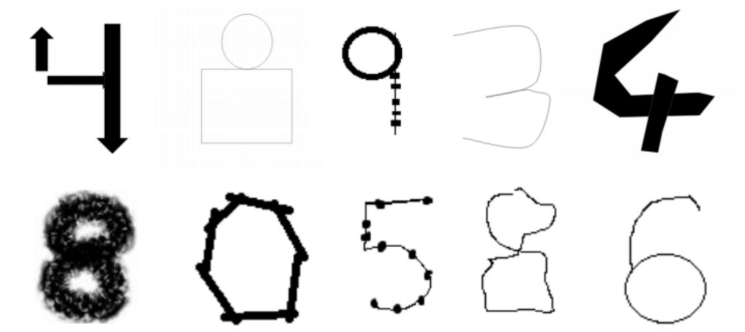


Figure 5: Qualitative results - Some examples that were correctly classified

8 Conclusion

In this project we showed that a neural network can be successfully trained on the topological features featurized in the way described in this project. We also showed that such a model has the capability to generalize on more non-standard digits with varying line thickness and distracting features.

9 Future Work

There are many future directions for this project. Due to restrictions of time, they were not pursued. Some of the obvious ones are to quantify the difference in performance of standard convolutional neural networks and model described in the project on the non-standard digits dataset that was hand created for the purposes of this project. The other direction is to test this model on adversarial examples of the standard CNN model and also investigate what could be the adversarial examples for the topological model. Another direction is to combine both the approaches to make use of the strengths of each of the models to get an overall superior model.

10 Contributions

This project was done with only one team member, however I would like to thank Prof. Gunnar Carlsson for the discussions and suggestions he provided for this project.

References

[1] Hofer, Christoph Karl et al. “Deep Learning with Topological Signatures.” NIPS (2017).

[2] A. Adcock, E. Carlsson, and G. Carlsson. "The ring of algebraic functions on persistence bar codes". CoRR, (2013).

[3] The library GUDHI. http://gudhi.gforge.inria.fr/

[4] The library Dionysus. http://www.mrzv.org/software/dionysus2/

[5] Keras python library.

[6] Tensorflow python library.