Simulating Hide and Seek with Convolutional Neural Networks and Boosting

Yuki Zaninovich, Kyle Harrington

Department of Computer Science, Tufts University / Beth Israel Deaconess Medical Center, Harvard Medical School

INTRODUCTION

Hide and seek between prey and predator is an inherently competitive interaction that increases in difficulty for both parties as time progresses; initially, the prey has no clue of its landscape and the predator doesn't know how to spot prey efficiently, but both become smarter as it learns about its opponent and environment. This allows us to model their behavior with machine learning, a subfield of computer science concerned with classifying datasets through algorithms that learn by being trained, and test the extent of their capabilities in the limit of asymptotically difficult problems (e.g. predator attempting to find pseudo-perfectly camouflaged prey).

The scope of this project is concerned with teaching the predator to detect prey. The algorithm is simply fed a 250 x 250 input image and information on whether a prey resides in it or not. Through a sequence of training steps, the algorithm begins to learn the features of an image that are indicative of a potential hidden prey.

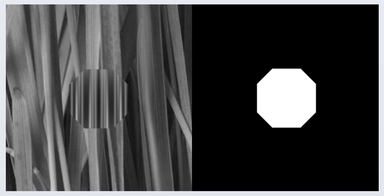

Figure 1: A sample prey image and its corresponding classification image

METHODS

Convolution:

Convolution is a process by which information from multiple sources are combined into one. Within the context of this problem, we "convolve" images by extracting information from a group of neighboring pixels.

We initially begin with an input image where each pixel provides information on the color of that pixel. Then we create an image significantly smaller than the input, called the kernel, and take the dot product of it and a group of neighboring pixels in the input. Repeating this process by "sliding" the kernel over the input image one pixel at a time will yield an output image containing new information.

METHODS

Convolution adds significant value in image recognition problems by producing information on spatially correlated features of the image. This is due to the "pooling" effect generated by performing matrix multiplication on neighboring pixels. For example, a kernel with negative column values on the left and right borders and positive values in the middle columns is essentially a "vertical line detector"; taking a dot product of it and an area of the image that has a vertical line will yield an extreme output value, since the change in vertical pixel gradation will be captured. While this is an example of a rather simple feature that can be extracted via convolution, abstract features of images can be detected with more complex kernels.

Figure 2: Result of convolving with a horizontal line detector (Source: aishack)

Convolutional Neural Network:

A neural network attempts classification with a structure similar to that of our brain. On a high-level, it takes a certain amount of input, and performs a linear combination on those numbers with some set of weights to produce an output value, which is then mapped to a classification. For example, if we are trying to predict whether a given house will sell and you know its price, how old it is, and its location, one could feed each of these values into a pre-trained neural network to get a prediction. Training a neural network is synonymous to optimizing the set of weights, which determines how much of a say each input contributes to the final classification.

A Convolutional Neural Network is a machine learning algorithm that combines the above two techniques. The value in each pixel of the kernel are the weights, and we attempt to optimize them by observing how useful the output image resulting from a convolution with it and the input image is.

METHODS

One way to find the optimal set of weights is through random perturbation. Starting with a kernel fitted with a random distribution, we alter each pixel value by a random, small amount and convolve the input using it. If it reports better accuracy than all past kernels did, it is labeled as the "best kernel" so far. Otherwise, the kernel is disposed of and a slightly modified version of the "best kernel" is tested in the next iteration of the kernel optimization process.

Another method is the Delta-Rule, defined as $\Delta w_j = \eta(t-o)i_j$ where Δw_j is the change in weight j, η is a constant called the learning rate, t is target output, o is actual output, and i_j is the j^{th} input value.

Both weight learning processes are repeated until the change in all weights becomes negligible, and hence convergence is observed.

Adaptive Boosting:

In the world of machine learning, there is always a trade off between time and accuracy. Even the simplest algorithms can yield high accuracy after enough training iterations, but can take a long time to reach that level. Conversely, sophisticated algorithms rarely converge instantaneously.

Boosting attempts to overcome this by joining a cohort of fast, weak classifiers together to create a single strong classifier. Adaptive Boosting (AdaBoost), a type of Boosting algorithm, weighs the results of each weak classifier based on accuracy, and iteratively adds them to form one single output. The power behind AdaBoost stems from its ability to specialize on data points that were misclassified in previous Boosting iterations. This is accomplished by assigning equal weights to each instance initially, and increasing or decreasing them based on the accuracy of its classification by the current Boosting iteration.

Boosting vs. Depth:

The traditional approach to image recognition has been to implement a deep CNN, which are CNN's with multiple layers of neurons, meaning convolutions are compounded such that higher order feature patterns can be detected. This project primarily focuses on the potential upside to integrating Boosting instead.

METHODS

The immediate allure of Boosting is that it guarantees convergence after a certain number of iterations; in the worst case each Boosting iteration specializes on one distinct instance, a capability other algorithms don't necessarily have. That said, Boosting that many times is computationally expensive when we are dealing with decently large images, and the point to investigate is whether a decent accuracy can be attained far earlier.

Another advantage is the expedited training speed. As deeper layers are traversed, they receive less "error signals" and therefore require more training time. Furthermore, adding more layers to a deep CNN requires re-training all previous layers, while incorporating an additional Boosting iteration solely involves tuning that one kernel.

RESULTS

Currently, our classifier has shown to attain 57% accuracy when performing 500 kernel optimization steps and 20 Boosting iterations. It appears as though random perturbation is not reliable enough to produce an optimal kernel in a computationally feasible amount of time. We anticipate the replacement of it with Delta Rule to converge significantly faster, as it incorporates more information when learning.

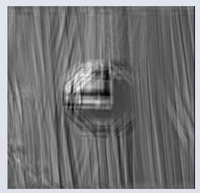

Figure 3: Output of current algorithm on prey image

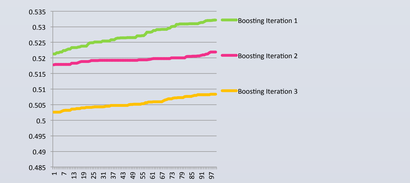

Figure 4: Change in error over kernel optimization steps on different Boosting phases