Extractor documentation

version 1.0, updated 1-14-15

Contents of this document

Extractor has been written and tested with GNU Octave 3.6.4, compatibility with other versions of the Octave environment is not guaranteed.

This project is an active work in progress and the documentation, functionality and source code included are subject to change. The project source code for iterations prior to the GitHub repository 1-14-15 commit may be obtained upon request.

I. Introduction

A. Motivation

The initial purpose of this project was experimentation and exploration of feature extraction and encoding methods to approximate those of Artificial Neural Networks trained with the traditional Backpropagation of error gradients. This iteration focuses on generating a single layer of a Convolutional Neural Network and is based on the success of previous iterations of the same project that extracted features for standard Neural Network layers. Since the primary focus of this project is that of experimentation, it may include disabled features but the project should now be functional and relatively stable.

B. Style and design

This project is being developed iteratively with readability and transparency as a primary concern. More efficient functionality is certainly possible, however attempts have been made to reduce the occurrence of blocks of code with nebulous functionality or purpose. A code style similar to that commonly used in Python (https://www.python.org/dev/peps/pep-0008/) has been adapted where applicable.

Each of the project's functions is located in its own file of the same name and prepended with "Extractor_" denoting that it belongs to this project. Supporting functions developed external to this project do not follow this style. The top of each project file (with the exception of extractor_params.m) contains a comment block with a short description of its purpose, its arguments, and its return values. For this implementation, structure objects were used in place of the more cumbersome Octave classes in order to keep the project as simple as possible.

C. Environment

GNU Octave was initially selected as the project development platform for its simplicity, ease of use, central focus on Matrix manipulation and the author's familiarity with the environment. Many of these properties lend themselves well to the iterative development of a simple experimental application of this type. However, this project seems to have outgrown this environment and now needs to be ported to a more object friendly language. The next major release is planned to be developed in Python using the Numpy and TensorFlow libraries.

D. Planned updates

The following lists desired updates in order of priority:

- Add functionality to better indicate the state and properties of each solution
- Add reduce_fraction parameter with functionality to alter the rate of reduction in extractor_reduce.m
- Add solution_precision_n parameter with functionality to combine several candidate solutions when arriving at the final solution
- Implement in Python utilizing TensorFlow and Numpy libraries

II. Methods

A. Project goals

Understanding the principles behind the abilities of a given Artificial Neural Network [ANN] is not a simple endeavor. Much has been written concerning the interesting properties of ANNs [1] and about useful configurations of said networks [2,3], and few other methods of training are as popular or successful as Back Propagation for generating ANNs used for general classification tasks. This success is evident based on the methods used to train the top models for common vision classification benchmarks [4].

The goal of this project is to hopefully reduce the complexity of a ANN enough so that it may be approached and studied at a more fundamental (and digestible) level. The approach of this particular project is to break this process into the following steps: 1. Extract features, 2. Estimate the utility of each feature, and then, 3. Use these features to generate or encode an ANN layer with the hope that the chosen encoding scheme produces a layer with a capacity approaching that of one trained with Backpropagation.

B. Algorithm

The method employed in this project is as follows: To simplify the network architecture as much as possible the training data is split into two separate classes, these classes are denoted the "null" and "target" classes, where the null data set includes instances from all classes with the exception of the target class. Therefore, the network we are attempting to encode is a binomial classifier.

A random subset of the training data is sampled and the remaining instances not sampled are then included in the validation set. Using this sampled data, we extract and aggregate candidate features and then use these features to build a solution (ANN layer) local to the sample space. The solution is evaluated against the local sample data and then additional features are added if all instances are

not considered "separable" or solved. The process of adding features to the solution is iterative, where the most "difficult to separate" instances have the most influence over the next round of feature extraction. This process used generate "candidate" solutions is repeated sample_n times. Finally, most general and informative solution is selected from the candidate solutions based on a predefined "fitness" metric.

This method is analogous to a focused or specialized form of evolutionary search. For each step, a feature is selected from many candidate features based on its calculated fitness. For this implementation, a feature's fitness is defined as the product of its scaled activation ratio and the scaled sum of its correct activations. Once a Feature has been selected, it is added to the pool of candidate feature maps. This pool of feature maps is used to iteratively generate a solution. Up to max_fmap_n feature maps may be selected and used to encode a Convolutional layer. In this implementation, encoding a binomial classification layer is envisioned as a form of the Bin packing problem [5] and a simple and crude approach is explained in more detail below.

The final round of selection takes place among the collection of candidate solutions generated. In summation, the algorithm uses two primary selective processes to search for a Convolutional Neural Network layer solution: feature template selection and solution selection.

The goal of this approach is to use stochastic batch processing to (hopefully) sufficiently explore a small subset of the problem space, collecting and combining features that have promising characteristics. The assumption that the collected pool of features [feature maps] is sufficient to describe the problem space is made based on the chosen constraint parameters. Under this assumption and utilizing these feature maps, a layer of nodes is generated in a manner that aims to encode a variety of descriptive features sufficiently retaining the informative characteristics of the input layer while reducing the variation "entanglement" between the two label classes.

C. Citations

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, “arXiv.org” [1312.6199] Intriguing properties of neural networks, 2014.

[2] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” jmlr.csail.mit.edu, 2011.

[3] Y. Bengio, “Learning Deep Architectures for AI,” http://www.iro.umontreal.ca, 2009.

[4] R. Benenson, “What is the class of this image ?,” Classification datasets results, 2013. http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[5] R. Korf, “A New Algorithm for Optimal Bin Packing,” http://aaaipress.org/, 2002

III. Concepts

Alpha mask

Examples: alpha_mask, null_mask, target_mask
The matrix of learning influence values associated with each training or sample instance. These values are used to scale the gradient component for the respective training instances, allowing for dynamic control of instance influence as well as enabling the use of training sets with a wide disparity in number of instances included per class.

See extractor_xcost.

Activations

Examples: actv, or actvs
A matrix of node output values computed from a given matrix of input instances.

See extractor_fprop.

Templates

Examples: template_node, template_tile, template_instance,...
A template is an extracted or calculated pattern of various forms (node, tile, instance, etc.) used to generate, communicate and transform feature information.

See extractor_template

Candidates

Examples: candidate_templates, candidate_tiles
Term used to convey the idea that a group of templates or tiles are subject to a process of selection, with the most fit candidates passing their features or other useful components on to be further evaluated.

See extractor_extract, extractor_template

Tiles

Examples: template_tile, feature_tile, etc.
A tile is a tile_dim(1) x tile_dim(2) set of features extracted from a given feature "window" of an instance. The window has a receptive field size defined in the parameter rfield_dim. Tiles break the problem down into a simpler form, greatly reducing the number of feature elements per instance at the cost of greatly increasing the total number of example instances.

See extractor_params and extractor_decompose

Target and Null classes

Examples: target_instance, target_tile, etc.
The goal of the generated network layer is to maximize the activation of target class instances while simultaneously minimizing activation of null instances. The class of each instance is determined by the target_label parameter, where all class labels not equal to the target_label are considered to be in the "null" class.

See extractor_main

Feature maps

Examples: fmap, fmap_collection, etc.
A feature map is a collection of nodes where each node's weights are restricted to a given "receptive field" or "window" of the feature space. These receptive fields are then tiled to cover all (or most) input features. The dimensions of a nodes receptive field, the extraction pattern, and the number of tiles extracted to form a single feature map are defined in the parameters file.

See extractor_params

Collections and pools

Examples: fmap_collection
Collections and pools are simple structures used to aggregate instances, input data, features, etc.

See extractor_main, extractor_sample, extractor_extract, etc.

Samples

Examples: sample_data, sample_instance
In the context of this project a sample is a subset of a given data set.

See extractor_main

Solutions

Examples: solution, solutions
This project generates solutions in the form of a matrix of weight values representing a Convolutional Neural Network layer

See extractor_main, extractor_sample, extractor_encode

Entanglement

For the purposes of this project, Entanglement refers to the fact that, when considered alone, the individual features (pixel values) of the input data do not provide sufficient statistical information to accurately identify the instance's class (this would be a much easier problem if they did).

IV. Data structures

solutions, solution
A solution takes the form of a matrix containing a collection of feature map weight values:

solution = [fmap_1, fmap_2, ..., fmap_n]

The solutions collection is a cell structure containing a list of individual solutions:
solutions = {solution_1, solution_2, ..., solution_n}

fmap, fmap_collection
A feature map or fmap is a structure containing a n x m weights matrix (where a nodes weights are stored column-wise fashion) and a fitness value:

fmap.weights = [w_11; w_12; ...; w_1n, w_m1; w_m2; ...; w_mn]
fmap.fitness = fitness_value

template_nodes, template_pool
A template node is a structure containing the node's weight matrix and fitness value:

template_node.weights = [w_1; w_2; ...; w_n]
template_node.fitness = fitness_value

A template pool is a collection of nodes that also includes a n x m matrix of template weights (where each template is a row-vector) and a corresponding vector of fitness values:

template_pool.weights = [w_11; w_12; ...; w_1n, w_m1; w_m2; ...; w_mn]
template_pool.fitness = [fitness_1, fitness_2, ..., fitness_n]

V. Project overview

A. Operation flow

Please see the documentation/extractor_flow.pdf or Extractor_flow.png

B. Functions

Supporting functions:
The following supporting functions are written by 3rd parties.

fmincg.m
Copyright (C) 2001 and 2002 by Carl Edward Rasmussen. Date 2002-02-13

displayMNISTSample.m
Source by CAD
http://www.cad.zju.edu.cn/home/dengcai/Data/MNIST/images.html
Updated 12/13/15 by kotulc

loadMNISTLabels.m
Stanford.edu http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset

loadMNISTImages.m
Stanford.edu http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset

Extractor functions:
Note: This section may need to be updated in light of recent project updates

All functions listed below have "extractor_" prepended to their names. Please see the function source file for argument and return value specifications.

main
The driver function for the convolutional neural network feature extractor. PARAMS is a global variable is used to simplify function calls. See extractor_params.m file for details concerning the definition of this constants struct.

sample
Operate in the scope of sample_data instances. returns a local encoding (solution) and the collection of extracted feature maps for the target label class. Note: function may be expanded for generating null feature maps

encode
Generate a encoding/solution by assembling [packing] available fmaps into a layer based on generalization (e_sum given shape of instances meeting actv. threshold). Return the solution and sample_data with the updated alpha_mask

update
Update the fitness and all relevant data for each feature map contained in the null_fmaps and target_fmaps components of fmap_collection

evaluate
Generate two output layers, one trained directly on the values of the sample data, the other trained on the activations of the solution. Display metrics appropriate to evaluate the accuracy of each.

fitness
Calculate the fitness metric which is the product of sigmoid(r) and log(e_sum) where r is the offset and scaled ratio of a nodes excitatory activations to it's inhibitory activations. This metric seeks to find a balance between a nodes descriptive and generalization capacity. Add fitness and relevant descriptive factors to the node and return

extract
Extract and return a feature template extracted from the sample instances in sample_data. The returned feature template is considered the most "fit" of all generated candidate templates.

template
Decompose template_instance and eval_data into receptive field tiles and generate a single node activating the template tile against all null tiles. Return the node and all biased null tiles

reduce
Reduce the size of the set of nodes by half, compressing the representation of the excitatory instances in target_data trained against a collection of inhibitory instances null_data. Return the new reduced set of nodes along with a list of node fitness values corresponding to each node in the set
Add parameter for rate of reduction, default to 0.5

nodes
Generate a set of nodes with the goal of exciting instances contained in target_data and inhibiting those in null_data. Use cross-entropy optimization

subset
Return structure subset_data, a random subset of subset_n rows from data_set. diff_data is the set difference (data_set\subset_data), the values remaining in data after removing instances included in subset_data.

decompose
For each instance (column-vector) of the matrix 'instances' reshape the instance into its original dimensions, feature_dim(1) x feature_dim(2). Decompose each reshaped instance into tiles with a window dimension of receptive_dim(1) x receptive_dim(2). Return the matrix of tiles and corresponding tile mask.

fmap
Return all weights generated from shifting the position of the given template weights according to the set and tile dimensions. The weight bias template(1) is always the first element.

fprop
Given the network weight cell w and feature matrix x (including bias x), propagate the signal through the network and return the x of the last layer as the matrix 'x'

mask
Return a binary mask where active elements (1) represent members of the tile of dimension tile_dim extracted from a flattened matrix with original dimensions feature_dim(1) x feature_dim(2)

normalize
Normalize the row-vectors of x to be in the range [0,1]. The center parameter is a bool value that indicates if the returned values should be centered around zero, i.e. a range=[-1,1] instead of [0,1]. The scale parameter is a bool value indicating if the resulting row-vectors should be scaled so that each has a sum of 1.

sigmoid
Given the scalar, vector, or matrix z, compute the sigmoid for each element. Returns sigmoid vector, scalar, or matrix g

tindex
Return a matrix of indices indicating the starting and stopping positions (respectively) of a tile with receptive_dim dimensions. tile_indices matrix is size n x 2, where n is the number of tiles within a feature_dim dimensional matrix

xcost
Calculate the cost of the parameter values theta with the input values X and target activation values Y, with each instance weighted by the vector of alpha values.

xopt
Optimize the given matrix of weight values, minimizing the cost using the fmincg function.