Machine Learning Engineer Nanodegree

Capstone Proposal

David Howard October 9, 2017

Proposal

Domain Background

SLAM (simultaneous location and mapping) is a well known technique for using sensor data to create a map of a geographic region and also keep track of the sensors location within the constructed map. Typically this is done with either a particle filter or a Kalman filter. One application of SLAM operates on an unknown region and builds up knowledge about the region via sensor measurements. Another application uses existing knowledge, such as maps, along with sensors, and is aimed at determining the sensor's location within the known region. 1 (https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping)

In reading about SLAM, I found it interesting that I had previously been trainedto be a human SLAM algorithm when I was a helicopter pilot in the U.S. Army. In the 1970's and 80's, (pre-GPS) se trained to navigate high speed low level light by using a topographic map and identifying our location and path where the sensors were our aircraft compass, our eyes observing the terrain elevation features and comparing to the topo map. There was a name for this, Nap-Of-the-Earth flight. 2 (https://en.wikipedia.org/wiki/Nap-of-the-earth)

One source of potential data for use in a SLAM system is a digital elevation model, generically referred to as DEM. A great source of such data was created by the NASA Shuttle Radar Topography Mission (SRTM) . In 2000, the Space Shuttle was used to create a digital elevation map of the entire world (minus some polar regions). The NASA SRTM data is publicly available for download, in various height resolutions. SRTM Mission ([https://www2.jpl.nasa.gov/srtm/mission.htm)

Problem Statement

This project will attempt to create a simple SLAM type implementation using a convolutional neural network to determine position within a known region. The train/test data used for learning will be SRTM elevation data treated as an image. Input for evaluation will be images composed of pieces of the same data with possible modifications such as distortion, alignment and reduced resolution. Output will be a predicted position of the input images. The metric will be accuracy of the predictions over a test data set.

I did not research whether or not this has already been attempted, but I assume it has been studied. I purposely did not look for any existing research like this keep my approach independent.

Datasets and Inputs

SRTM data is available online ([https://dds.cr.usgs.gov/srtm) . SRTM data is available in multiple resolutions. The data is segmented into 1 latitude/longitude degree squares. The highest publicly available resolution is 1 arc-second per elevation posting (~30 meters at the equator), which results in a 3600x3600 matrix of elevations. The

data files are actually 3601x3601 in order to fill overlaps if multiple cells are composed together. Other resolutions are 3 arc seconds (~90 meters) and 30 arc seconds. Detailed description of the data (https://dds.cr.usgs.gov/srtm/version2_1/Documentation/SRTM_Topo.pdf).

The SRTM data is stored as .hgt files. The format is binary 16 bit unsigned integer, representing integer meters in the range 0..32767, with the value 32768 indicating 'no data' at that spot. Level 1 files are organized as 1201x1201 height postings. Level 2 files are 3601x3601 height postings. The first posting is the lower left corner of the 1 degree cell. There is a trailing row and column in each (the '01') that is there to provide overlap to the next cells.

The data will be preprocessed as follows:

```
 - read the binary .hgt to be processed (16 bit unsigned integer)
 - reshape from 1D to 2D of either 1201x1201 or 3601x3601
 - trim off overlap row and column, reshaping to 1200x1200 or 3600x3600
 - fill voids (voids have value 32768)
 - normalize from 16 bit unsigned 0..32767 to float (0.0 .. 1.0]
 - since different files have different ranges of elevation, the normalization will
 expand the range
 from its min/max elevation to the full 0.0 .. 1.0 range. This is intended to creat
e more definition
 in areas of flat terrain
 - subdivide the 1 degree cell into squares with an integer label. each subdivision
 will be an input 'image'
 - the size of the subdivisions will be determined during development
```

Example subdivision with labels

```
+----------------------------------+
|   0    |   1    |   2    |   3    |
+----------------------------------+
|   4    |   5    |   6    |   7    |
+----------------------------------+
|   8    |   9    |  10    |  11    |
+----------------------------------+
|  12    |  13    |  14    |  15    |
+----------------------------------+
```

The integer label can be directly converted to an actual latitude/longitude location relative to the cell location.

Solution Statement

The solution approach will be to use a convolutional neural network to identify the location of pieces of elevation data within a 1 degree SRTM data cell using CNN image recognition techniques. The solution will look something like the ML Nanodegree dog image project.

Specifically, the input data will be a 1 degree data cell, divided into individual squares of some smaller size that will be treated as input images. The input images will be labeled with indices representing the location within the 1 degree cell. After training, predictions will be made using similar small squares of elevation data. Initially this test data will be identical to some of the training inputs. This is to test whether or not the model can predict

anything at all. Since the test data will be a subset of the training data, if the model is constructed correctly it should have a high prediction metric. Once a model is working at that level, training and test will be attempted with various distortions so that training and test data will be different enough to evaluate the method. The will be experimentation with augmentation of the training/test data to see if results can be improved.

Arriving at a solution, if it works at all, will involve experimentation with the organization, size and resolution of the training/test data. Ideally the training data will be a full 1-arc second, 3600x3600 cell subdivided into individual labeled squares where the label will indicate the position of the square.

Benchmark Model

In this case, lacking an independent model to compare against, the benchmark will be a naive model, possibly a 2 layer feed forward architecture. (as suggested by the reviewer).

Evaluation Metrics

The outputs of the model will be predictions of the probabilities of a test image location matching one of the actual locations using 'softmax'. The outputs will be a set of probabilities (like the dog project). The evaluation metric will be log-loss over the range of true labels vs the predicted probabilities.

Project Design

Benchmark Step

- input a 1 degree of elevation data
- preprocess it into a numpy array of floating point values [0..1.0]
- divide the array into a set of N rectangles of the input data of identical size
- using Keras, construct a naive feedforward model with inputs of the input rectangle size and outputs representing probabilities of location over the 1 degree area
- train the model using all the input data
- test against a subset of the input data
- iterate until predictions have a high probability of success

Evaluation Step

- input a 1 degree of elevation data
- preprocess it into a numpy array of floating point values [0..1.0]
- divide the array into a set of N rectangles of the input data of identical size
- using Keras, construct a CNN with inputs of the input rectangle size and outputs representing probabilities of location over the 1 degree area
- train the model using all the input data
- create a test data set using a subset of the input data with distortions (rotations, overlaps, sizes, reduced resolution)
- iterate and evaluate accuracy

The iterations of the evaluation step will include training and testing of input data with and without image preprocessing (ImageDataGenerator), and variations on distortions of the test data set.

Development and training will be performed using Python 3, Keras with Tensorflow-gpu backend on Linux with an Intel I7 processor and NVidia 1050GTX graphics card.