Computational Power Analysis

David Laredo*

Abstract

The following report presents an analysis of the asympotic complexity of the proposed algorithms to be used for the development of the inteligent HVAC system as well as a brief analysis on the computational power of the computer we have destined for our experiments. Based on these two analysis we will draw our conclusions on whether the available computational power is enough for the development of our project.

1. Computational Complexities of the Algorithms

In this chapter we will briefly discuss each one of the algorithms intended to be used in the development of the Intelligent HVAC System. Along with the description of such algorithms we will present a very brief analysis on their computational complexity.

1.1. Artificial Neural Network

Artificial Neural Networks (ANNs) are computing systems inspired by the biological networks that constitute the brain. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Further, they may have a threshold such that only if the aggregate signal is below (or above) that level is the downstream signal sent. Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times. [1]

The multilayer perceptron is an artificial neural network structure and is a nonparametric estimator that can be used for classification and regression. It is a supervised learning algorithm

*dlaredorazo@usmerced.edu

that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 1 shows a one hidden layer MLP with scalar output. [2]

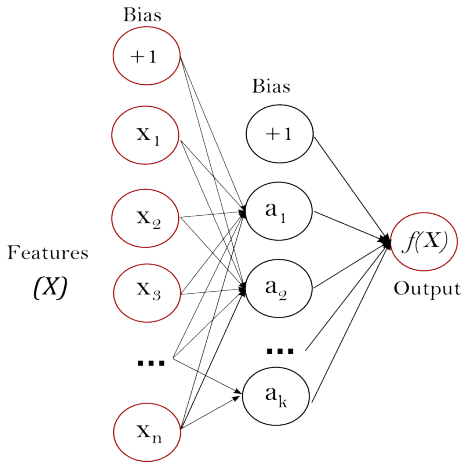

Figure 1: Multilayer perceptron

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i | x_1, x_2, ..., x_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$ - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

The advantages of multilayer perceptrons are:

- Capability to learn non-linear models
- Capability to learn models in real-time (on-line learning)

The disadvantages of multilayer perceptron (MLP) include:

- MLP with hidden layers have non-convex cost function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- MLP requires the tuning number of of hyperparameters such as the *number of hidden neurons, layers and iterations*.
- MLP is sensitive to feature scaling

1.1.1 Computational Complexity

Suppose there are n training samples, m features, k hidden layers, each containing h neurons - for simplicity, and o output neurons. The time complexity of the feedforward-backpropagation algorithm is

$$O(n \cdot m \cdot h^k \cdot o) \tag{1}$$

Also for the sake of simplicity lets assume that $m > h > o$, hence the time complexity for feedforward-backpropgation algorithm is

$$O(n \cdot m^{k+2}) \tag{2}$$

Additional to the computation of the gradients using the backpropagation algorithm we need to consider the complexity (convergence ratio) of the algorithm used for solving the minimization problem inherent to the neural network, recall that at each iteration a minimization problem is solved. For the best case, this is superlinear (L-BFGS) and for the worst case it is linear (Stochastic Gradient Descent).

1.2. Anomaly Detection

Assume we have a sample $\mathbb{M}$ drawn from the same distribution. An outlier, novelty, or anomaly is an instance that is very much different from other instances in the sample. An outlier may indicate an abnormal behavior of the system; for example, in a dataset of credit card transactions, it may indicate fraud; in an image, outliers may indicate anomalies, for example, tumors. [3]

Outlier detection is not generally cast as a supervised, two-class clas- sification problem of seperating typical instances and outliers, because generally there are very few instances that can be labeled as outliers and they do not fit a consistent pattern that can be easily captured by a two- class classifier. Instead, it is the typical instances that are modeled; this is sometimes called one-class classification. Once we model the typical instances, any instance that does not fit the model (and this may occur in many different ways) is an anomaly.

Outlier detection basically implies spotting what does not normally happen; that is, it is density estimation followed by checking for in- stances with too small probability under the estimated density. As usual, the fitted model can be parametric, semiparametric, or nonparametric. In the parametric case, for example, we can fit a Gaussian to the whole data and any instance having a low probability, or equally, with high Mahalanobis distance to the mean, is a candidate for being an outlier. [4]

Figure 2 shows how novelty detection can separate data into regular observations and abnormal observations.

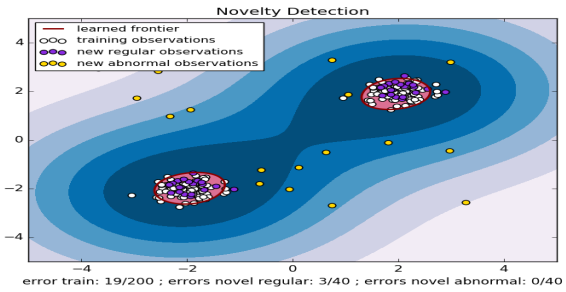

Figure 2: Novelty detection

One common way of performing outlier detection is to assume that the regular data come from a known distribution (e.g. data are Gaussian distributed). From this assumption, we generally try to define the âĂIJshapeâĂİ of the data, and can define outlying observations as observations which stand far enough from the fit shape. The process for defining the "shape" of the data is as follows and assumes that all of the samples come from a known distribution, say Gaussian for the purposes of this explanation.

Assume a training set $T = \{x^{(1)}, \cdots, x^(m)\}$ where $x \in \mathbb{R}^n$. Suppose that each feature $x_i from x$ is distributed according to some Gaussian distribution $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. Hence $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$. The task is now to find the corresponding means μ_i for $i = 1, \cdots, n$ and standard deviations σ_i^2 for $i = 1, \cdots, n$ of the n different distributions. This can be easily done for the Gaussian distribution as follows

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \tag{3}$$

and

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2 \tag{4}$$

We can now describe the simplest anomaly detection algorithm

1. Choose features x_i that might be indicative/sensitive of/to anomalies.
2. Fit parameters $\mu_1, \cdots, \mu_n$ and $\sigma_1^2, \cdots, \sigma_n^2$ using equations (3) and (4) respectively.
3. Given a new example x, compute $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$
4. Tag the new sample x as an anomaly if $p(x) < \epsilon$, where *epsilon* is a hyperparameter defined by the user.

1.2.1 Computational Complexity

Usually the complexity of this algorithm is negligible compared to the complexity of the feedforward-backpropagation algorithm. As can be seen from the Algorithm in Section 1.2 the asymptotic complexity of such algorithm is $O(n)$ where n is the number of choosen features.

2. Computational Power of the Equipment

In this chapter we present an analysis on the features of the available equipment for our experiments and final implementation of the system. We will make emphasis on the number of operations per second (FLOPS) that our equipment can handle.

Floating Point Operations per Second (FLOPS) is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations. For such cases it is a more accurate measure than measuring instructions per second [5]. The FLOPS value can be used to determine the power for both the CPU and GPU.

The amount of Giga FLOPS (GFLOPS) for each node/computer can be calculated by using the following equation [6].

$$GFLOPS = CPU\ speed\ in\ GHz \times number\ of\ CPU\ cores \times CPU\ instruction\ per\ cycle \times number\ of\ CPUs\ per\ node \tag{5}$$

For the case of Graphics Processing Units (GPUs), the amount of flops the GPU can support can be computed by [7]

$$FLOPS = 2 \times number\ of\ parallel\ GPU\ processing\ cores \times peak\ clock\ speed\ in\ MHz \tag{6}$$

The computer used for our experiments has an Intel Core i7-7700K Processor; with a CPU speed of 4.2 GHz average and 4.5 GHz overclocked, 4 cores [8], and 16 IPC [9]. Using these values and eq. (5), the FLOPS range for the CPU is: 268.8 GFLOPS to 288 GFLOPS.

The computer used for our experiments also comes with a dedicated GPU. The GPU used in our system is a NVDIA GeForce GTX 1080Ti with 11GB GDDR5X; with a boost clock of 1582 MHz and 3584 NVIDIA CUDA Cores [10]. Using the given values and eq. (6) the FLOPS value for the GPU is 11339.776 GFLOPS which is approximately 11,300 GFLOPS.

By combining the FLOPs value for both the CPU and GPU, the total FLOPs range of the server is 11608.576 GFLOPS to 11627.776 GFLOPS. This can be converted to FLOPS/month by multiplying the given range by 2592000, using the assumption that a month is 30 days. This gives a final FLOPS/month of 3.00894×10^{19} FLOPS/month to 3.01392×10^{19} FLOPS/month.

3. Our problem

Part of our problem is the detection and classification of faults in the HVAC system of the SE2 building in the Merced campus of the University of California. Such system has an approximate of 2000 sensor and we take samples each 5 minutes. It is our intention to use data from the past 2 years to train our model, this gives us a size for the training set of $n = 210240$ training samples. Thus, lets assume that our ANN will have the following topology: $m = 2000$ for the 2000 input features (this is just a rough estimate since we will apply dimensionality reduction and feature extraction techniques to reduce this number as much as we can), $k = 4$ layers for an ANN of 1 input layer, 2 hidden layers and 1 output layer. In this case we assume that the number of output layers (fault classes) o is much smaller than the number of input layers, we expect a number on the order of 100 different types of faults, as for the number of neurons on each hidden layer we will set it arbitrary to 2000 (the number of input neurons) as it has been shown that the number of hidden layers for other classification/regression problems is usually smaller than the number of input layers, thus, setting the number of hidden layers equal to the number of input layers will help us set a boundary on the number of computations.

Using the above information and eq. (1) the asymptotic complexity of our ANN will be $O(210240 * 2000 * 2000^2 * 100) = O(1.6819 \times 10^{17})$ FLOPS, this neglecting the computational time spent on the minimization problem inherent to the computation of the weights for the ANN.

4. Conclusions

Based on the estimate of FLOPS our equipment can handle which within the order of 3×10^{19} FLOPS/month and the complexity for the training of our modeled ANN we estimate our which is of the order of $O(1.6819 \times 10^{17}$ FLOPS we conclude our system can do the training for our model in approximately one month of computations, provided that we take the most of our equipment, such as the pipe-lining and parallel computing.

References

[1] Artificial neural network -wikipedia. `https://en.wikipedia.org/wiki/Artificial_neural_network`. Accessed: 2017.

[2] Artificial neural network -scikit learn. `http://scikit-learn.org/stable/modules/neural_networks_supervised.html#`. Accessed: 2017.

[3] E. Alpayding. *Introduction to Machine Learning*. Dover, 2014.

[4] Anomaly detection -scikit learn. `http://scikit-learn.org/stable/modules/outlier_detection.html`. Accessed: 2017.

[5] Flops definition. `https://en.wikipedia.org/wiki/FLOPS`. Accessed: 2017.

[6] How to calculate peak theoretical performance of a cpu-based hpc system. `http://www.novatte.com/our-blog/`

197-how-to-calculate-peak-theoretical-performance-of-a-cpu-based-hpc-system. Accessed: 2017.

[7] Console gpu power compared: Ranking systems by flops. https://www.gamespot.com/gallery/console-gpu-power-compared-ranking-systems-by-flop/2900-1334/. Accessed: 2017.

[8] Intel cor i7-7700k processor, technical specifications. https://ark.intel.com/products/97129/Intel-Core-i7-7700K-Processor-8M-Cache-up-to-4_50-GHz. Accessed: 2017.

[9] How to determine the amount of flops my computer is capable of. https://scicomp.stackexchange.com/questions/11306/how-to-determine-the-amount-of-flops-my-computer-is-capable-of. Accessed: 2017.

[10] Nvidia geforce gtx 1080-ti, technical specifications. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/. Accessed: 2017.