COSMIC Sizing of Machine Learning Image Classifier Software Using Neural Networks

Arlan Lesterhuis [1] and Alain Abran [2]

[1] COSMIC Measurement Practices Committee
[2] École de Technologie Supérieure – ETS, University of Québec, Canada
lesterhuisa@kpnplanet.nl
alain.abran@etsmtl.ca

Abstract. Development of machine learning software has now penetrated a large diversity of domains, in both academia and industry. From the initial realm of research with a focus on innovation and creativity, its scaling up in industry requires improved planning, monitoring and control of the development and implementation process. Such industry planning and monitoring is difficult without relevant measurement techniques adapted to the problem at hand. This paper illustrates how generic software functions can be extracted from machine learning (ML) system requirements and their functional size measured in COSMIC function points - ISO 19761. An application of these concepts is presented using an example of an ML image classifier software with a feedforward neural network.

Keywords: Machine learning, neural networks, COSMIC, function points, ISO 19761

1 Introduction

The development of machine learning (ML) software has now penetrated a large diversity of domains both in academia and industry. From the initial realm of research with a focus on innovation and creativity, the scaling up of ML software in industry requires improved planning, monitoring and control of its development and implementation process. Such industry planning and monitoring is difficult without relevant measurement tools adapted to the problem at hand. While there is in the ML literature a large body of knowledge on the mathematical aspects of the variety of ML algorithms and analysis of their performance with various datasets, once implemented in software applications, there is very little in the literature about the software itself that needs to be developed in order to implement the ML algorithms in specific industry contexts.

Generally, the literature describes the 'system viewpoint' of ML, bundling together all the tasks carried on by ML researchers, including the design of the ML system, its coding, operation and data analysis. Considering that ML expertise is very specialized and requires considerable expertise, while being in very high demand as well as in short supply, it would be valuable to segregate the ML specific tasks from the software development tasks for which expertise is more widely available and less specialized.

These coding tasks could then be delegated to staff with programming expertise, thereby freeing up the ML software developers and data analysts, thereby allowing them more freedom to address additional ML challenges. Of course, the prerequisite for such delegation is that software tasks, such as specification of the functions allocated to software be untangled from ML specific analytical tasks and well described prior to delegation. This means that generic software functions be segregated from ML specific functionalities. Success in this endeavor may allow parallelism of tasks in ML projects, with the possibility of shortening their development cycle.

While it is expected that in every software there will be unique aspects making each software distinct from any other, there are as well functionalities common and generic throughout. This genericity is at the basis of software functional sizing measurement methods such as COSMIC Function Points – ISO 19761. In this paper, the functional principles underlying this COSMIC measurement method are used to address two objectives:

- segregation of the generic (classical) functionality to be allocated to software, and not specific to ML;
- use of the international standard to size the generic functionality identified.

Success in both will facilitate delegating tasks to data analysts, as well collecting data in a standardized fashion in order to develop estimation models for planning purpose, and for on-going monitoring of the software tasks within an ML development project.

Section 2 presents overviews of machine learning and COSMIC function points. Section 3 presents the system view of the ML image classifier case study used in this paper, their generic functions allocated to software, followed by their measurement with COSMIC function points. Section 4 presents a summary and suggestions for future work.

2 Related work

2.1 Machine learning and neural networks

A neural network is an ML application that can 'learn' to classify input data with the help of training examples of that input data. The example selected here from [1] is a neural network to be trained to classify handwritten digits. The neural network can learn to assign a handwritten digit, the desired digit, with an accuracy of over 97% depending on the network.

During training each training example is input together with its desired value, the latter being used to determine the error between desired and actual output. Through this training, a cost ('error') function C of the neural network quantifies the average over the error of all individual training examples in a mini batch. An example of such a function is $C(w, b) = (1/2n).\sum_x(y(x)-a(x))^2$, where n indicates the number of training examples in a mini-batch, x is a training example, y(x) its desired output, a(x) its actual output, while w and b represent the weights and biases in the hidden layers.

In the backpropagation algorithm the 'overall error' C(w, b) is reduced by systematically and repeatedly adapting the weights and biases so that the output a(x) from the network approximates y(x) for all training inputs x: the neural network 'learns' [1,2]. Learning takes place on a basis of three sub-sets:

- the training set, used for training the network;
- the test set, for testing the result of training;
- the validation set, used to determine the values of the three 'hyper parameters' learning rate (indicated by η), the size of the mini-batches to be used and the number of epochs of training.

The purpose of the backpropagating algorithm is to find a global minimum of function C, or at least a minimum for which the error is acceptable for the purpose of the application. In practice it is useful to experiment with the size of the changes of the weights and biases supplied by the backpropagation algorithm. The size of the changes will then be multiplied by a positive factor η (eta), the learning rate. To prevent overfitting a regularization parameter (indicated by λ) is sometimes added.

2.2 Software functional size with COSMIC function points

Function points quantify the functional requirements of software and are used for various purposes in software project management, including effort estimation, project planning, project monitoring, productivity studies and benchmarking [3-5]. In the COSMIC functional size measurement method [3, 4] there are four types of data movements:

- entries and exits each move a data group in and out of the software, from/to functional users.
- reads and writes each move a data group from/to persistent storage.

A data group is a set of attributes of interest to a functional user of the software being measured, i.e. a 'thing' in the real world of the functional users about which the software must enter, store, or output data.
The unit of measurement of the COSMIC FSM method is one data movement of one data group, referred to as one COSMIC function point (CFP).

3 Case study: an ML image classifier of manuscript digits

3.1 System view: ML image classifier functions

Of a given file of images of separate individual handwritten digits each image must be classified, i.e. assigned its correct digit. A file of randomly selected training images is available, each image showing the handwritten digit and the corresponding digit it represents. An image of a digit consists of 28x28 pixels. The pixels are greyscale, with a value of 0.0 representing white, a value of 1.0 representing black, and in between values representing shades of grey.

On the basis of functional re-use of a pre-programmed feedforward neural network algorithm, including its cost function, a neural network must be developed that can learn to classify the images of the file. To initialize the network, it is therefore assumed that specifying the numbers of layers and the numbers of weights and biases per layer suffices. Also, to initialize the values of the weights and biases it suffices to specify the mean and standard deviation.

Learning takes place on the basis of three sub-sets of the training images, called the training set, the test set and the validation set. The training set is used for training, the test set for testing the result of training. The validation set is used to determine the values of the hyper parameters learning rate (indicated by η), the size of the mini batches to be used and the number of epochs of training. The size of the mini batches to be applied must be determined. All images are stored with their sub-set name ('training', 'test', or 'validation') for re-use.

The learning performance is monitored by printing and displaying the classification accuracy and displaying the cost (error) per epoch of training. To detect overtraining, two graphs are required, one displaying the accuracy per epoch, the other the cost per epoch. The training set is enlarged by adding one elastically distorted copy of each image in the training set.

For acceptance by the client the classification accuracy is required to be not less than 98%, verified on the basis of the test set. It must be possible to tune the network, i.e. investigate the accuracy and speed by varying the main parameters of the network structure (the number of its layers, the number of units per layer and the hyper-parameters to be applied).

3.2 Functional view of the requirements allocated to generic software

The functional users of the generic software to be measured from the system requirements in section 3.1 are (Fig. 1):

- the data analysts of the neural network, i.e. those who tune the network so as to meet the accuracy requirement;
- the reused neural network algorithm (the feedforward algorithm in Figure 1): this reused software does not have to be measured in this example; it is considered a functional user rather than a software component to be measured.

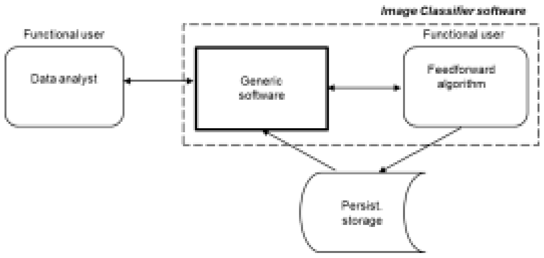

Fig. 1. Context diagram of the generic software

For each image of separate individual handwritten digits, the image classifier software (the generic and algorithm software) assigns and records its correct digit.

The context for this case study explicitly states that there is functional reuse of a pre-programmed feedforward neural network algorithm, including its cost function. Consequently, to create the feedforward neural network the data analyst needs only specify a sequence of numbers in which:

- its length indicates the number of layers,
- each number indicates the number of units in its layer, and
- each unit in the hidden layers has one weight per input and one bias.
- The feedforward algorithm software receives the training parameters, including the hyper parameters and then:

1. assigns random values to the weights and biases on the basis of the mean and standard deviation;
2. groups all training images randomly into mini batches, each consisting of a fixed number of training images;
3. forward propagates the training images of a mini-batch and determines the average cost (deviation, error) of the actual and desired output values of the training images in the mini-batch;
4. backpropagates the changes, which the algorithm determines on the basis of the average error of the mini batch just processed, to all weights and biases backwards through the layers in the network and stores the (updated) values;
5. processes all mini batches of training examples (finishing an epoch of training);
6. repeats steps 2) to 5) for the specified number training epochs.

It is assumed that the feedforward algorithm software stores the training session parameters so that it is possible to train anew with one or more changed parameters, other parameters remaining the same. It also stores all the data needed to meet the requirements of the generic software.

Requirement 0 - Pre-processing. Images must be pre-processed. Note: since pre-processing is specific to each context (and not described-specified in the above system requirements), its measurement is not included in this case-study.

Requirement 1 - Initialization of the feedforward network architecture. The network architecture is initialized by the parameters of the neural network architecture specified by the data analyst.

Requirement 2 - Preparing training.

1. The software enlarges the training set by adding one distorted copy of each image into the training set. The software receives the required expansion instruction from the data analyst. In the absence of details on ‘expansion instruction’ an assumption is made here that this will consist of a single data group. If in other cases there is more in the ‘expansion instructions’, including more than one data group, this could then lead to additional data movements since there would be more data groups.
2. The software receives from the data analyst the number of images for the three sets of training, test and validation images, the members of which are randomly chosen; all images must be stored with their sub-set name.

3. Determining the learning rate η. The software receives the training parameters to produce the graph 'Cost per epoch' (with the validation images). By repeating training with different learning rates, the data analyst can determine a suitable value of the learning rate η with the help of these graphs by comparing the rates of decrease of cost (i.e. error) – Fig. 2.

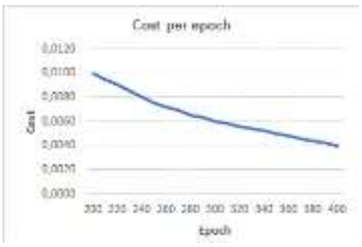

Fig. 2. Cost per epoch

4. Determining a suitable number of epochs. For each execution of the training step (with the validation images), the software must print the graph of Fig. 3.The data analyst determines a suitable number of epochs with the help of the graph 'Accuracy per epoch' by training with the validation images and differing numbers of epochs. The analyst selects the smallest number of epochs with which the required accuracy can be reached – Fig. 3.
5. Determining the mini-batch size. The data analyst determines the mini-batch size from the graph in Fig. 4. The software receives from the data analyst the following inputs to produce the graph – Fig. 4: a number of epochs, a plotting period in seconds, and a number of intended mini-batch sizes.

For execution of the software in the training step (with the validation images), the software must:

- read the classification accuracy per mini-batch size, and
- plot the four graphs of the classification accuracy of the mini-batch sizes versus time, one by one, in one continuous run, i.e. without interruption or stopping.

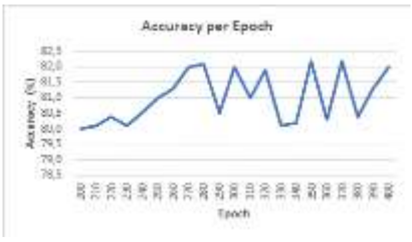

Fig. 3. Accuracy per epoch

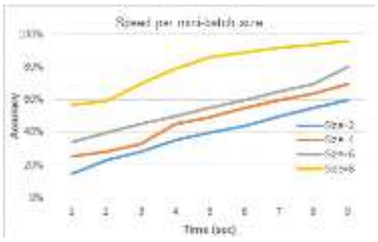

Fig. 4. Speed per mini-batch size

Requirement 3 - Training. The software receives from the data analyst the training session parameters (mean, standard deviation, number of epochs, number of images per mini batch, learning rate (η), regularization parameter (λ) and the sub-set name). During training the number 'classification accuracy per epoch' and the corresponding elapsed training time is printed to monitor the learning performance.

3.3 COSMIC size of the generic software

The measurement of the above functional processes using COSMIC function points and the software data movements within each functional process are presented in Table 1 together with their FP Id, sizes in CFP, data movements (DM) and data groups.

Functional process 1 - FP1: Initialize the feedforward network architecture. The software receives from the data analyst the parameters to create the feedforward architecture: a sequence of numbers the length of which indicates the number of layers and each number indicates the number of units in the layer, and where each unit in the hidden layers has one weight per input and one bias.

Functional process 2 – FP2: Expand the images. The data analyst inputs the required expansion instruction, then this functional process copies each image, distorts it and adds the result to the training set.

Functional process 3 – FP3: Divide images into 3 sub-sets. The data analyst inputs the number of images within each set, then this functional process divides the images into the three sub-sets of training, test and validation images, adds the sub-set name attribute value to each image and stores the image data.

Functional process 4 – FP4: Display cost per epoch (Fig. 2). The data analyst requests to display the graph of the cost ('error') of (the last mini batch of) each epoch.

Functional process 5 - FP5: Display classification accuracy per epoch (Fig. 3). The data analyst requests to display the graph of the classification accuracy of the images.

Functional process 6 - FP6: Determine mini-batch size (Fig. 4). The data analyst inputs a number of epochs and a number of mini-batch sizes to be examined and determines the desired mini-batch size visually on the basis of the graph. The software**:**

- executes neural network algorithm with validation data,
- graphically plots the last known epoch accuracy at each point of time.

Functional process 7 – FP7: Train the network. The data analyst inputs the training session parameters (mean, standard deviation, number of epochs, mini-batch size, learning rate η, regularization parameter λ) to train the network. For monitoring the training, the epoch ID, number of correctly classified images, total number of images and elapsed time must be printed.

The total functional size of the generic software is the sum of the sizes of its functional processes FP1 to FP7, that is:

4 CFP + 4 CFP + 4 CFP + 6 CFP + 6 CFP + 10 CFP + 5 CFP = 39 CFP

Table 1. Functional Sizes of FP 1 to FP7 in CFP

FP Id & size	DM	Data group/ data attributes
FP 1 size = 4 CFP	Entry	Number of units from data analyst
	Exit	Number of units to feedforward algorithm
	Entry	Result of initialization from feedforward algorithm
	Exit	Error/confirmation message to data analyst
FP 2 size = 4 CFP	Entry	Expansion instruction
	Read	Image data
	Write	Training image
	Exit	Error/confirmation message
FP 3 size = 4 CFP	Entry	Sub-set of images (sub-set name, number of images)
	Read	Image data
	Write	Image data with sub-set name
	Exit	Error/confirmation message
FP 4 size = 6 CFP	Entry	Epoch ID range
	Read	Epoch ID, epoch cost stored by the feedforward algorithm
	Exit	Epoch ID (x-axis, multiples of 50)
	Exit	Cost (y-axis, multiples of 0,001)
	Exit	Epoch cost
	Exit	Error/confirmation message
FP 5 size = 6 CFP	Entry	Epoch ID range
	Read	Epoch ID, epoch accuracy stored by feedforward algorithm
	Exit	Epoch ID (x-axis, multiples of 10)
	Exit	Epoch accuracy (multiples of 0.5%)
	Exit	Epoch ID, epoch accuracy
	Exit	Error/confirmation message
FP 6 size = 10 CFP	Entry	Training session parameters (without number of images per mini batch)
	Exit	Training session parameters (without no. of images per mini batch) to feedfor-ward algorithm
	Entry	Mini-batch size, input the sizes to be compared
	Exit	Mini-batch size, sizes to be compared to feedforward algorithm
	Read	Elapsed time, epoch accuracy per mini-batch size stored by feedforward algo.
	Exit	Elapsed time (x-axis, in seconds)
	Exit	Epoch accuracy (y-axis: multiples of 20%)
	Exit	Mini-badge denotation from entry above
	Exit	Epoch accuracy per mini-batch size at point of time
	Exit	Error/confirmation message
FP 7 size = 5 CFP	Entry	Training session parameters (mean, std dev., number of epochs, no. of images per mini batch, learning rate (η), regularization parameter (λ), sub-set name)
	Exit	Training session parameters to feedforward algorithm, sub-set name
	Entry	Epoch ID, number of correctly classified images, total number of images, elapsed time from feedforward algorithm
	Exit	Print epoch ID, no. correctly classified images, total no. images, elapsed time
	Exit	Error/confirmation message

4 Summary and future work

In this paper an ML image classifier case study was used to illustrate how to move from a ‘system viewpoint’ of ML functionalities to the description of the generic software development tasks for which expertise is more widely available and less specialized. Since these coding tasks can then be delegated to staff with programming expertise, the ML experts can be freed up, allowing them more freedom to address additional ML challenges. The case study illustrates how the generic software functionality has been segregated from the ML specific functionalities. In this paper, the COSMIC function points technique was used to carry out the segregation. While it recognizes that in every software there will be unique aspects making each software distinct from any other, there are functionalities that are common and generic throughout. This genericity is at the basis of software functional sizing measurement methods such as COSMIC function points – ISO 19761, and has been used to address two objectives:

- segregation of the generic functionality to be allocated to software and not specific to ML;
- use of the international standard to size the generic functionality identified.

Success in both facilitates delegating tasks to ML data analysts and software developers, as well as collecting data in a standardized fashion to develop estimation models for planning purposes, and for on-going monitoring of the generic software tasks within an ML development project. It is to be noted that success in this endeavor will allow parallelism of tasks in ML projects, with the possibility of shortening their development cycle.

Future work will include additional steps to verify the breadth and depth of the generic software functions described in the set of ML requirements used, identification of ambiguities, and updates with corresponding size adjustments. Further validation will include verification with actual ML software already developed by industry. Additional empirical research work is also required to consolidate the insights developed in the research reported here. In particular, additional case studies from other domains may provide additional types and sources of generic functionality that could then be considered for scaling purposes.

References

1. Nielsen, M. A.: Neural networks and deep learning, Determination Press, available on www.neuralnetworksanddeeplearning.com (2015).
2. Graupe, D.: Deep learning neural networks, World Scientific (2016).
3. COSMIC Group: The COSMIC Functional Size Measurement Method – Measurement Manual, version 4.0.2, available on https://cosmic-sizing.org/publications/measurement-manual-v4-0-2/, (2017).
4. Abran, A. and Dumke, R. (Eds.): COSMIC Function Points Theory and Advanced Practices, CRC Press. ISBN 978-1-4398-4486-1 (2011).
5. Abran, A.: Software Metrics and Software Metrology, John Wiley & Sons and IEEE-CS Press, New Jersey, p. 328 (2010).