Article from

CompAct

April 2018
Issue 57

Large Portfolio Variable Annuity Valuation Powered by GPUs and Deep Learning

By Huina Chen and Henry Bequet

Recent technological advancements in Graphical Processing Units (GPUs) and deep learning are drastically changing the landscapes of many fields, including financial analytics. In this paper, we apply GPUs and deep learning to address computational challenges in the valuation of large portfolios of variable annuities. Our numerical experiments show that using GPUs leads to a 10 times speedup compared with traditional Monte Carlo valuation on multi-threaded CPUs; while using GPU-based deep learning achieves another order of magnitude performance improvement.

INTRODUCTION

Variable annuity is a type of insurance contract that allows for asset accumulation via investing in mutual funds provided by insurers. It often provides guarantees, also called riders, to protect the policyholder from market downturns, such as the 2007–2008 financial crisis. The predominant guarantees are Guaranteed Minimum Death Benefits (GMDB), Guaranteed Minimum Accumulation Benefits (GMAB), Guaranteed Minimum Income Benefits (GMIB), and Guaranteed Minimum Withdrawal Benefits (GMWB).

Insurance companies holding large portfolios of variable annuity policies are exposed to risks from honoring the guarantees should adverse events occur. A popular risk management practice is to dynamically hedge these guarantees. Insurers buy hedging portfolios consisting of financial derivatives, hoping their payoffs offset the payouts of the guarantees to policyholders. The hedging portfolio requires intraday rebalance according to the Greeks, such as dollar delta, of the guarantee liabilities. Traditionally, the guarantees are evaluated using Monte Carlo simulations on every policy in the portfolio. This is because the product structure is complicated and the portfolio is highly heterogeneous (Gan and Lin, 2015). Monte Carlo simulations can be time consuming. For example, in one of our experiments, it took 64 CPU cores 44 minutes to calculate the dollar deltas for a portfolio of one million synthetic variable annuities with either a GMDB rider or both GMDB and GMWB riders for 10,000 scenarios.

Several papers have been published to efficiently evaluate the dollar deltas of large portfolios of variable annuities using a spatial interpolation framework (Gan, 2013; Gan and Lin, 2015; Hejazi et al., 2015). The idea is to select a small sample of variable annuity policies (the representative contracts), calculate their dollar deltas with the expensive Monte Carlo simulations, and then estimate the dollar deltas of other policies in the portfolio as weighted sums of the pre-calculated dollar deltas of the representative policies. The weights are determined according to the distances between the focal policy and the representative ones. Gan and Lin (2015) pointed out that a number of modeling choices significantly impact the accuracy of the spatial interpolation results, including the sampling method and the number of the representative contracts, as well as the distance function used to calculate the weights in the weighted sum. Hejazi and Jackson (2016) proposed a partial neural network to learn the distance function.

In this article, we use GPUs and deep learning to solve large portfolio variable annuity valuation problems with high speed and accuracy. It takes a GPU card with 4,992 cores less than one minute to calculate the dollar deltas for the one million policies using Monte Carlo simulations. The speed is attractive for intraday rebalances of the dynamic hedging program. However, GPUs alone cannot compute fast enough for capital calculation when nested simulations are required. If the inner loop simulations are replaced with the approximation function trained by deep learning, we can perform capital calculation on

the same portfolio 10 times faster than the nested Monte Carlo simulations, cutting computation time from days to hours.

GPUS

GPUs were initially designed to perform graphical operations for video games. These operations often involve similar or repeated computations on multiple frames, and need to be completed as fast as possible. The technology was later used in the non-graphical areas, such as solving large systems of equations. These non-graphical tasks gave rise to the General Purpose GPUs or GPGPUs. Good candidates of GPGPUs are applications that require identical processing on many versions of the data, and have a high ratio of computations versus the amount of data that needs to be processed. For the rest of this article, when we refer to GPUs, we really mean GPGPUs.

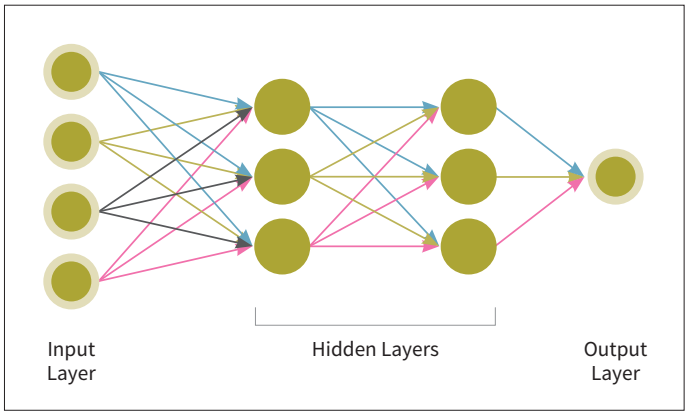
The real power of GPUs lies in its price tag. One NVIDIA Tesla K80 GPU card has 4,992 threads and costs less than $4,000. We can easily insert four K80 cards into a computer and compute 20,000 tasks simultaneously. Computing 20,000 tasks in parallel using CPUs will require a computer grid of millions of dollars. The low cost of GPUs makes daunting computation tasks such as deep learning economically possible. In the meantime, machine learning software, such as SAS (Bequet and Chen, 2016), provide data scientists with an easy methodology to call GPU functions without the knowledge of GPU languages. Inexpensive hardware and easy-to-use software liberate application developers from computational challenges, and enable them to focus on what they are good at: defining problems, collecting and processing data, designing algorithms and analyzing results. Consequently, more and more deep learning applications are springing up in a wide range of fields including health care, transportation, speech recognition, environmental science and more.

DEEP LEARNING

Deep learning is a branch of machine learning. It is inspired by the human brain's biology and its ability to learn via observing and experiencing. Deep learning models are large multilayer artificial neural networks. Artificial neural network started in the 1940s. Its winding journey finally entered into a productive era in recent years, thanks to fast and economical computer accelerators such as GPUs, a flood of digitalized data from the Internet, and virtually infinite storage spaces.

An artificial neural network is a network of computational neurons organized in layers. It has one input layer, at least one hidden layer, and one output layer. We call a neural network a deep net when there are more than one hidden layers. The quintessential deep learning models are the feedforward deep networks. In a feedforward deep net, the input data enter the input layer, go through one hidden layer after another in sequence, and reach the output layer to produce the target value(s). Neurons of one layer take as input the outputs of neurons in the previous layer. There are no feedback connections in which the outputs of a layer are fed back to itself or the previous layers. The feedforward computation can be described as a directed acyclic graph shown in Figure 1.

Figure 1
A Feedforward Deep Learning Network



Consider a feedforward deep network with L layers, and for each layer l, there are I_l nodes, in which the first layer output x^1 equals to the feature vector x from the input data, and the last layer output x^L corresponds to the calculated target variable vector $\hat{y}$. The calculation in each node through the neural network can be recursively represented as

$$x_i^l = g_i^l\left(w_i^{l^T} x^{l-1} + b_i^l\right) \text{ for } l = 2, \ldots L \text{ and } i = 1, \ldots I_l,$$

in which function g_i^l is an activation function, such as RELU, logistic sigmoid, or hyperbolic tangent functions; vector w_i^l contains the weights; and scalar b_i^l is a bias term. For notational simplicity, we use vector θ to include weights and biases from all the nodes in the model. The objective is to choose θ to minimize a cost function $J(\theta)$. The cost function defines the error between the target value $\hat{y}$ calculated by the network, and the desired value y passed from the input data. $J(\theta)$ can be mean square error (MSE) for regression problems, or cross-entropy for classification problems. It can be other function depending on the specific application.

Two types of financial applications are good candidates for function approximation using feedforward deep network. 1) Fit functions for hard-to-model assumptions, like policyholder behaviors. 2) Approximate functions to replace computational intensive calculation, such as seriatim valuation or stochastic simulation. An advantage of using feedforward deep network to approximate functions is that there is no

need for prior knowledge about the true model. Constructing a neural network is more art than science. Financial analysts, such as actuaries, can construct a feedforward network by trying different combinations of hyperparameters. There are two types of hyperparameters. 1) Model hyperparameters, such as the total number of neurons, the connection between neurons, and the activation functions. 2) Training method hyperparameters, such as cost function, optimization solver, learning rate and initial weights. Often, there is not a single best combination of hyperparameters for a particular problem. Usually the bigger the network and the larger the training data size, the better approximation the trained network achieves. However, it comes with the price of computational efficiency. Our goal is to find a satisfactory set of hyperparameters to achieve target accuracy and efficiency with the constraints of available computation power and training data.

NUMERICAL EXPERIMENTS

In this section, we use an example of large portfolio variable annuity valuation to demonstrate the speed and accuracy that GPUs and Deep Learning can achieve.

We compare the performance of the valuation in the following three settings: Monte Carlo valuation using multi-threaded CPUs, Monte Carlo valuation using GPUs, and GPU-based deep learning valuation. All tests are done in the testing computer with 500GB RAM, 64 hyper-threaded cores at 2.30GHz, and an NVIDIA Tesla K80 GPU card with 4,992 CUDA threads.

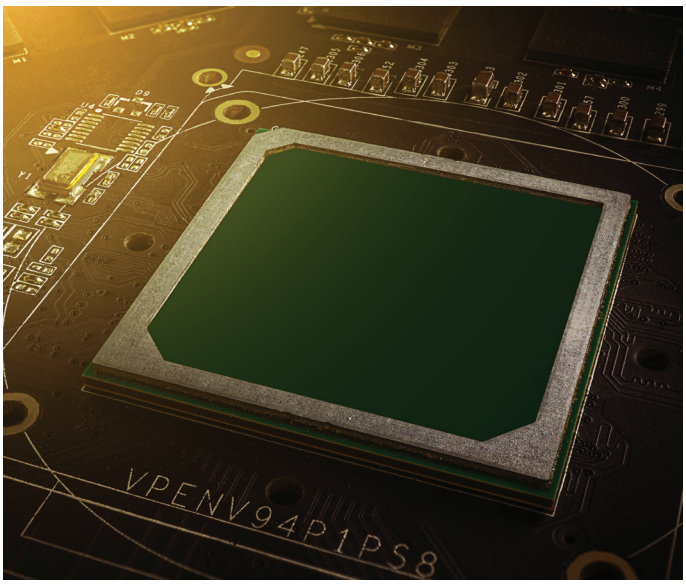
Portfolio Data

The portfolio consists of one million synthetic variable annuity policies with either a GMDB rider or both GMDB and GMWB riders. Each policy has seven attributes which contribute to the policy's valuation. They are guarantee type, gender, age, account value, guarantee value, GMWB withdrawal rate and maturity. Each policy is generated by uniformly drawing values of each attribute from its respective range. For the purpose of comparison, we use the same attributes and value ranges of the input portfolio listed in Table 1 in Hejazi et al. (2015). We also use the same log-normal distribution with a 3 percent risk-free rate and a 20 percent volatility to generate 10,000 risk neutral equity scenarios. The mortality rates follow the same 1996 IAM tables provided by the Society of Actuary. The projection horizon is 25 years.

The model calculates the dollar delta for each of the one million variable annuity policy.

Monte Carlo Valuation

The Monte Carlo valuation algorithm follows Gan (2013). For each policy, we calculate the dollar delta for each of the 10,000 equity scenarios. A policy's dollar delta is the average of the dollar deltas across all scenarios. It takes 44 minutes using the 64 CPU



cores, or 52 seconds using the 4,992 GPU threads, to compute the million policies' dollar deltas on the testing computer.

Deep Learning Valuation

To achieve higher performance, we use deep learning to approximate Monte Carlo valuation on the million variable annuities.

In our experiment, we construct a fully connected feedforward deep neural network with one input layer, eight hidden layers, and one output layer. The input features include two categorical features and six numerical ones. Categorical features are guarantee type (zero for GMDB and one for GMDB+GMWB) and gender (zero for male and one for female). Numerical features are maturity, age, account value, GD/AV (the ratio of guaranteed death benefit over account value), GW/AV (the ratio of guaranteed remaining withdrawal amount over account value) and withdrawal rate. GW/AV and withdrawal rate are zero for policies with only the GMDB rider. For policies with both GMDB and GMWB riders, the time zero values of GD/AV equal to GW/AV equal to the ratio of guarantee value over account value. To ensure fast convergence for network training, we standardize the numerical feature values by taking their z-scores. Each hidden layer has 1,024 neurons with RELU activation function. The output layer calculates the weighted sum of the eighth hidden layer's 1,024 outputs to produce the value of target variable dollar delta.

To train the network, we generate 10,000 variable annuity policies, 8,000 for training and 2,000 for validation. They are 1 percent the size of the input portfolio we need to evaluate. They follow the same distribution as the million-policy portfolio we want to evaluate. We calculate their dollar deltas using Monte Carlo valuation, which takes half a second on the GPUs. Should

valuation results from past valuation dates be available, there would be no need to generate new training data with Monte Carlo simulations. Actuaries who are working on production have plenty of historical data to use as inputs for network training.

We train the network using back propagation with the Adam optimizer to find a set of weights and bias to minimize the cost function

$$J(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(\frac{\hat{y}_i - y_i}{1+|y_i|}\right)^2.$$

To speed up training, we employ a mini-batch training technique with a batch size of 100. The learning rate is set to 0.001. The initial weight values are generated using truncated normal with mean 0 and standard deviation 0.1. The initial bias values are set to zeros. The network is trained for 88,800 iterations within 14 minutes on the GPUs. The values of $J(\theta)$ are 0.0005 for the training set and 0.0028 for the validation set.

It is worth pointing out the importance of selecting a good cost function that suits the particular problem we are solving. The dollar deltas can vary in a wide range among different variable annuity policies in a portfolio. We do not choose

$$\text{MSE}(\theta) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2,$$

because it favors those weights and bias that reduce the errors for the y_i's with large absolute values; therefore the accuracy of the model for the y_i's with wsmall absolute values compromised. We also choose not to use

$$\text{MSE}(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(\frac{\hat{y}_i - y_i}{y_i}\right)^2,$$

because it is very likely that some policies have dollar deltas at or very close to zero. Using MSRE as the cost function would cause numerical problems. We try a few variations of MSRE. Cost function

$$J(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(\frac{\hat{y}_i - y_i}{1+|y_i|}\right)^2$$

gives us the best optimization result.

Once the network is trained, it can be used to approximate the Monte Carlo valuation for variable annuity policies with similar characteristics as the training data, so long as the risk neutral assumptions for equity scenarios stay the same. The trained deep net can replace the entire one-level Monte Carlo valuation. It can also substitute each inner loop Monte Carlo valuation at all time steps along the outer loop scenarios for a nested simulation. In our example, it takes four seconds to compute the dollar deltas for the 1,000,000 policies using the trained deep net. The relative error of the portfolio dollar delta

$$\frac{\sum_{i=1}^{N}\hat{y}_i - \sum_{i=1}^{N} y_i}{\sum_{i=1}^{N} y_i}$$

is 0.0004. It would have taken the same GPU card eight days to complete the nested Monte Carlo valuation for the same portfolio with 1,000 outer loop real world scenarios each having 10,000 inner loop risk neural paths. With the trained deep net to perform the inner valuation, we can complete the nested calculation in 14 hours. We can further reduce the computation time by using more GPU cards simultaneously.

Using the Many Task Computing framework (Bequet and Chen 2017), we are able to integrate CPU and GPU tasks in the same computation job flow without any manual data movement. The end-to-end computation seamlessly conducts data generation and enrichment on CPUs, Monte Carlo simulation and neural network training/inference on GPUs. Figure 2 shows the high level computation job flow.

Performance Results

Table 1 shows the performance results for evaluating one million variable annuity policies using different technologies. We list the hardware information to provide reference for interested readers.

Figure 2
An End-to-End Job Flow for Variable Annuity Valuation With Deep Learning

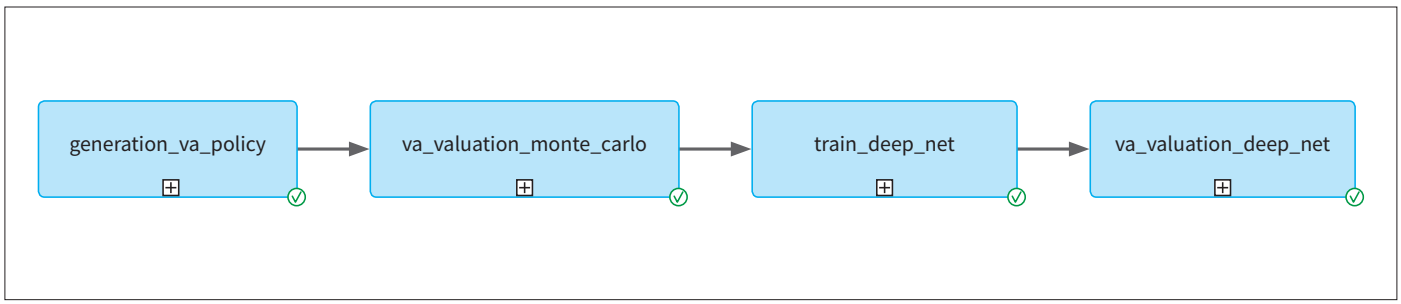


Table 1
Performance under Different Technologies

Technology		Hardware	Monte Carlo Simulation Times (in seconds)
CPU	Monte Carlo Valuation with SAS	64 HT Intel E5-2698 v3 @ 2.30 GHz 500 GB RAM	2,640
GPU	Monte Carlo Valuation with CUDAC	NVIDIA K80 @ 840 MHz 4,992 CUDA Cores	52
	Deep Learning with CUDAC	NVIDIA K80 @ 840 MHz 4,992 CUDA Cores	4

The four-second computation time with deep learning is the time for inference only. We do not include the network training time here because the neural network only needs to be trained once, and can be used for inference many times, as long as the portfolio's characteristics and company's long term view on equity movements do not change.

CONCLUSION AND FUTURE WORK

We have shown that GPUs and GPU-based deep learning can improve computation efficiency by several orders of magnitude. This facilitates timely analysis for better decision making.

As actuaries continue pushing the boundary of product innovation, more complicated modeling is expected, which demands higher computing performance. Fortunately we are living in a world of constant technology breakthroughs. Application Specific Integrated Circuits (ASICs) designed for deep learning training and inference will perform analytics even faster than what we have described in this paper. Preliminary results (Joupi, et al., 2017) indicate that we would at least get another order of magnitude of performance improvements. We will work on financial analytics with ASICs-based deep learning and share the findings with readers in the future. Meanwhile, we see deep learning as a nice tool to help actuaries discover the real patterns of policyholder behaviors. Policyholder behaviors, such as guaranteed living benefits utilization and dynamic lapse, are hard to model. Because deep learning algorithms learn models directly from data, we believe actuaries can train deep neural networks with relevant data and find the credible policyholder behavior assumptions for better valuations. ■

Huina Chen is a principal research statistician developer at SAS Institute. She can be contacted at *huina.chen@sas.com*.

Henry Bequet is a director of development at SAS Institute. He can be contacted at *henry.bequet@sas.com*.

REFERENCES

Bequet, H., Chen, H., 2016. Many Task Computing With GPU Acceleration: An Infrastructure for Easy Principle-Based Modelling. 2016 Society of Actuary Annual Meeting & Exhibit.

Bequet, H., Chen, H., 2017. Accelerate your SAS Programs with GPUs. Paper SASSD706-2017.

Gan, G., 2013. Application of Data Clustering and Machine Learning in Variable Annuity Valuation. Insurance: Mathematics and Economics 53 (3) (2013) 795–801.

Gan, G., Lin, X.S., 2015. Valuation of Large Variable Annuity Portfolios under Nested Simulation: A Functional Data Approach. Insurance: Mathematics and Economics 62, 138-150.

Hejazi, S.A., Jackson, K.R., Gan, G., 2015. A Spatial Interpolation Framework for Efficient Valuation of Large Portfolios of Variable Annuities.

Hejazi, S.A., Jackson, K.R., 2016. Efficient Valuation of SCR via a Neural Network Approach.

Joupi et al., 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. 44th International Symposium on Computer Architecture (ISCA), 2017.