Ultra-fast and Accurate Derivatives Pricing with Deep Learning

by Kareem Kudus and Ivan Sergienko, Riskfuel

Ian Finder, Senior Program Manager, Accelerated HPC Infrastructure at Microsoft, recently published a blog post "Azure GPUs with Riskfuel's technology offer 20 million times faster valuation of derivatives". Following an exciting, and at times, heated discussion with the quant finance community, we decided to follow up here to provide more technical details. We recommend reading Ian's blog before proceeding further.

Instrument

The financial instrument we study is the foreign exchange (FX) barrier option. Barrier options are path-dependent exotics that share many similarities with ordinary ('vanilla') options. The essential difference is that they become activated (or extinguished) only if the underlying FX spot price breaches a predetermined level (the barrier). This additional condition makes a barrier options less expensive than a vanilla option with the same strike and maturity.

There are many different 'flavours' of barrier options depending on whether the barrier is 'knock-in' (activating) or 'knock-out' (extinguishing). For example, a barrier option where the underlying must drop below a barrier in order for the option to be activated is referred to as a 'down-and-in' option. There are also down-and-out, up-and-in, up-and-out options, as well as variants with both upper and lower barriers (i.e. double knock-outs). In what follows, we showcase the results for a double knock-out option with a partial barrier, i.e. the barrier observation window starts in the future and lasts until option maturity.

Model

The presence of the barrier feature makes valuation of these options more difficult than vanilla options. Unlike barrier options, vanilla options are not path-dependent and their value is only dependent on the distribution of the FX values at maturity. This allows vanilla option values to be calculated analytically using the closed form Black-Scholes equation. This is not the case with barrier options. Instead, numerical methods must be employed. Some common numerical approaches are discussed in this Wikipedia article.

The traditional model we consider uses the Crank-Nicolson finite-difference method, where a preliminary calibration step generates local volatilities from a full FX delta-volatility surface. In total, the Riskfuel deep learning model was trained to reproduce this model along 81 input dimensions: 60 points on the volatility surface, 16 points on the domestic and foreign interest rate curves and a few trade specifics such as time to exercise and barrier level. While dimensionality is important, so is the size of each dimension. This model was trained over a wide range of input values suitable for handling extreme scenarios. It remained well within its valid range even during the recent COVID-19 market rout.

Generating Training Data

Once the domain of approximation has been determined, the first step is to generate training data on the domain. We use synthetic data, rather than historical data, to train our models. This allows our neural networks to produce accurate values during future market states that may not have previously occurred. In addition to market data, we cover a wide range of trade parameters, not only those that exist in the portfolio at the time of training.

Part of this process involves determining the best trade-off between the amount of training data produced, the accuracy of the training data, and the desired accuracy of the Riskfuel model. Riskfuel models get increasingly accurate as the amount and quality of training data increases. The interplay of these two parts is non-trivial. For example, as shown in the seminal paper “Deeply Learning Derivatives” by R. Ferguson and A. Green, a deep neural net can be *more accurate* than traditional models when given enough training data. We re-affirm that result below.

As detailed in the Azure blog, the traditional finite-difference model takes on average about 2.25s to value one trade on one core of an Azure F72s_v2 instance. This is likely slower than typical production settings. Since we only need to generate training data once, we can afford to dial up the accuracy on the traditional model. For the FX barrier model discussed here, we generated 100 million training data points over a domain large enough to cover the needs of an XVA trading desk, which use a wide range of scenarios to simulate future counterparty risk exposures. As was pointed out to us, with a single valuation averaging more than 2s, generating the training set requires about 7 core-*years*. That’s where Azure comes in. Using 100 F72s_v2 instances (7,200 cores) we were able to generate the training data in less than 10 hours.

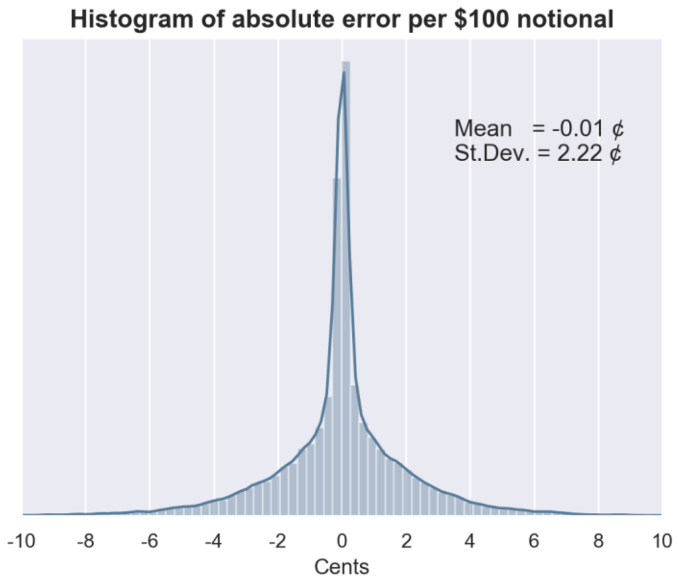
Training

Once we have our training data, we are ready to start the training phase. We generally start with a rectangular architecture, where each of the hidden layers has the same number of neurons. Later, we go through a post-processing step where we remove redundant neurons to optimize both memory usage and speed of computation at inference time. We test a wide array of hyperparameter combinations including different depths, widths, and learning rates.

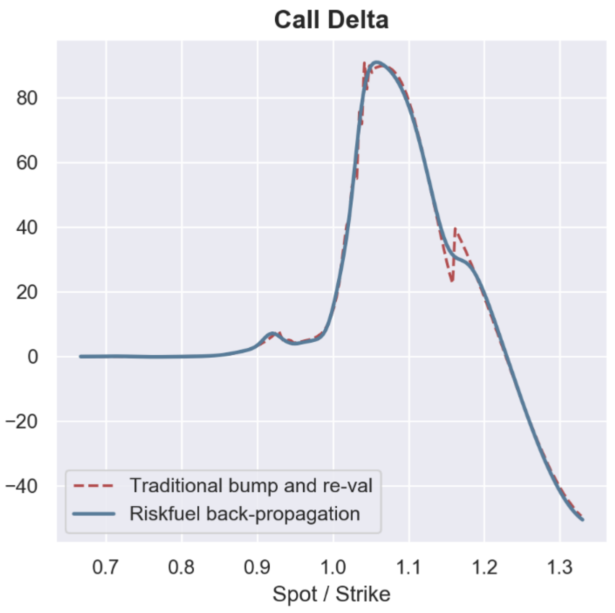
As part of our study with Microsoft, we had access to Azure ND40rs_v2 VMs. These VMs have 8 Nvidia V100 GPUs connected together with high-speed direct inter-GPU NVLINK. This brought significant performance improvements versus our on-premises cluster sporting a few dozen 1080ti GPUs. With 32 GB of GPU memory on each Azure V100 we can hold our entire dataset in memory, reducing costly transfer time from CPU to GPU memory. NVLINK provides a total bandwidth of 300 GB/second; this makes scaling up to multi-GPU training seamless. Using the Azure ND40rs_v2 VM we can now train a model in a matter of hours rather than days.

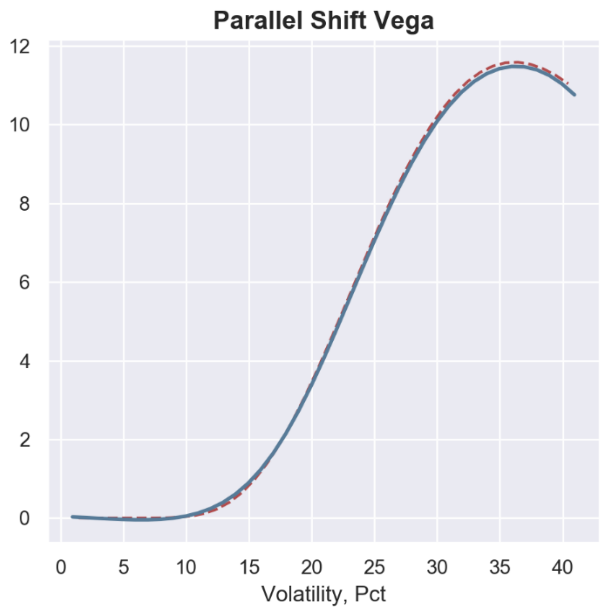
Accuracy

Before we address the model's speed, we discuss the accuracy of our learned model. The histogram below shows the distribution of errors defined as the difference of option values between the neural network approximation and the traditional client model. The error values are in cents per $100 notional. The deals are from a random validation set, which was not used by the model during training or model selection. The range of option values in this set is $0 to $80. The error histogram shows practically no bias and very high accuracy with Mean = -0.01 c and St. Dev. = 2.22 c.

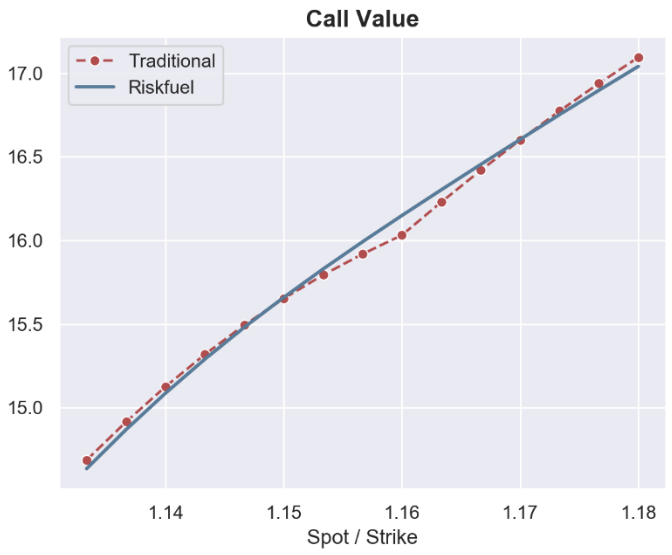


In addition to pricing, quantitative finance models have to produces "the greeks" – derivatives of price functions with respect to various risk factors, which are used in execution of trading strategies and risk management. Can deep learning models be as accurate as traditional pricers in producing these values? Our answer is yes! And they can be even better because, under the hood, neural nets are smooth analytical formulas. The graphs below show the derivatives with respect to the FX spot price (delta) and volatility (vega). The non-monotonic behaviour of both graphs is due to volatility smiles and the presence of barriers, a well-known effect for barrier options. The red broken lines were calculated using the traditional model and the bump-and-revalue finite-difference approach.





The delta plot on the left is particularly interesting around the point Spot/Strike = 1.16. While the deep learning back-propagation algorithm produces a smooth function, the application of a naïve finite-difference scheme to the traditional model shows a significant jump. To investigate this further, in the next graph we zoom in on the plot of the value of the call option vs Spot.



We observe that the two value calculations are quite close. However, the traditional model produces a small dip at Spot/Strike = 1.16. This result is spurious, due to a numerical discretization error in the model's underlying finite-difference scheme. As a result of this small dip, the slope jumps by about a factor of two, consistent with the above delta graph. The Riskfuel model does not have this spurious singularity. In effect, our deep neural network learned to distinguish and ignore numerical noise from discretization error, a significant result for practical applicability of neural networks for derivatives pricing and hedging.

Speed

In the following table we summarize our training time on different hardware configurations. The times listed describe how long one mini-batch of more than 32,768 (2^{15}) training samples took to process during training. This includes both a forward and backward pass through the computational graph. During the forward pass we calculate all the 32,768 predicted option values, and during the backpropagation we calculate the massive gradient vector in the space of model parameters (weights and biases) required for the training algorithm.

Once the model is trained, the backpropagation facilities of modern deep learning packages (Riskfuel uses PyTorch) could also be used to calculate the greeks, as discussed above. With the exception of a few market leaders which have implemented Algorithmic Differentiation (a tailored version of back-propagation that typically requires doubling of the code base) in their code, most banks use the bump and re-value method for calculating the greeks. In our case, this would require calculating the option values an extra 81 times, resulting in total of 82 calculations per trade. We include this multiplier in our calculations of the throughput.

Per batch of 32,768 training samples

Processor	Time for one forward and backward pass (ms)	Throughput (effective valuations per second)
Intel Xeon E5-2690 CPU	277.0	9,700,274
V100 GPU	16.2	165,862,716
V100 with Mixed Precision GPU	9.0	298,552,888
8 x V100 Mixed Precision GPUs	3.0	895,658,666

To put these numbers in context, the traditional model takes about 2.25 s to perform a single valuation. Even using a fairly large server, such as the Azure F72s_v2 with 72 CPU cores, this results in a throughput of just 32 valuations per second.

From the performance data, we can see that Riskfuel's model produces a throughput over 303,133 (9,700,274 / 32) times higher than the traditional model when run on a CPU. Moving our model onto a single V100 increases our throughput twentyfold. Using mixed precision to exploit the tensor cores on the V100 more than doubles our throughput. Finally, we can achieve an additional 3x increase in throughput by scaling across all eight GPUs on the Azure ND40rs_v2 Virtual Machine. Our max throughput on the Azure ND40rs_v2 Virtual Machine is about 28 million (895,658,666 / 32) times higher than the traditional barrier model.

To learn more about Riskfuel, visit us at riskfuel.com or email sales@riskfuel.com.