ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Stochastic Finance Group
Prof. Josef Teichmann

Semester Project

Learning Risk-neutral Measures with Neural Networks

Spring Term 2022

Supervised by:
Josef Teichmann
Jakob Heiss
Hanna Sophia Wutte

Author:
Michele Meziu

Contents

Abstract

A common assumption in Mathematical Finance is the existence of a risk-neutral measure, which gives the probability distribution of the future price of assets traded in an arbitrage-free market. In past literature it was often assumed that for stocks the density of this distribution is log-normal. However data from the stock market contradicts this hypothesis. In this work, we use data regarding European options on the NIKKEI 225 index to learn the risk-neutral probability density function of this index. We use the general framework of Deep Learning to learn the distribution without making assumptions on its shape. Our methodology can be used to price any kind of derivative with fixed maturity, also derivatives with more complex payoff functions than European options.

Chapter 1

Introduction

1.1 Motivation

Risk-neutral measures are an essential tool in the pricing of financial derivatives. Because of the fundamental theorem of asset pricing, it is possible to determine the price of a derivative security as the discounted expectation of its payoff under the risk-neutral measure. Moreover, a characterization of the risk-neutral probability density of asset prices is extremely useful for arbitrage and hedging.

The type of securities we focus on in this work are European options. They are widely traded worldwide in the financial markets and they are easier to model than their American counterparts. In particular we work on European options on the NIKKEI 225 stock index traded in the Tokyo stock exchange.

In the past the most common way to price European options on stocks was the Black-Scholes formula. The derivation of this formula relies on assumptions which are not thought to hold in real-world financial markets. One major discrepancy between the Black-Scholes assumptions and real-world stock prices is that stock prices follow a geometric Brownian motion, which implies that the distribution of possible stock prices at the end of any finite interval is a log-normal distribution. This is not observed in practice. On the contrary, extreme price changes are observed much more often than it would be if prices were log-normally distributed.

Nowadays one of the most used approaches in financial derivatives pricing is Deep Learning. One of the main reasons of its success is that artificial neural networks (ANNs) are non-parametric models and do not intrinsically encode assumptions on the market or on asset prices. There are many possible approaches in using ANNs to price European options. The most common one is to directly learn the option price or the implied volatility given option parameters. In our work, instead, we aim at using prices of options on the NIKKEI 225 stock index with maturity T to learn the risk-neutral density of the index at time T. The resulting risk-neutral density is used to accurately price options not included in the training set with the same maturity date.

1.2 Overview and related work

Because of their power and versatility, ANNs have seen wide use in the financial industry to price and hedge options. A recent literature review on this topic is the work of Ruf and Wang [1].

In this work it is clear that the most common approaches are to learn the option price C, the price-strike ratio C/K or the implied volatility σ_I.

However, learning the risk neutral density has been tried in the the work of Schittenkopf and Dorffner [2]. Here the approach is to learn the risk-neutral density of the underlying asset by using mixture density networks. They use the obtained density to price European options on the FTSE 100 index. They report only the absolute pricing error, without mentioning how large the error is relative to the true option price. It would have been interesting to compare the results of our model with theirs, yet this fact makes the comparison of their accuracies meaningless.
Neural networks have been used to learn the risk-neutral density and to price European options in the work of Lai [3]. This paper compares three different methods: kernel regression, spline interpolation and a neural network. The conclusion of the work is that the neural network has the worst performance of the three. The model that is used is a weighted sum of RBF kernels, with only one hidden layer of RBF neurons. Moreover, a different approach is used for obtaining the risk-neutral density. The author does not try to directly model the density, but to firstly derive a pricing function C and then differentiating it twice with respect to the strike price K to obtain the risk-neutral density function. Contrary to this method, we do not need to compute explicit derivatives, which would be intractable for large ANNs.

Chapter 2

Preliminaries

In this section we introduce basic notions that will be used throughout the work.

2.1 Pricing a derivative using the approximated risk-neutral measure

Assume the risk-free rate r is zero. Let A be a fixed asset and D a derivative having A as an underlying asset. Let X be the non-negative random variable expressing the value of A at $t = T$. Given a risk neutral measure $\mathbb{Q}$, let the risk-neutral probability density of A at time T be $q(x)$. If $p(x)$ is the payoff function of D (we assume that the payoff is paid completely at time T), then the price of D at $t = 0$ can be simply written as

$$C(D) = \mathbb{E}_{\mathbb{Q}}[p(x)] = \int_0^\infty p(x)q(x)dx.$$

The goal of this work is to approximate $q(x)$ with a density $n_\theta(x)$ parametrized by the neural network weights θ.
Moreover we want to approximate the integral by using Monte-Carlo integration. Given a proposal distribution $f(x)$, we approximate the price of the derivative $C(D)$ using N Monte-Carlo samples

$$C(D) = \int_0^\infty p(x)q(x)dx \approx \sum_{i=1}^{N} \frac{p(x_i)n_\theta(x_i)}{f(x_i)}. \tag{2.1}$$

$x_1, \ldots, x_N$ are sampled according to the proposal distribution $f(x)$. In our work we used the uniform distribution on the interval $[\epsilon, L]$, where ϵ is a small positive number and L is a large number such that the probability that $X > L$ is negligible. We tried also using a lognormal distribution as $f(x)$, but this gave numerical stability problems and made training harder because of the possibly very small denominator in 2.1, which causes the value of the loss to explode.

Chapter 3

Methods

In this section we explain the main characteristics of our model.

3.1 Model architecture

We use a Multi Layer Perceptron model (MLP), with two nodes in the input layer: one for the price x, for which we want to query the probability density at time T, and one for its logarithm $\log x$. We think that adding the logarithm to the input makes it easier for the MLP to learn changes in the second derivative of the output distribution.
There is one node in the output layer (the density).
There are three hidden layers, of sizes $128, 512, 128$, and a leaky ReLu activation after each layer.
T is not a parameter nor an input of the neural network, but is the maturity of the options used to train the model. Options must all have the same maturity to obtain good results.

3.2 Monte-Carlo loss

As explained in the previous chapter in 2.1, we use Monte-Carlo integration to predict the price of derivatives, in our case call and put options. We use the densities predicted by the ANN to compute the loss with which we train the neural network. Assume we have m different options in our training set. Let π_j be the price of the option in the market at time $t = T$ and $p_j(x)$ be its payoff function. Let H be the Huber loss function, which approximates a parabola centered at the origin but grows linearly away from 0. Then the loss of the neural network with respect to the m options is

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{j=0}^{m} H\left(1, \frac{L \sum_{i=0}^{N} p_j(x_i) n_\theta(x_i)}{\pi_j}\right). \tag{3.1}$$

We recall that $f(x_i) \approx 1/L$, which explains the L factor before of the sum.

3.3 Equidistant sampling

It is important that the proposal distribution outputs samples close to 0 and to L, to make sure that the network is learning the true density q also for unlikely prices.

Moreover, this is important for the pricing of deep-in-the -money and deep-out-of-the-money options. Therefore we did not generate samples independently, but we applied the following sampling procedure. We let $x'_1 = L/2N$, $x'_N = L - L/2N$ and the values in between linearly spaced (basically the numerator goes over all the N odd numbers between 1 and $2N-1$ included: $\frac{1}{2N}L, \frac{3}{2N}L, \dots \frac{2N-1}{2N}L$). Then we generate random variables $\epsilon_1, \dots, \epsilon_N$ with density Uniform($[-\frac{L}{2N}, \frac{L}{2N}]$). Finally we let $x_i = x'_i + \epsilon_i$.
We observe that the overall probability density remains the same and the samples are equidistant in expectation, because the intervals where $x_i + \epsilon_i$ falls are a partition of the interval $[0, L]$ and $\mathbb{E}[\epsilon] = 0$.

3.4 Scheduling the number of Monte-Carlo samples

To make training faster, we start with a low value of N and we increase it gradually. In the first epochs the network is still learning the general shape of the risk-neutral density, hence few samples are sufficient. We set manually the parameters $c1$ and $c2$ which control the number of epochs and the number of Monte-Carlo samples. The scheduling is shown in the following table.

Up to epoch	MC samples
100*c1	128*c2
300*c1	256*c2
350*c1	512*c2
375*c1	1024*c2
387*c1	2048*c2
393*c1	4096*c2
396*c1	8192*c2

Chapter 4

Experiments

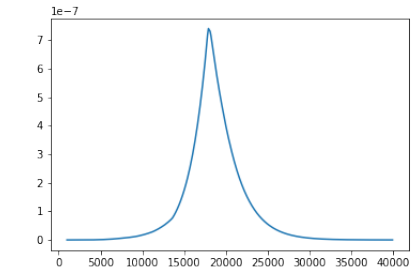
In this section we show plots regarding various experiments we have made. The notebooks used to train the model and produce the plots are available at `https://github.com/Antares01/semester_project_ml_for_finance`.

4.1 Setup

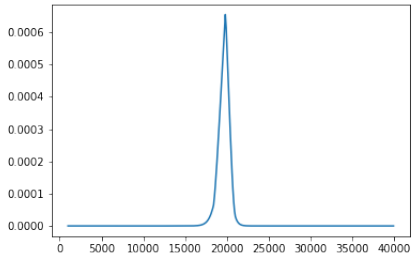
We use the JPX Tokyo Stock Exchange Prediction dataset featured in a currently running Kaggle competition. The data is publicly available at `https://www.kaggle.com/competitions/jpx-tokyo-stock-exchange-prediction/data`. To find the options data we use one needs to open the `options.csv` file in the `supplemental_files` folder. The dataset has opening and closing prices for each day of options on the NIKKEI 225 index. For training and testing we used options traded on the 2017-01-04 with maturity on the 2017-01-12. Sixty of them chosen randomly are used for training and the other fifteen are used for testing.

4.2 Vanilla model

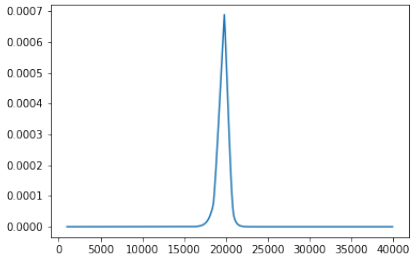
We run the vanilla model with epochs parameter $c_1 = 20$ and Monte-Carlo sampling parameter $c_2 = 2$. The loss is computed on all sixty training options without using batches. Adam is used as an optimizer with learning rate 10^{-4}. We use gradient clipping to prevent exploding gradients, which can occur in the first few epochs of training when the output of the ANN is far from being a probability density.
The model achieves a Mean Absolute Percentage Error (MAPE) of 7.6%. This means that, on average, the absolute value of the difference of the predicted price and the closing price of the option at the end of the last trading day is 7.6%.
In the figure 4.1 we show the changes in the learned risk-neutral density during training. The final density is almost identical so we don't show it here. However, it is interesting to inspect the final density in a logarithmic plot, as shown in figure 4.2. We notice a local minimum around the value of 10'000 Yen. The question whether this is a true property of the risk-neutral density or an artifact of our model is left to further investigation.
Another interesting experiment is to look at the correspondence between the increase of Monte-Carlo step and the decrease of the training loss. To this end, see figure 4.3. The first increase of Monte-Carlo steps happens at $100 * c1 = 2000$ epochs, and we effectively see a decrease in the loss at that point. All the subsequent increases of Monte-Carlo steps, including the second one at epoch $300 * c1 = 6000$ are not clearly visible. However we notice that in the last few epochs the value of



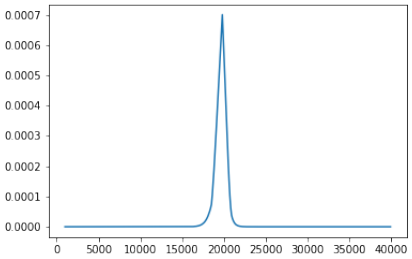
(a) After 100 * C1 epochs with 128*C2 samples



(b) After other 200 * C1 epochs with 256*C2 samples



(c) After other 50 * C1 epochs with 512*C2 samples



(d) After other 25 * C1 epochs with 1024*C2 samples

Figure 4.1: The learnt probability density function after four stages of training

the loss is oscillating less, probably indicating improved training loss stability due to the more accurate computation of the Monte-Carlo price.

4.3 Time to maturity model

This is a model which provided worse performance than the vanilla model, achieving a MAPE of 14.8% on the same dataset. We discuss briefly its idea. We give also the time to maturity T as input together with the query price x. The value of T is fed into an auxiliary MLP which is then connected to the baseline model described previously. The model has been pretrained on datasets with varying T, with the idea that the auxiliary MLP would learn to scale the probability density according to T. Then the layers of the auxiliary MLP are frozen and fine-tuning is performed on the same dataset discussed previously. The loss is also slightly modified as it can be seen in the code. In figure 4.4 we show plots for this model.
We hypothesize that the model fails due to overfitting. The density in log scale shown in figure 4.4a has some artifacts that might be due to overfitting the density function to the options in the training set. Moreover, we see that the loss achieves much lower values than in the previous model, which corroborates our supposition.

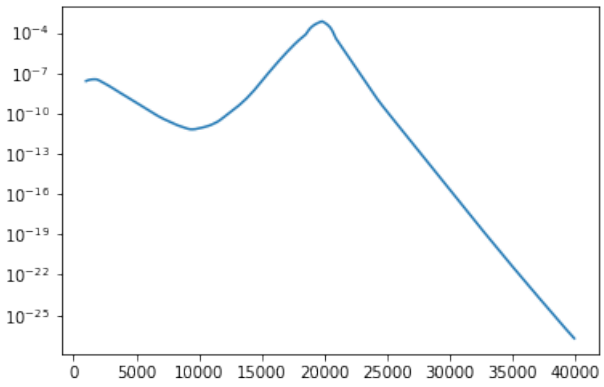


Figure 4.2: Final density learned by the neural network with the y axis in the log scale.

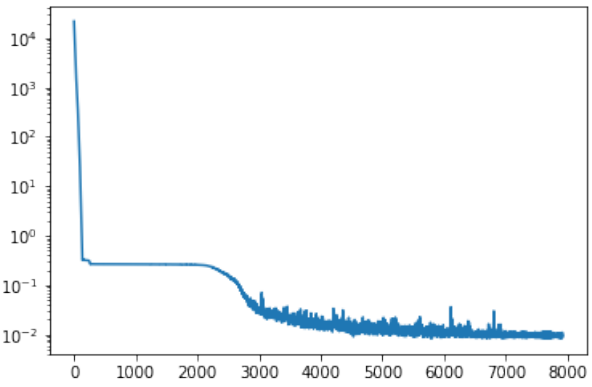
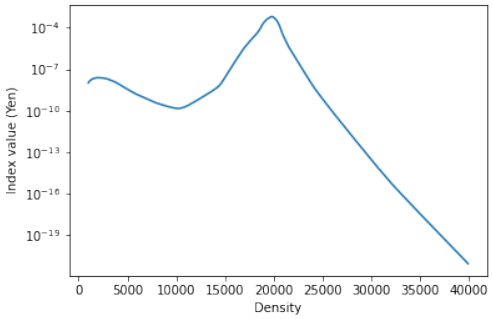
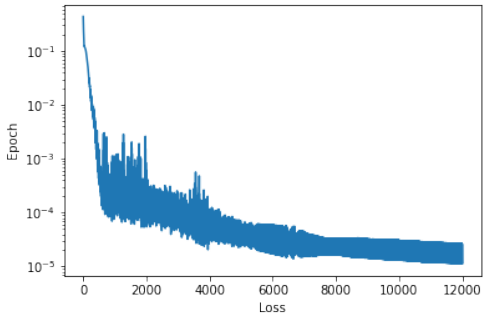


Figure 4.3: Loss value with epochs on the x axis. y axis in the log scale.



(a) Density function. y axis in the log scale.



(b) Loss history. y axis in the log scale.

Figure 4.4: Training plots for the time to maturity model

Bibliography

[1] J. Ruf and W. Wang, "Neural networks for option pricing and hedging: a literature review," *Journal of Computational Finance*, pp. 1–46, Nov. 2019.

[2] C. Schittenkopf and G. Dorffner, "Risk-neutral density extraction from option prices: improved pricing with mixture density networks," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 716–725, 2001.

[3] W.-N. Lai, "Comparison of methods to estimate option implied risk-neutral densities," *Quantitative Finance*, vol. 14, no. 10, pp. 1839–1855, 2014.