

Swaption Pricing

A neural network approach

Nathan Simonis

Acknowledgment

Many thanks to Frédéric Bossens, Nicolas Cosme and Thomas Wuilbaut for the help and insights throughout the project.

TABLE OF CONTENTS

- 1 **Motivation**
- 2 **Methodology**
- 3 **Neural Network**
- 4 **Data**
- 5 **Training**
- 6 **Results**
- 7 **Conclusion**

- A classic problem in financial mathematics is the calibration of an asset pricing model to observed market prices.
- This is done by finding the parameters that make the model and market prices similar.
- In practice one has to compute the model prices for several calibration instruments and this, for several iterations of the minimization algorithm.
- Depending on the model used, this procedure can be very time consuming as there may not be an analytical formula. This renders a model unusable in practice because it cannot be calibrated.
- In order to reduce the calibration time, we replace the model computations by an estimator that is much faster. The approximator is a deep neural network.
- The use of a fast estimator widens the range of acceptable models since we are no longer limited by an analytical tractability constraint.

- In an offline procedure, for a given pricing model (HW1F), we compute 100.000 volatility surfaces implied by the model for a given yield curve and model parameters. We then "learn" the relationship between the yield curve, model parameters and the volatility surface.
- While the approach is generalizable to other models we use the hw1f model in order to generate data relatively quickly. In the case of a model where we have no analytical formula, it would take at least a few days to generate the data.

- We consider a dataset \mathcal{D} of examples $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ where \mathbf{x}_n is the n^{th} input vector of dimension D . This vector contains the yield curve, the hw1f volatility parameter and the swap tenor. The output vector \mathbf{y}_n of dimension J is the n^{th} grid of implied volatilities for 9 strikes and 8 option maturities. The number of pairs N is 100.000.
- The goal of supervised learning is to use this dataset to produce a model that takes a new vector \mathbf{x}' as input and returns a prediction of the output vector \mathbf{y}' based on the relationship that exists between the two.
- We need to find a function f that approximates the output "well enough" given new inputs:

$$\mathbf{y}_n \approx f(\mathbf{x}_n), \forall n$$

- As predictor f we choose to use a fully connected feedforward neural network with 4 hidden layers (L) and 32 nodes (K) per layer.
- The output at the node k in layer l is given by:

$$x_k^{(l)} = \phi \left(\sum_i w_{i,k}^{(l)} x_i^{(l-1)} + b_k^{(l)} \right)$$

where $w_{i,k}^{(l)}$ is the edge from node i in layer $l - 1$ to the node k in layer l .

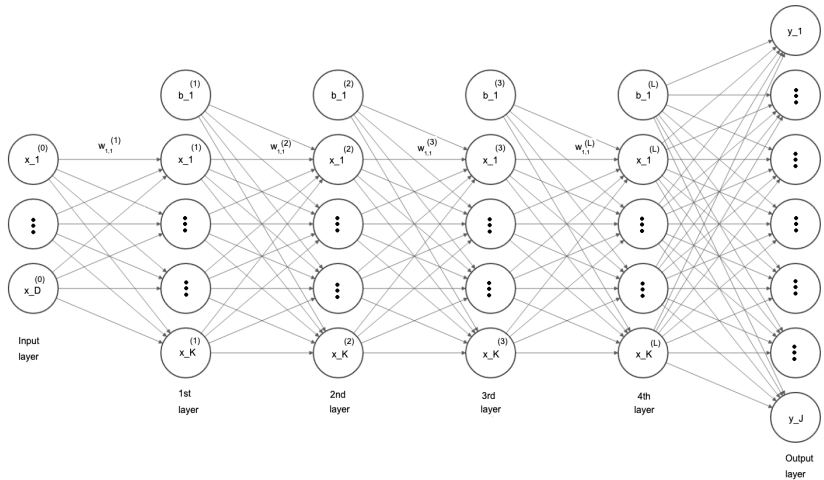
- The function ϕ is the activation function:

$$\phi(z) = \begin{cases} z & \text{if } z > 0, \\ \alpha(e^z - 1) & \text{otherwise.} \end{cases}$$

For this application we use the ELU activation function with $\alpha = 1$.

- This structure can approximate any continuous function arbitrarily closely as long as we allow K to be large enough.

Neural Network (2)



- Let $f(\mathbf{x})$ be the output of the neural network. We consider the Root Mean Square Error (RMSE) loss function:

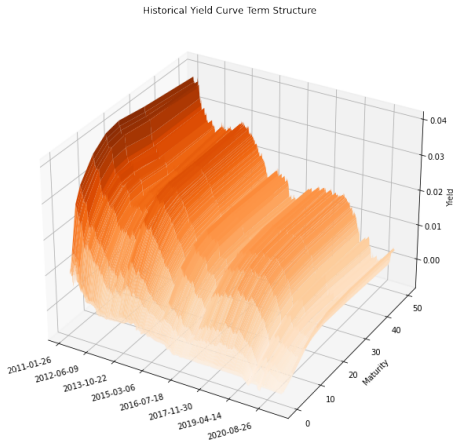
$$\mathcal{L} = \sqrt{\frac{1}{N} \sum_{n=1}^N [\mathbf{y}_n - f(\mathbf{x}_n)]^2}$$

as the measure of how well the NN approximates the grid of implied volatilities of the hw1f model.

- We choose the parameters of the NN w and b in order to minimize the loss function. This is done with the Adam algorithm that is an extension of Stochastic Gradient Descent.
- The neural network is implemented with the Keras library in Python.

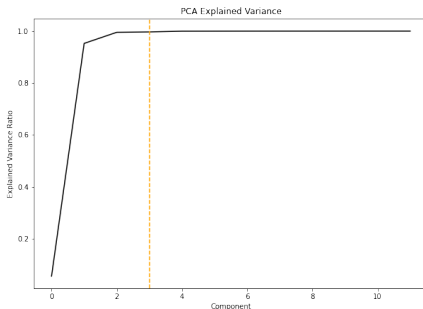
Generating Data: Input Vector

- We use 10 years of historical ESTR curves downloaded from Bloomberg. We sample these historical curves with replacement to reach 100.000 examples.



Generating Data: Input Vector (2)

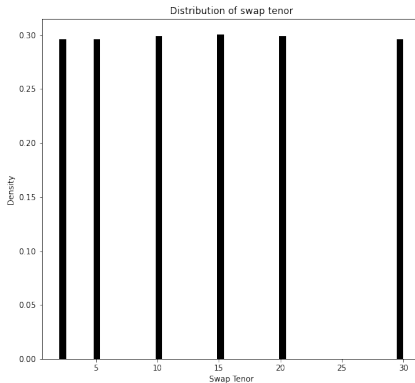
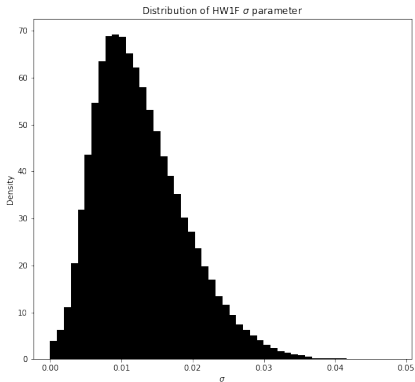
- In order to reduce the dimensionality of the problem and eliminate redundant variables from the curve, we perform a principal component analysis.



- We retain 99.9% of the variance of the curve by only considering the first three principal components. We only include these in the input vector.

Generating Data: Input Vector (3)

- The hw1f volatility parameter is sampled from a normal skew distribution with a mean of 0.5% and a variance of 1%.
- The tenor of the swap is sampled uniformly over discrete values ranging from 2 to 30 years.



- For each input vector we generate a grid of implied volatilities for 9 strikes and 8 option maturities.
- $\Delta \text{ strike} = \{-0.02, -0.015, -0.01, -0.005, 0, 0.005, 0.01, 0.015, 0.02\}$. This yields: $\text{strike} = \text{swapRate} + \Delta \text{strike}$.
- Option maturities = $\{180/360, 1, 2, 5, 10, 15, 20, 30\}$

- The input variables are normalized between -1 and 1:

$$\tilde{x} = 2 \times \frac{x - \min(x)}{\max(x) - \min(x)} - 1$$

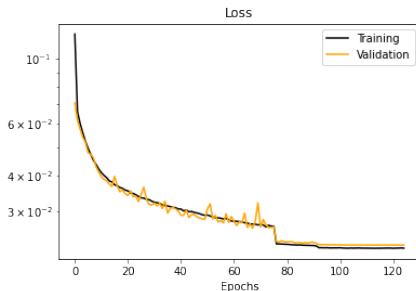
- The output variables are standardized:

$$\tilde{y} = \frac{y - \text{mean}(y)}{\text{std}(y)}$$

- In summary, we have 100.000 samples of a pairing $(\tilde{\mathbf{x}}_n, \tilde{\mathbf{y}}_n)$ where the input vector has 5 dimensions (first 3 principal components of the yield curve, hw1f vol, swap tenor) and the output vector has 72 dimensions (combinations of strikes and option maturities).

Training the Neural Network

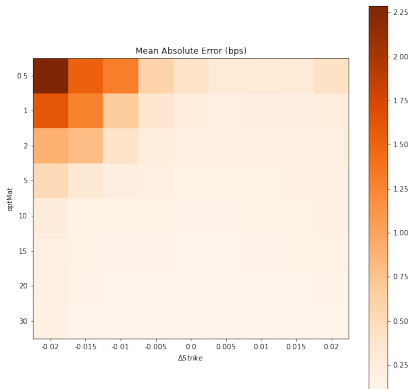
- To find the weights, we split the data into 2 parts, 64.000 samples are used to train the network and 16.000 samples are used to track the validation error while training. We stop training the NN when the validation error stops improving.
- The final 20.000 samples are held-out and only used to report the performance of the NN.



- We converge after 125 epochs of 32 batches. The training and validation error are very similar. On the hold-out dataset, we obtain scores of 0.00013 and 0.99915 for the $RMSE$ and R^2 , respectively.

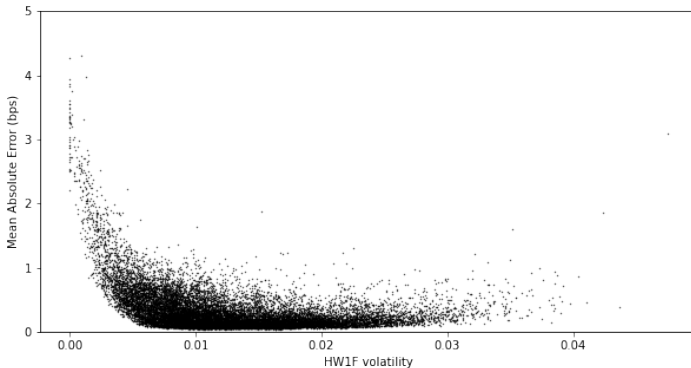
Results - Out of sample

- It appears that the NN commits on average higher absolute errors for options that are deeply in the money and have a short maturity.
- For the other combinations of strikes and maturities of the option, the absolute error is less than 0.5 bps.



Results (2)

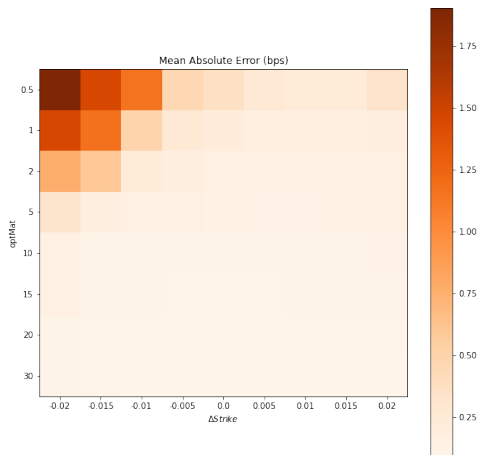
- We plot the mean absolute error of the NN for different values of hw1f volatilities.



- In general, the NN makes smaller errors for hw1f volatilities between 0.005 and 0.02. In practice, these values are much more realistic.

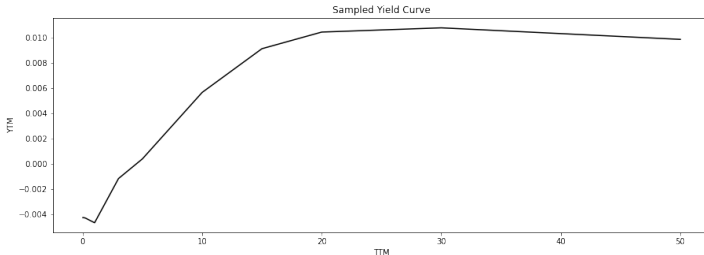
Results (3)

- If we only consider values of hw1f volatilities that are within this range in the testing set, we observe smaller mean absolute errors.



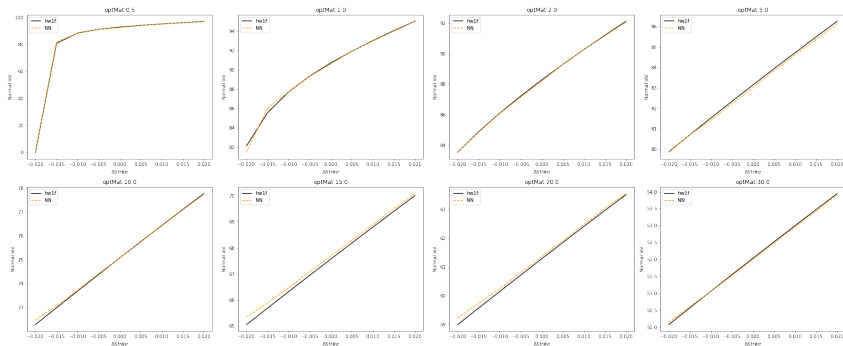
Result (4)

- We randomly select an input sample from our test data and make a prediction of the volatility surface with the NN.
- The sample in question has a hw1f volatility of 0.015 and a swap tenor of 20 years. The sample curve is shown below:



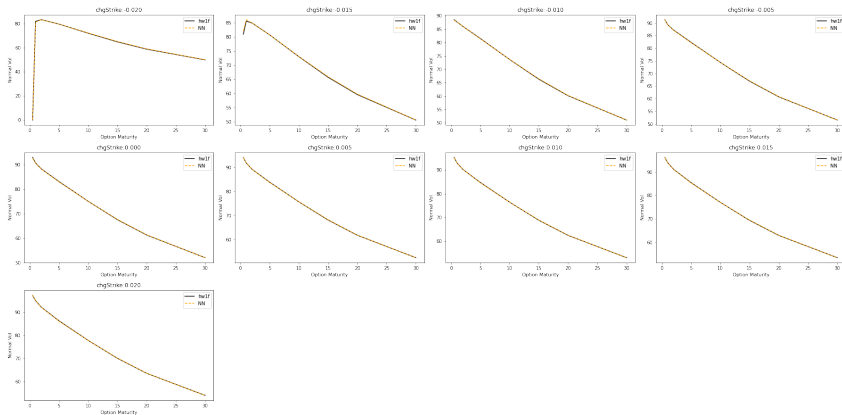
Result (5)

- For the sample, we fix the option's maturity and observe the smile implied by the NN and hw1f.



Result (5)

- For the sample, we fix the strike and observe the time dependency for the NN and hw1f



- With the NN, we are able to calculate 100.000 implied volatility surfaces in $1.47 \text{ s} \pm 192 \text{ ms}$.
- For more complex models, the NN calculation time will be the same. Compared to pricing with a Monte Carlo simulation which would take at least several days, we get an acceleration of several orders of magnitude.
- The presented architecture is very flexible and can be adapted to support many other products and models. This would allow the use of more realistic models that are currently not feasible due to the computational cost.
- The price to pay for this time saving is reduced accuracy. However, the error seems very reasonable for a majority of strike and maturity combinations.

- Try this approach on models that contain more parameters. We will thus see if the model remains robust when we increase the dimensionality.
- Learn a model that needs to be computed through Monte Carlo and therefore contains noise in the output.
- See how the approximation error is reflected in the model parameters during calibration.

- Horvath, Blanka and Muguruza, Aitor and Tomas, Mehdi, Deep Learning Volatility (January 24, 2019). Available at SSRN: <https://ssrn.com/abstract=3322085>
- A Itkin, Deep learning calibration of option pricing models: some pitfalls and solutions (June 8, 2019). Available at arXiv: <https://arxiv.org/abs/1906.03507>