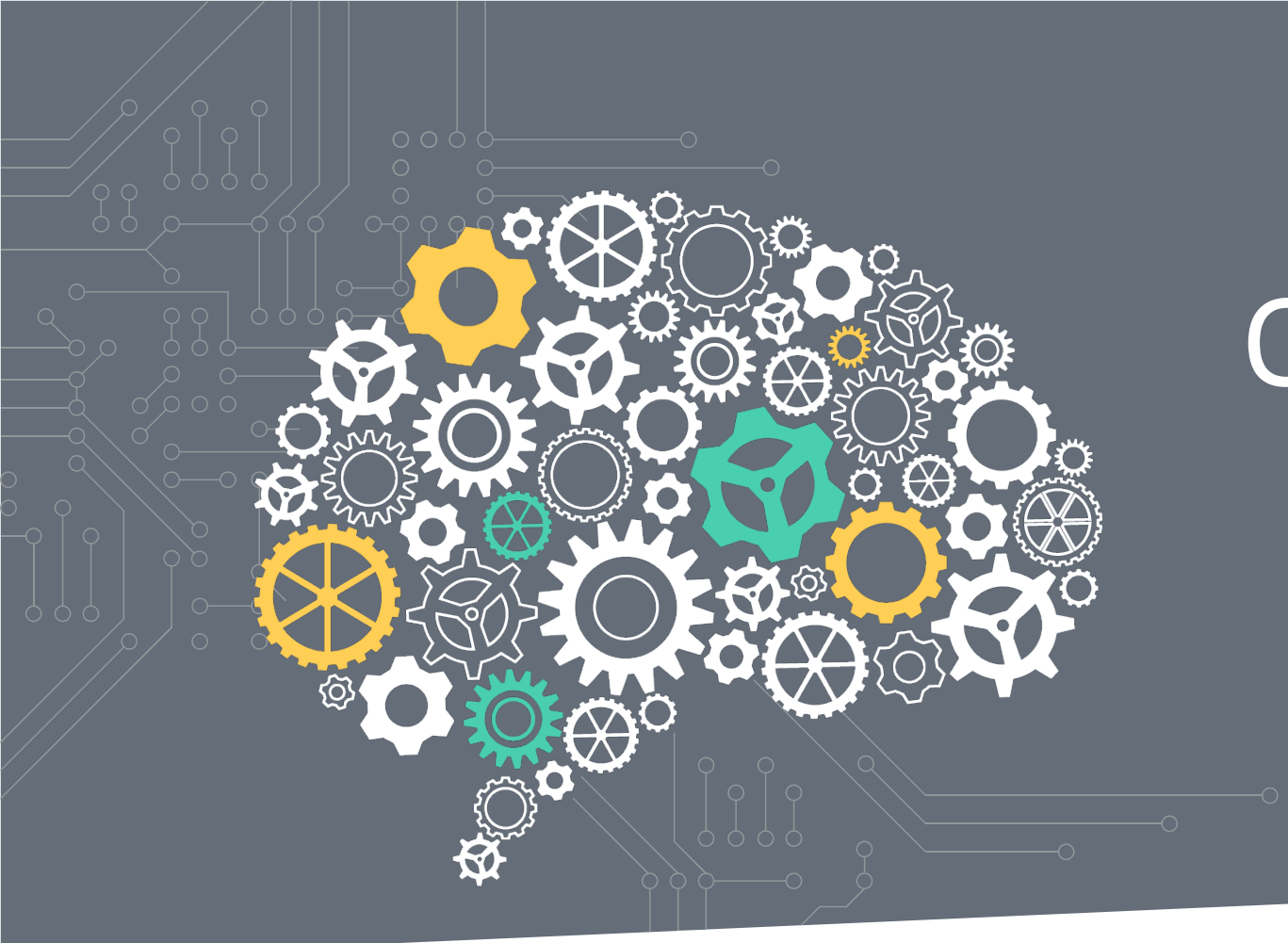


Option Pricing Using Neural Networks

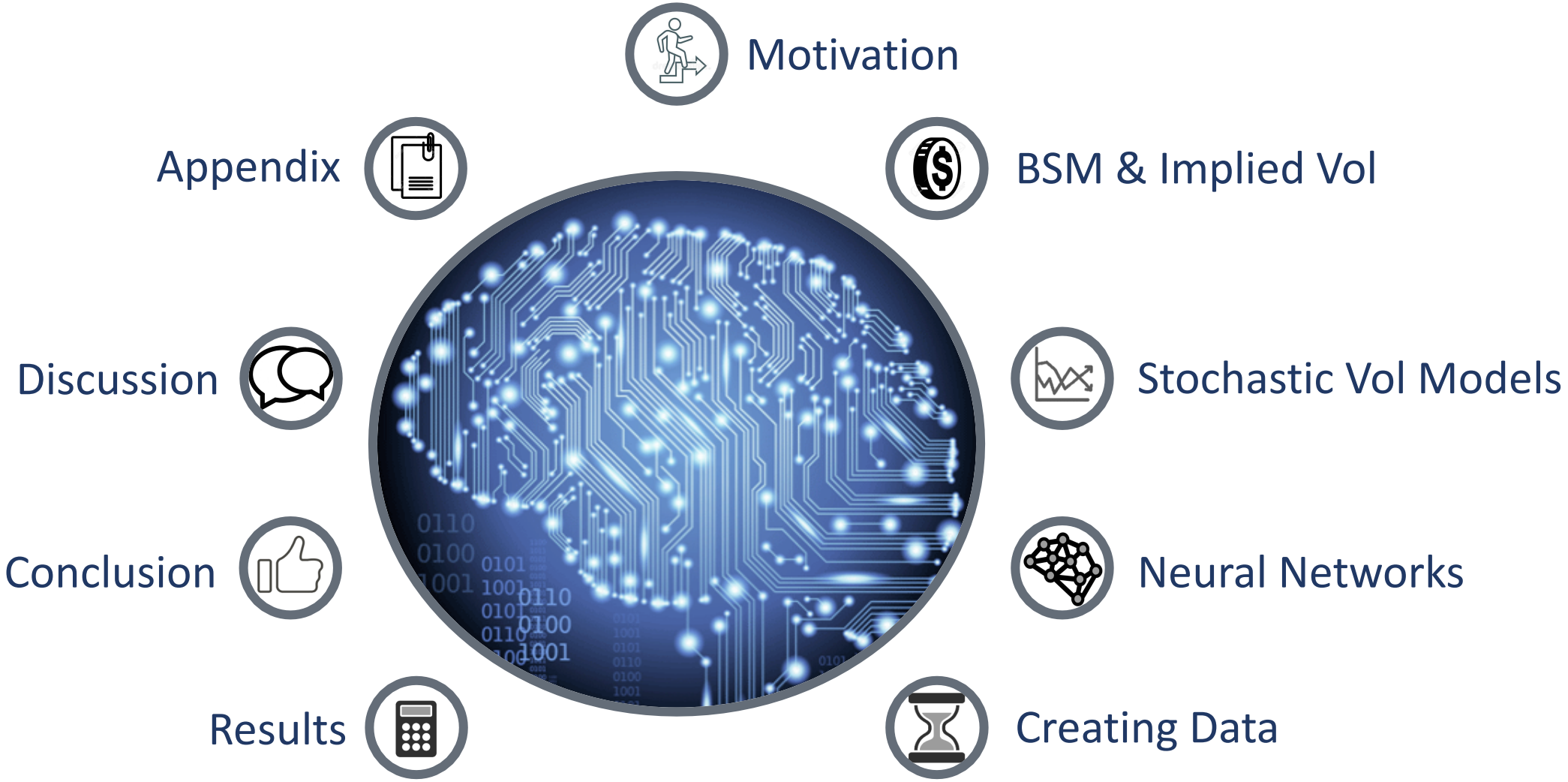
University of Copenhagen

Master Thesis in Mathematics-Economics

Supervisors:
Rolf Poulsen
Kenneth Nielsen

Censor:
David Pedersen

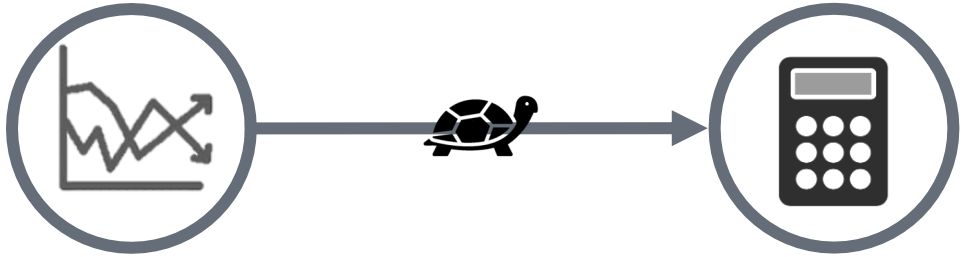
Contents

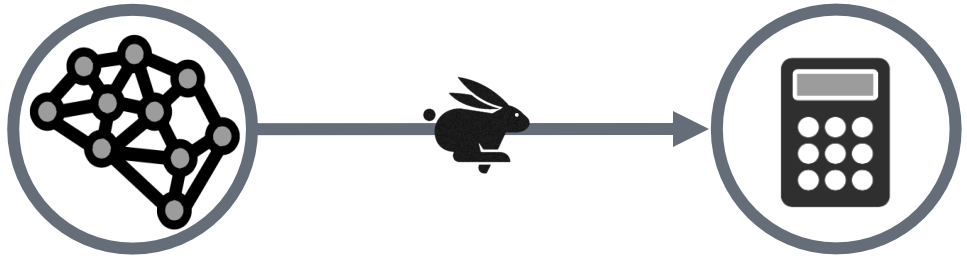

Motivation
BSM & Implied Vol
Stochastic Vol Models
Neural Networks
Creating Data
Results
Conclusion
Discussion
Appendix

Motivation

Motivation

- Due to the lack of computational efficiency, complex pricing models have been ignored in many aspects of finance
- Machine learning (ML) and neural networks (NN) are becoming a standard piece of kit
- NNs have been proposed to
  - speed up slow function calculations and
  - accurately approximate complex and multidimensional models
- ML and NNs are however not without their shortcomings and still infrequent in banking (primarily due to legacy systems)
- Our goal is to test whether or not the above two statements hold for the, in practice, most used stochastic volatility models – Heston and SABR





BSM & Implied Volatility

BSM & Implied Volatility

BSM Option Pricing Formula

- **Process:**

$$dS_t = \mu S_t dt + \sigma S_t dW$$

- **Model parameters:** Drift μ, Stock S, volatility σ, GBM W, Strike K, Interest rate r, Time t, Maturity T, Call C
- **Vanilla Call Price**

$$C_t = S_t N(d_1) - Ke^{-r(T-t)} N(d_2)$$

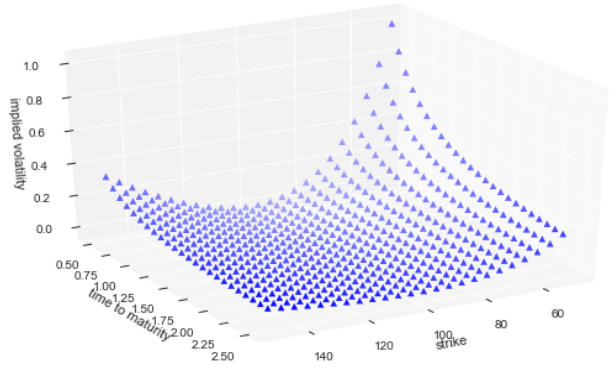
$$d_1 = \frac{\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} = \frac{\ln\left(\frac{S_t}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

- **Problem:** Assumes constant volatility (σ) when it is in fact moving

Implied Volatility (IV)

- Remove the assumption of constant volatility by introducing IV into BSM
- Implied volatility captures the market's view of the likelihood of changes in a given security's price
- $\sigma^* = BSM^{-1}(C^{mkt}; S_t, K, T, r)$
- **Brent Method:** Iterative numerical technique to solve an options IV
  - "Fast"
  - Robust



Stochastic Volatility Models

Stochastic Volatility Models

The Heston Formula

- **Process:**

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$$
$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v}_t dW_t^v$$
$$\langle dW_t^S, dW_t^v \rangle = \rho dt$$

- **Model parameters:** Drift μ, Stock S, Volatility $\sqrt{v}$, Mean reversion speed κ, Mean reversion level θ, Correlation ρ, Vol of Vol σ, Variance v
- **Lewis Lipton call price:**

$$C(S,K,\tau)$$
$$= e^{-r_d\tau}S - \frac{e^{-r\tau}K}{2\pi}\int_{\infty}^{-\infty} \frac{e^{\left(i\varphi+\frac{1}{2}\right)X+\alpha(\varphi)-\left(\varphi^2+\frac{1}{4}\right)\beta(\varphi)v}}{\varphi^2+\frac{1}{4}}d\varphi$$

- Use Brent's method to find the implied volatility

The SABR Formula

- **Process:**

$$dF_t = \sigma_t F_t^{\beta} dW_t^F$$
$$d\sigma_t = \xi\sigma_t dW_t^{\sigma}$$
$$\langle dW_t^F, dW_t^{\sigma} \rangle = \rho dt$$

- **Model parameters:** Forward price F, Alpha/volatility σ, Vol of vol ξ, CEV β, Correlation ρ
- **SABR Approximation:**

$$\sigma_{Approx}(K,F)$$
$$= \frac{\sigma_0}{q(K,F)}\left(\frac{z}{x(z)}\right)\left[1 + \left[\frac{(1-\beta)^2}{24}\frac{\sigma_0^2}{(FK)^{1-\beta}} + \frac{1}{4}\frac{\rho\beta\xi\sigma_0}{(FK)^{\frac{1-\beta}{2}}} + \frac{2-3\rho^2}{24}\xi^2\right]T + \cdots\right]$$

Neural Networks

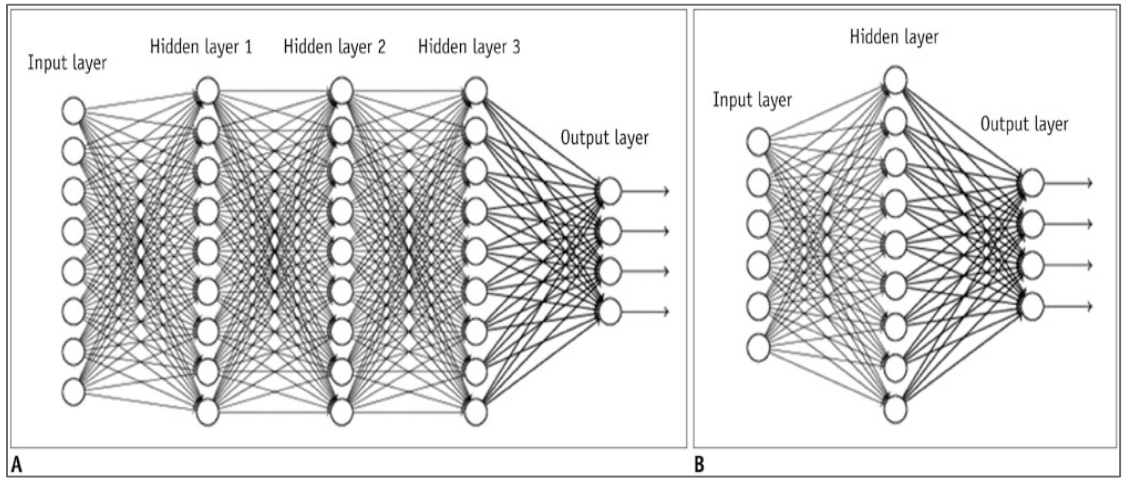
Architecture of Feed-Forward Neural Networks (FFNN)

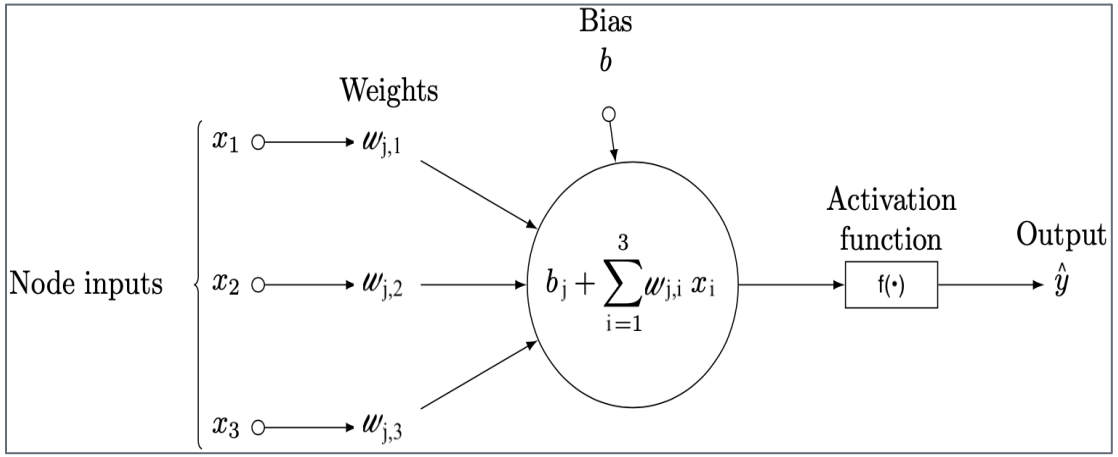
Example (right):

- Feeding inputs x_i
- Each input parameter has a corresponding weight $w_{j,i}$ and bias b_j used for the first hidden layer and its one node.
- The value of the node is

$$f\left(b_j + \sum_{i=1}^{3} w_{j,i}\, x_i\right)$$

- f is the non-linear activation function and $\hat{y}$ is the predicted value





General:

$$a^{[0]} = z^{[0]} = X$$
$$a^{[l]} = f^{[l]}\left(z^{[l]}\right) = f^{[l]}\left(b^{[l]} + w^{[l]} a^{[l-1]}\right)$$
$$\hat{y} = a^{[L]}$$

- $w^{[l]}$ matrices of weights, $b^{[l]}$ vectors of biases, $f^{[l]}$ activation functions mapping from $\mathbb{R}$ to $\mathbb{R}$
- Image to the left shows a deep neural network (DNN) (A) and artificial neural network (ANN) (B). The circles are known as neurons
- The above picture shows an example of just one neuron

General Concepts

The Universal Approximation Theorem

- In 1989, Cybenko stated that a FFNN with a single hidden layer should in theory be enough to approximate any continuous function
- In practice, multiple layers are preferred
- As in McGhee (2018) mild assumptions on the activation function can be made

Train/Validation/Test-Sets

- **Training Dataset:** The sample of data used to fit the model
- **Validation Dataset:** The sample of data used to evaluate the model throughout training
- **Test Dataset:** The sample of data not yet seen by the model (used for the final evaluation). Also known as out-of-sample (OOS) data

The Bias/Variance Tradeoff

$$Err(x) = Bias^2 + Variance + Irreducible\ Error$$

- **Bias:** Is the part of the generalization error, which is due to wrong assumptions. High bias equals under-fitting
- **Variance:** Is the high sensitivity to small variations in the training data. High variance equals over-fitting
- **Irreducible Error:** Is due to the noisiness of the data itself

Data Scaling

- Unscaled input variables can result in a slow or unstable learning process, whereas unscaled target variables can result in exploding gradients
- Larger values influence the gradient more resulting in prioritization of certain inputs at each iteration
- Normalize data:

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

Activation Functions (AF)

- One of the most important concepts of NN is the activation function
- The AF enables a NN to capture non-linearities in the data
- Linear AFs result in linear regression only. Will also cause problems when doing gradient descent (the derivative of the identity is constant)
- The optimal AF is simple as it needs to be computationally efficient

- **Sigmoid activation function:**

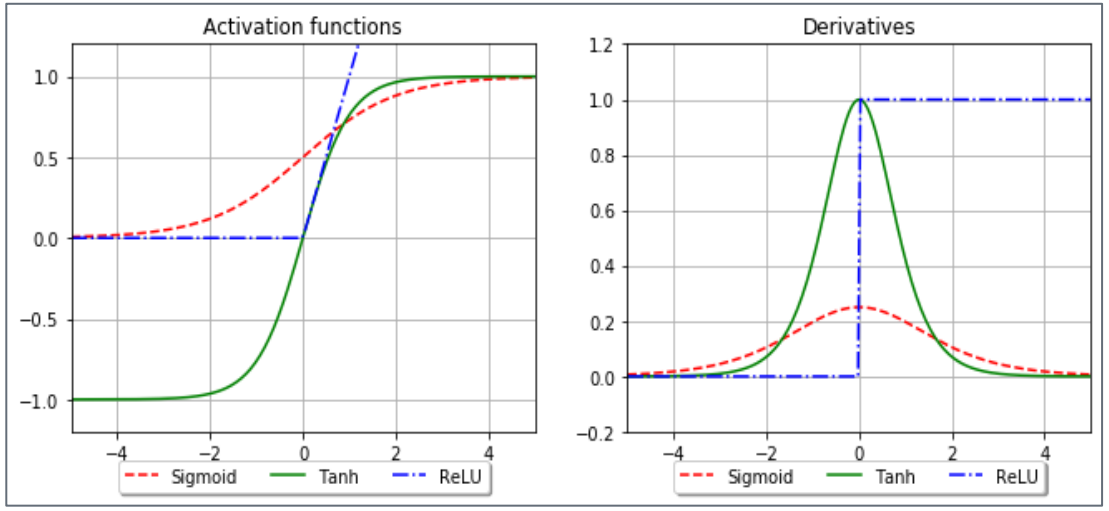
$$f(z) = \frac{1}{1+e^{-z}}, \qquad f'(z) = f(z)\big(1-f(z)\big),$$
$$f(z) \in (0,1)$$

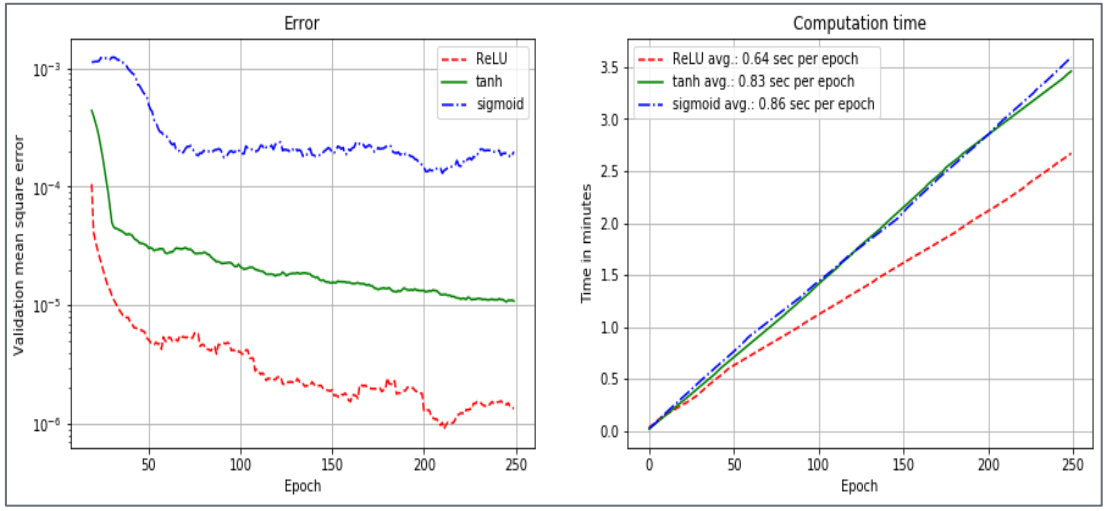
- **Tanh activation function:**

$$f(z) = \left(\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}\right), \qquad f'(z) = 1-f(z)^2,$$
$$f(z) \in (-1,1)$$

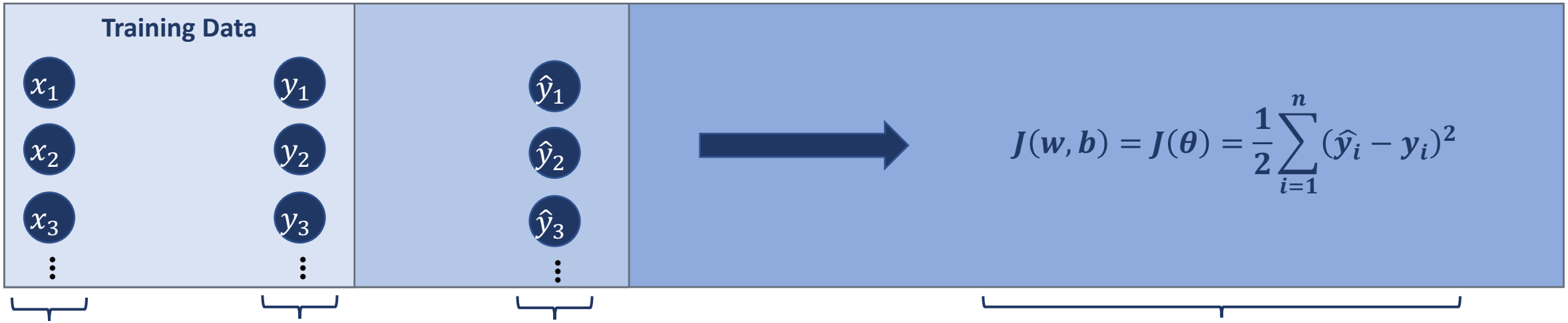
- **ReLU activation function:**

$$f(z) = \max(0,z), \qquad f(z) = \begin{cases} 0, z < 0 \\ 1, z \geq 0' \end{cases} \qquad f(z) \in [0,\infty)$$





Stochastic Gradient Descent



Sample inputs in layer [0], indexed by a set n

Corresponding target outputs / labels

Corresponding real outputs, layer [L]

Typical **Loss Function**, to be minimized via choice of (W, b): Mean Squared Error (MSE)

Vanilla Batch Gradient Descent:

$$\theta \longmapsto \theta - \eta \nabla_\theta J(\theta)$$

where the **Learning Rate** η can change from iteration to iteration. The gradient vector containing the partial derivatives of the objective function is computed via adjoint algorithmic differentiation (ADD), more commonly known as **Backpropagation**

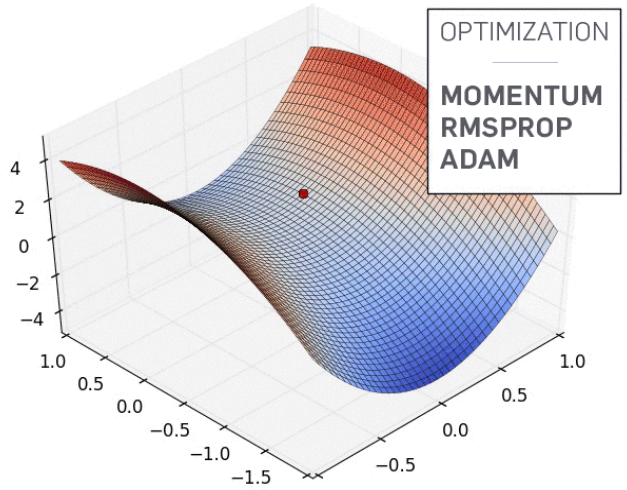
Stochastic Mini-Batch Gradient Descent:

$$\theta \longmapsto \theta - \eta \nabla_\theta J(\theta; x_{i;i+K}, y_{i;i+K})$$

where at each iteration a new **mini-batch** of $K \subseteq n$ is chosen randomly, where again the gradient is computed via backpropagation

Gradient Descent Optimization Algorithms

- Gradient descent has issues navigating ravines (pathological curvatures) i.e. areas where the surface curves more steeply in one dimension than in another
- One could use a slow learning rate to deal with a ravine but might result in too slow convergence to the minima
- We want something that can get us slowly into the flat region at the bottom of a ravine first, and then accelerate in the direction of minima. Second derivatives can help us do that



Adam Optimizer:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{v}_{t,i} + \epsilon}} \hat{m}_{t,i}$$

$$\hat{m}_{t,i} = \frac{\beta_1 m_{t-1,i} + (1-\beta_1) g_{t,i}}{1-\beta_1^t}, \qquad \hat{v}_{t,i} = \frac{\beta_2 v_{t-1,i} + (1-\beta_2)(g_{t,i})^2}{1-\beta_2^t}$$

where ϵ prevents division by zero, $m_{t,i}$ estimate of the first moment (mean), $v_{t,i}$ estimate of the second moment (variance), $g_{t,i}$ gradient of the current mini-batch, β_1, β_2 fixed hyper-parameters

Adam is based on the following idea:

- Since K is stochastic, so is $J_K(\theta)$. Since all mini-batches are equally likely, we have $\nabla_\theta J_n(\theta) = \mathbb{E}[\nabla_\theta J_K(\theta)]$. Hence, at each epoch, use a weighted average of previous batch gradients
- If the second moment $\mathbb{E}[(\nabla_\theta J_K(\theta))^2]$ is high, reduce the learning rate via re-scaling
- Estimate the above moments via exponential averages over their values at previous epochs

Creating Data

Creating The Data

Heston Data

SABR Data

- Generate the IV surface at each feed-forward pass by outputting 10 IVs at every calculation
- Generate 290-thousand input-output pairs for training/validating and 10-thousand for testing

$$x = (\tau, r, \sigma, \rho, v_0, \theta, \kappa)$$
$$y = (\Sigma_1, \dots, \Sigma_{10})$$

- 200-thousand input-output pairs from sobol numbers and 100-thousand at random

	Parameters	Ranges
Inputs	Time to maturity: τ	[0.11,1.10] (year)
	Risk-free rate: r	[0.00,0.10]
	Volatility of volatility: σ	[0.05,0.50]
	Correlation: ρ	[-0.90,0.00]
	Initial variance: v_0	[0.05,0.50]
	Long-term mean: θ	[0.05,0.50]
	Mean reversion speed: κ	[0.10,1.90]
	Strikes (moneyness): $K_1, \dots, K_{10}$	[0.8,1.25] (even intervals)
Outputs	Implied volatilities: $\Sigma_1, \dots, \Sigma_{10}$	[0.14,0.81]

$$x = (T, \sigma_0, \xi, \rho, K_1, \dots, K_{10})$$
$$y = (\sigma_1, \dots, \sigma_{10})$$

- Same data as McGhee 2018 (200-thousand random and 100-thousand specifically selected)

	Parameters	Ranges
Inputs	Maturity: T	[0.003,1.00] (year)
	Initial volatility: σ_0	[0.05,0.50]
	Volatility of volatility: ξ	[0.01,11.03]
	Correlation: ρ	[-0.90,0.90]
	Strikes (moneyness): $K_1, \dots, K_{10}$	[0.21,12.77]
Outputs	Implied volatilities: $\sigma_1, \dots, \sigma_{10}$	[0.03,1.11]

Results

Learning The Heston- And SABR-Models: Setup

Technology

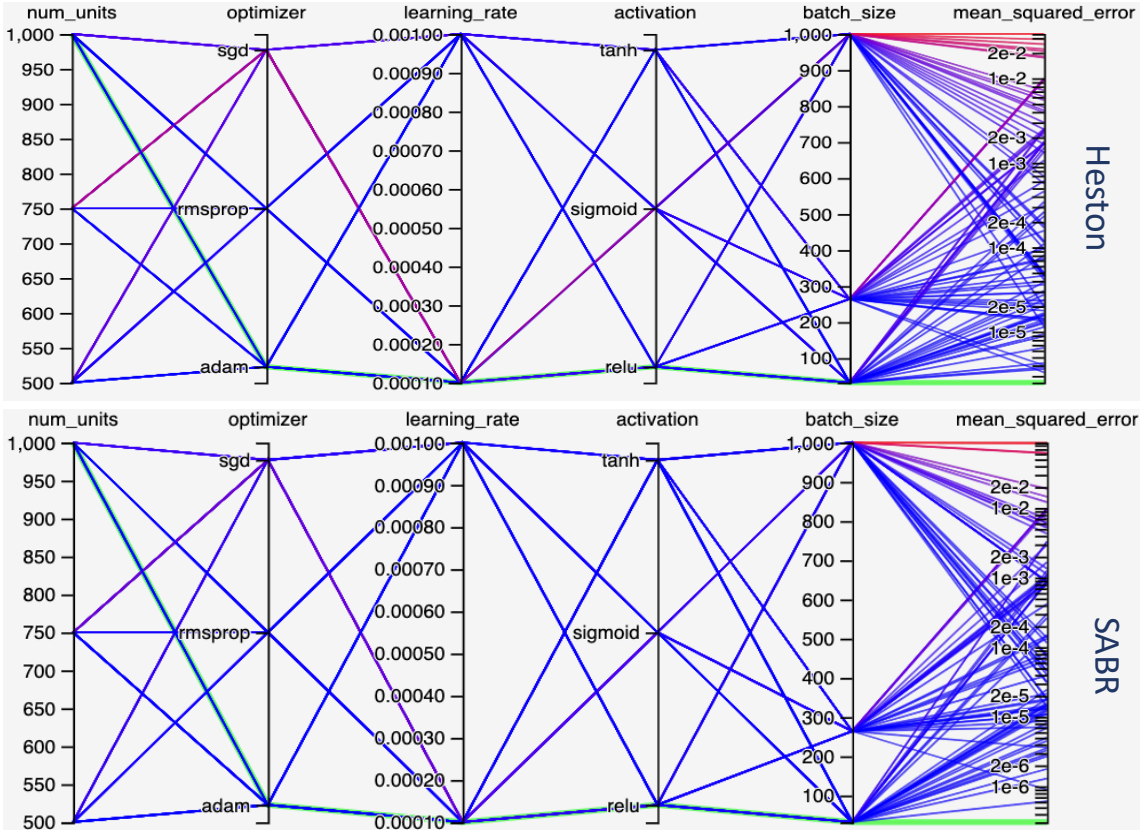
python TensorFlow Keras

Choice of Hyper-parameters

- Hidden layers: 1
- Hidden layer size: 1000
- Number of inputs: Heston=7, SABR=14
- Number of outputs: 10
- Activation function: ReLU
- Weight initializer: Glorot
- Bias initializer: Zeros
- Gradient descent method: Mini-batch
- Batch-size: 32
- Optimization algorithm: Adam
- Learning rate (η): 0.0001
- 1st moment estimates (β_1): 0.9
- 2nd moment estimates (β_2): 0.999

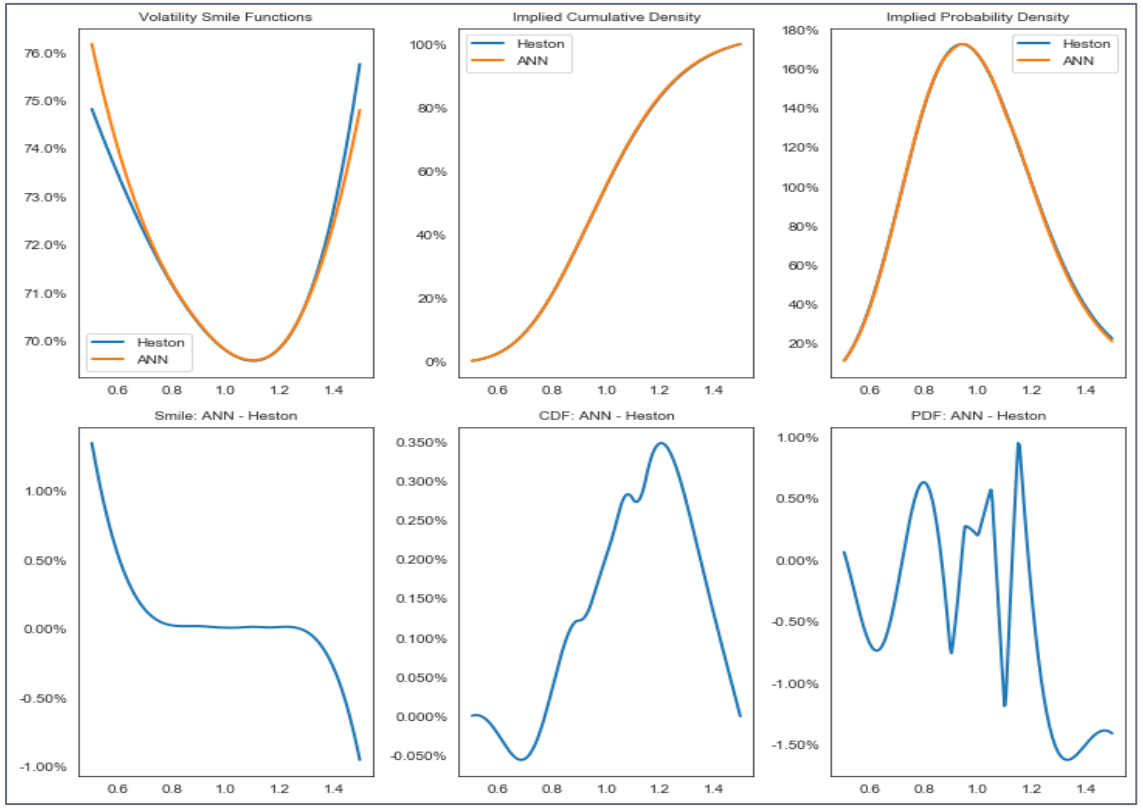
Conditions

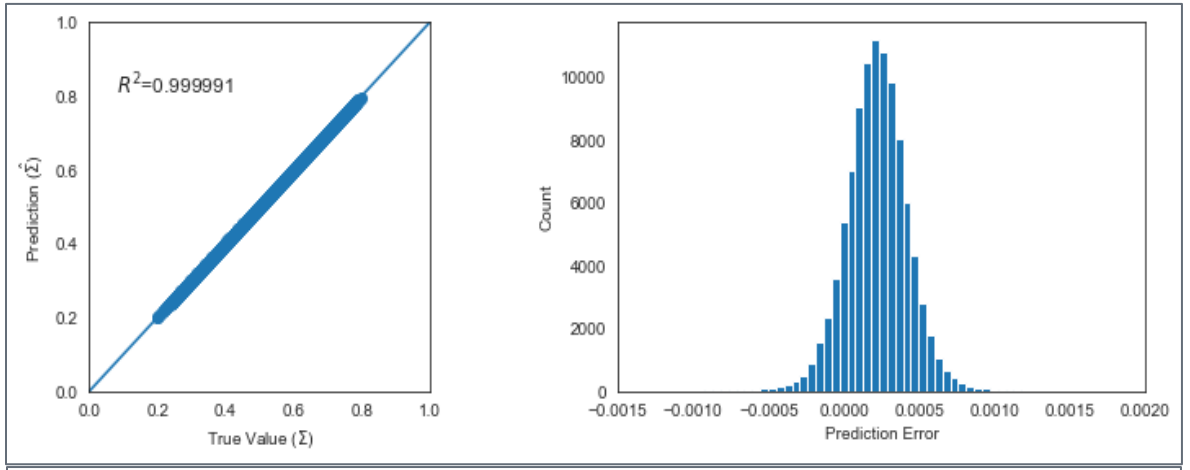
- *Condition 1*: If the MSE$\leq 10^{-10}$ training stops
- *Condition 2*: If training time exceeds 10 hours, training stops
- *Condition 3*: Early stopping after 100 epochs



Test Results: Heston ANN

Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	5.15e-08	6.65e-08	9.33e-08	709	4h 36min 52s



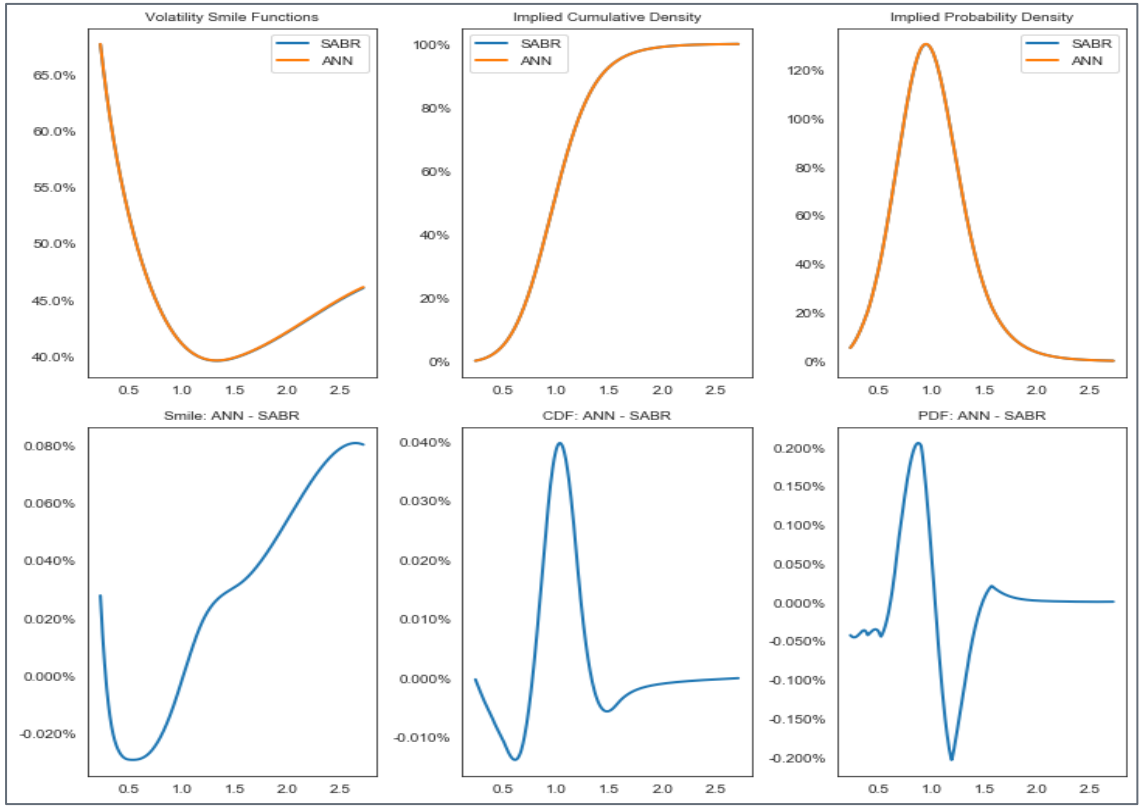


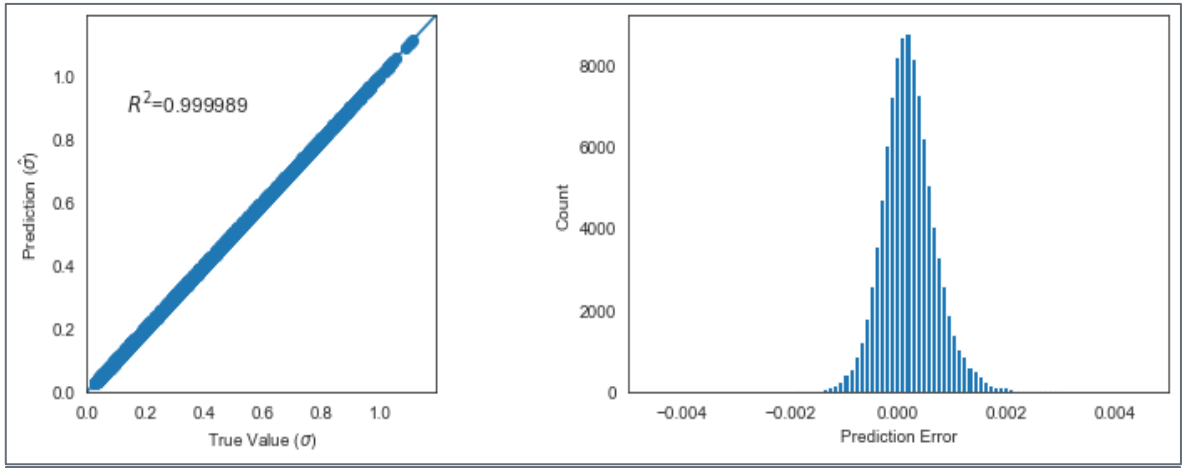
- High “on-line” accuracy and fast “off-line” training
- Error with outliers [-0.55%,0.50%] and without outliers [-0.04%,0.07%]
- Most errors are at the edge of the domain space
- Cubic spline interpolation the ANN follows the true values well
- Cubic spline extrapolation not so much
- Smooth first and second derivatives (CDF and PDF respectively)
- 112+ times faster than numeric solution

Method	Number of smiles	Time	Time per smile	Time per volatility
Heston	1,000	4,680.44ms	4.68e-04s	4.68e-05s
ANN	10,000	41.7ms	4.17e-06s	4.17e-07s

Test Results: SABR ANN

Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	4.20e-07	3.75e-07	3.47e-07	691	4h 52min 28s





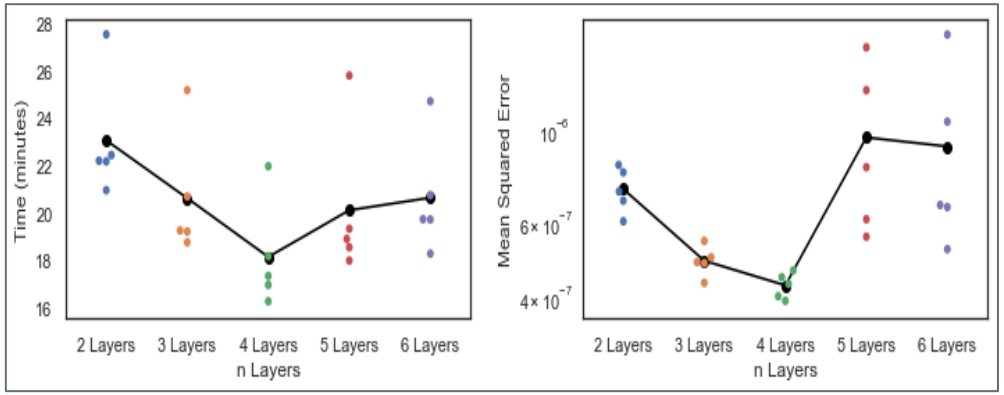
- High “on-line” accuracy and fast “off-line” training (worse than Heston)
- Error with outliers [-0.89%,1.01%] and without outliers [-0.50%,0.50%]
- Most errors are at the edge of the domain space
- Cubic spline interpolation the ANN follows the true values well
- No need for Cubic spline extrapolation (strikes are dynamicly chosen)
- Smooth first and second derivatives (CDF and PDF respectively)
- 1.8 times slower than analytic solution

Method	Number of smiles	Time	Time per smile	Time per volatility
SABR	1,000	23.2ms	2.32e-06s	2.32e-07s
ANN	10,000	41.2ms	4.12e-06s	4.12e-07s

Test Results: Heston DNN (1/3)

- A single layer might theoretically suffice to arbitrarily approximate any continous function but DNNs are used in practice
- **Theorem (Eldan et al.):** *There exists a simple (approximately radial) function on $\mathbb{R}^d$, expressible by a small 3-layer feedforward neural network, which cannot be approximated by any 2-layer network, to more than a certain constant accuracy, unless its width is exponential in dimension*
- Avoid extrapolation by increasing the strike range
- Train on a wider input range and test on a narrower input range
- Increase number of input-output pairs

	Parameters	Training/Wide Ranges	Test/Narrow Ranges
Inputs	Time to maturity: τ	[0.11,1.10] (year)	[0.11,1.10] (year)
	Risk-free rate: r	[0.00,0.10]	[0.00,0.10]
	Volatility of volatility: σ	[0.05,0.50]	[0.05,0.50]
	Correlation: ρ	[-0.90,0.00]	[-0.90,0.00]
	Initial variance: v_0	[0.01,0.50]	[0.05,0.50]
	Long-term mean: θ	[0.05,0.50]	[0.05,0.50]
	Mean reversion speed: κ	[0.10,2.00]	[0.10,1.90]
	Strikes (moneyness): $K_1, \dots, K_{10}$	[0.5,1.40] (even intervals)	[0.5,1.4] (even intervals)
Outputs	Implied volatilities: $\Sigma_1, \dots, \Sigma_{10}$	[0.08,0.84]	[0.19,0.83]

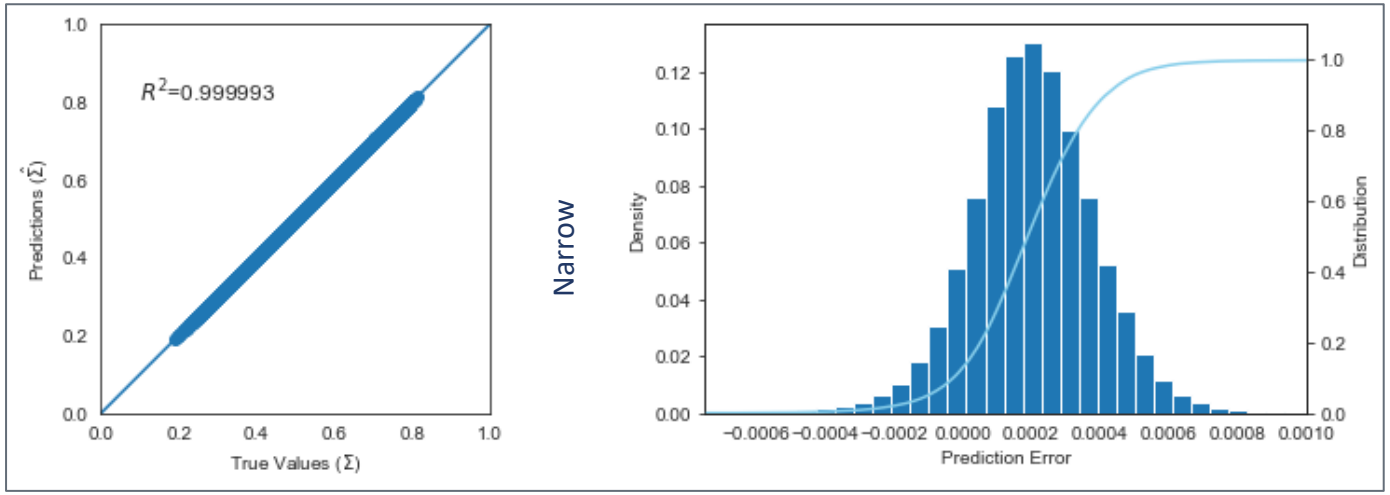


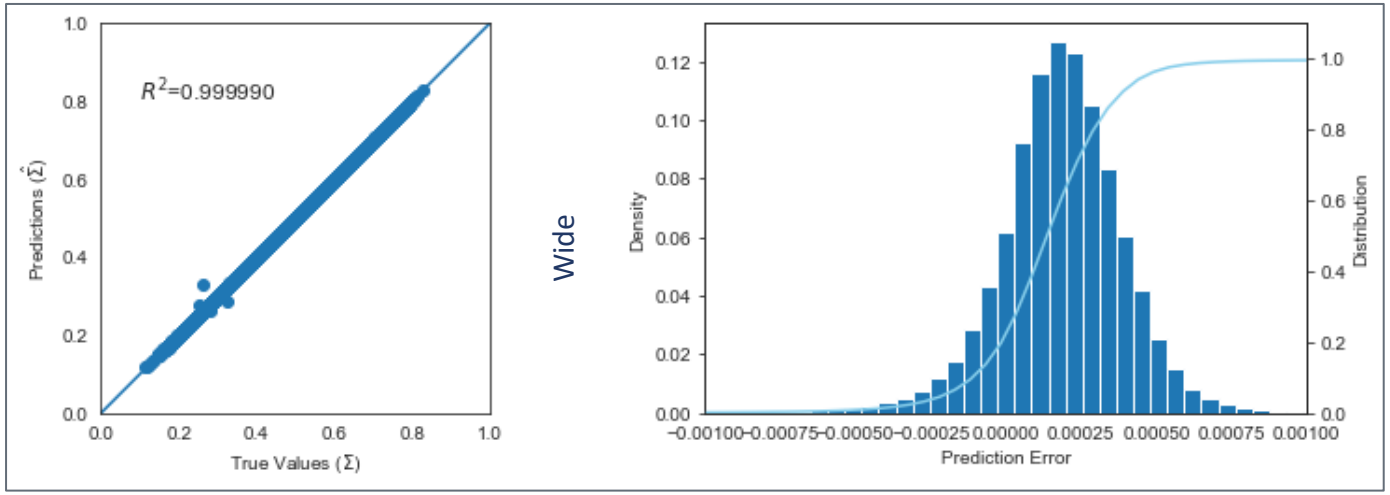
Above Figure:

- Number of parameters (weights and biases) are fixed (488,410)
- Training time is lowest at 4 layers i.e. increase depth rather than width of NN for optimal computational time
- MSE is lowest and most stable at 4 layers i.e. better approximation deeper NN (but not too deep)
- Above findings fall in line with other research findings

Test Results: Heston DNN (2/3)

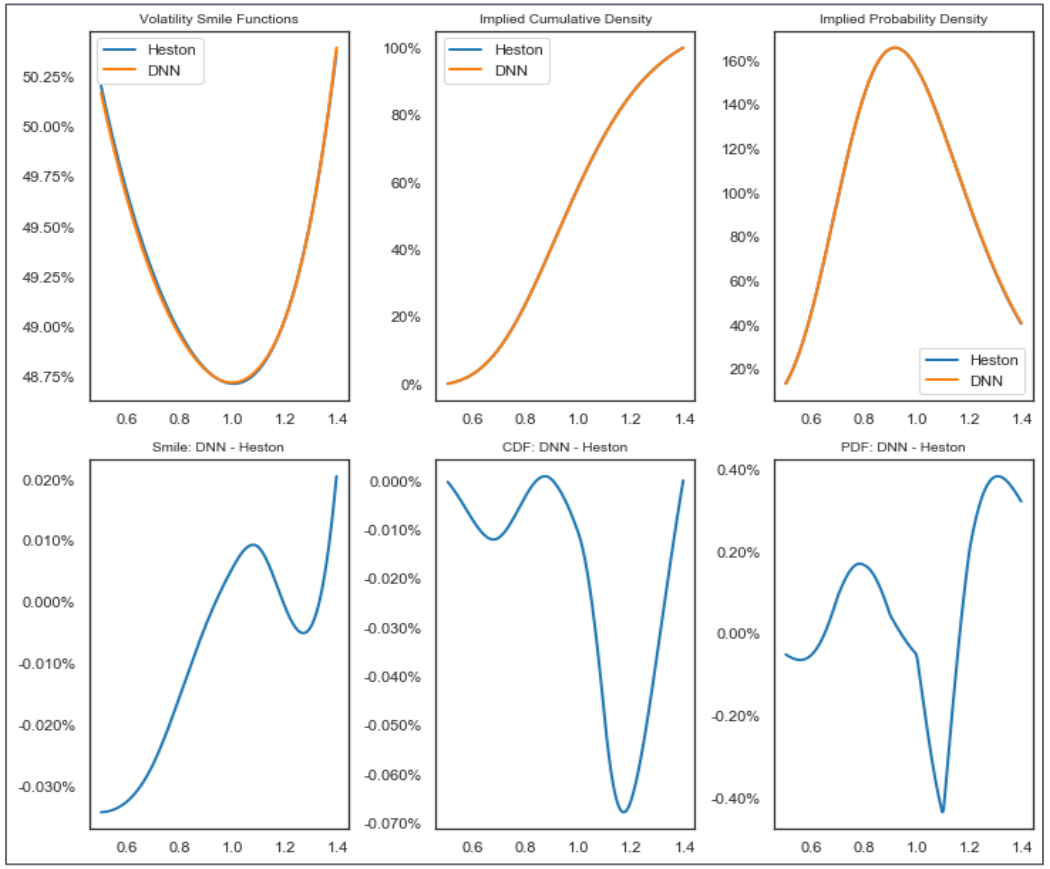
Train/Val sample	Test samples	MSE validation	MSE test-wide	MSE test-narrow	Epochs	Time
590,000	10,000	7.08e-08	1.20e-07	7.60e-08	501	10h





- Condition 2 was met i.e. stopped after 10h
- Lower MSE for the narrow test-data compared to both the wide and ANN test-data
- Most errors in [-0.05%,0.08%] for the narrow test-set and [-0.08%,0.09%] for wide test-set
- Seems to be good practice to train on a dataset that is wider than parameters of interest
- Both R^2's are close to 1 i.e. close to no distance between predicted and actual values

Test Results: Heston DNN (3/3)



$(\tau = 0.2799, r = 0.0146, \sigma = 0.1836, \rho = -0.3066,$
$v_0 = 0.2350, \theta = 0.2038, \kappa = 1.217)$

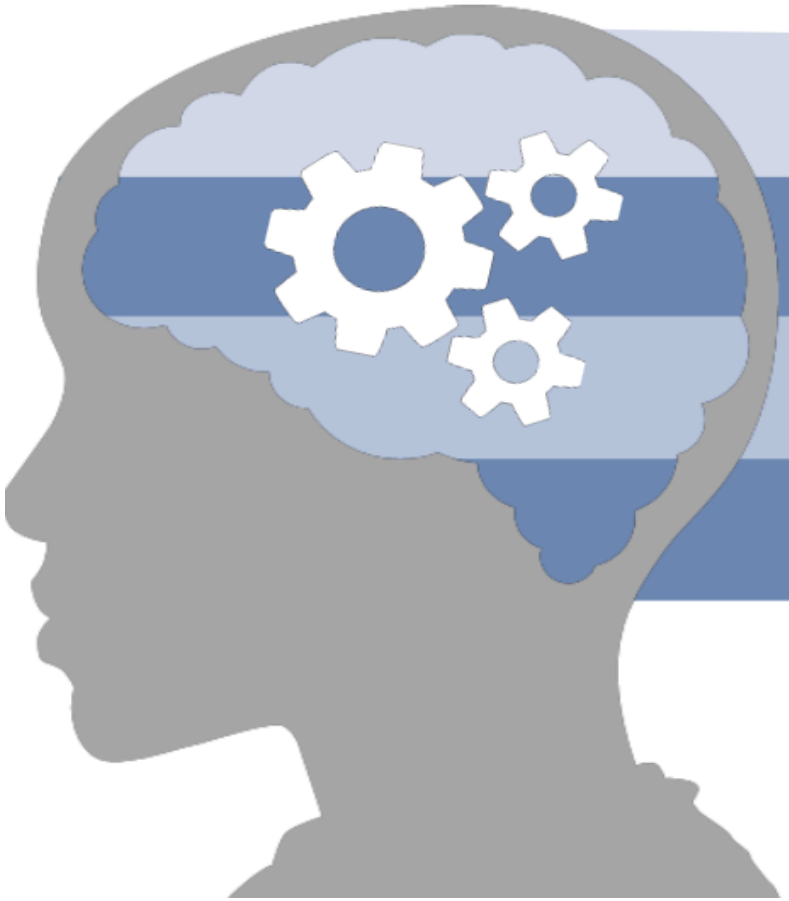
- Cubic spline interpolation the ANN follows the true values well
- Smooth first and second derivatives (CDF and PDF respectively)
- Smile function error [-0.03%0.02%]
- CDF error [-0.07%,0.00%] and PDF with error -0.3% at its worst
- Much better than for Heston ANN

- DNN still performs 37+ timers faster than numerical solution
- ANN is 3 times faster than DNN due to higher complexity of DNN

Method	Number of smiles	Time	Time per smile	Time per volatility
Heston	1,000	4,680.44ms	4.68e-04s	4.68e-05s
ANN	10,000	41.7ms	4.17e-06s	4.17e-07s
DNN	10,000	124ms	1.24e-05s	1.24e-06s

Conclusion

Conclusion



ANNs and DNNs can efficiently reduce the computing time of pricing financial options whilst maintaining a high degree of accuracy

The “mild” Universal approximation theorem can be used to encapsulate a complex model to a given degree of accuracy

SABR ANN accurately represents the functional form of the SABR Approximation and lays the foundation for applying more computationally expensive variants of the model

Heston ANN performs 112+ times faster and DNN 37+ times faster than its numerical method

Discussion

Happy With The Mild Approximation Theorem?

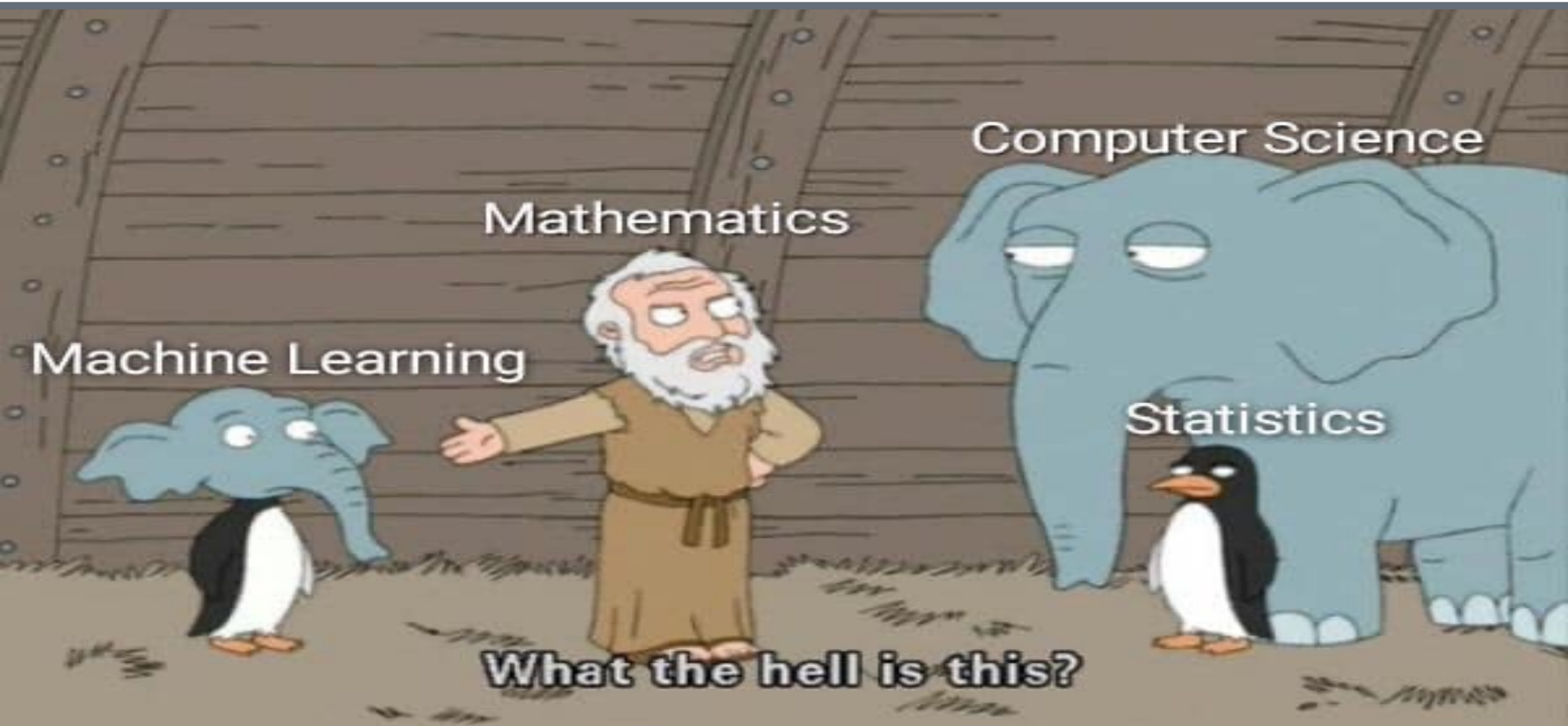
- Mild assumptions of the Universal approximation theorem were used due to the ReLU activation function not being differentiable at every point. This may not be accepted by all academics as can be seen from the following theorems

Theorem 1 (Universal approximation theorem – Hornik, Stinchcombe and White):
Let $\mathcal{NN}_{d_0,1}^{\sigma}$ be the set of neural networks with activation function $\sigma: \mathbb{R} \mapsto \mathbb{R}$ input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then, if σ is continous and non-constant, $\mathcal{NN}_{d_0,1}^{\sigma}$ is dense in $L^p(\mu)$ for all finite measures μ.

Theorem 2 (Universal approximation theorem for derivatives – Hornik, Stinchcombe and White):
Let $F^* \in \mathrm{C}^n$ and $F: \mathbb{R}^{d_0} \to \mathbb{R}$ and $\mathcal{NN}_{d_0,1}^{\sigma}$ be the set of single-layer neural networks with activation funciton $\sigma: \mathbb{R} \mapsto \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension 1. Then, if the (non-constant) activation function is $\sigma \in \mathrm{C}^n(\mathbb{R})$, then $\mathcal{NN}_{d_0,1}^{\sigma}$ arbitrarily approximates f and all its derivatives up to order n.

- Theorem 2 highlights that the smoothness properties of the activation function are of importance in the approximation of derivatives of the target function F^*
- In order to gurantee the convergence of the l-th order derivatives of the target function one needs to choose an activation function $\sigma \in \mathrm{C}^l(\mathbb{R})$
- Note, ReLU is not in $\mathrm{C}^l(\mathbb{R})$ for any $l > 0$, but ELU and Softplus is

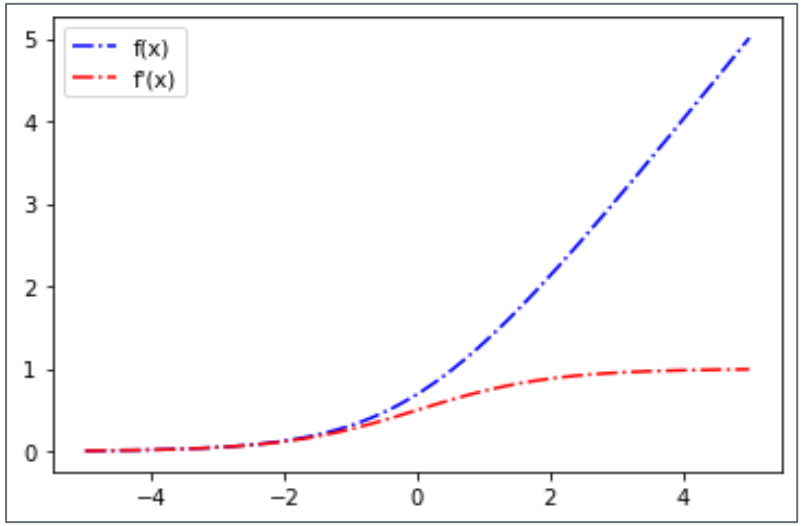
Q&A


Computer Science
Mathematics
Machine Learning
Statistics
What the hell is this?

Appendix

Why Not Use Softplus (Tested On SABR)?

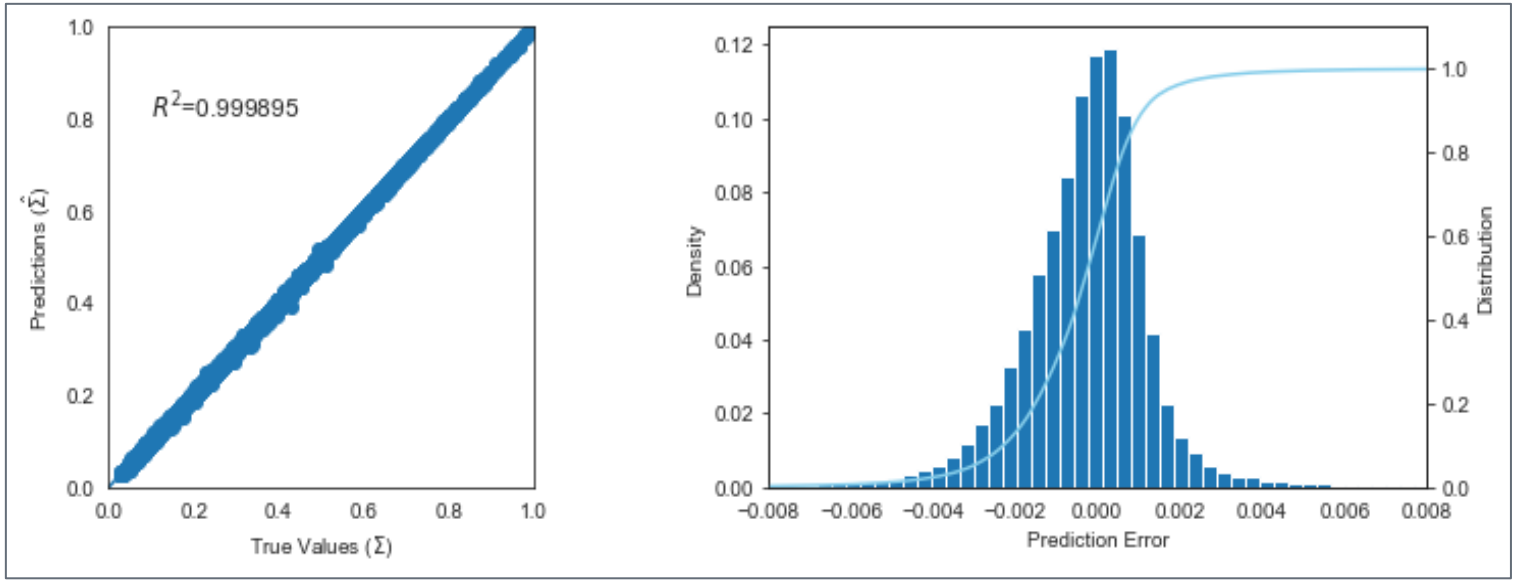
Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	3.00e-06	3.00e-06	2.82e-06	2015	10h



Softplus activation function:

$$f(x) = \ln(1 + e^{x})$$

$$f'(x) = \frac{1}{1 + e^{-x}}$$

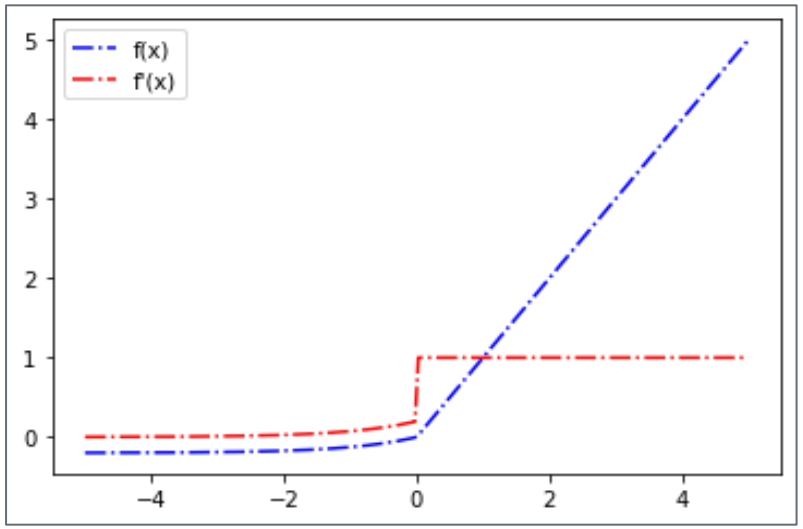


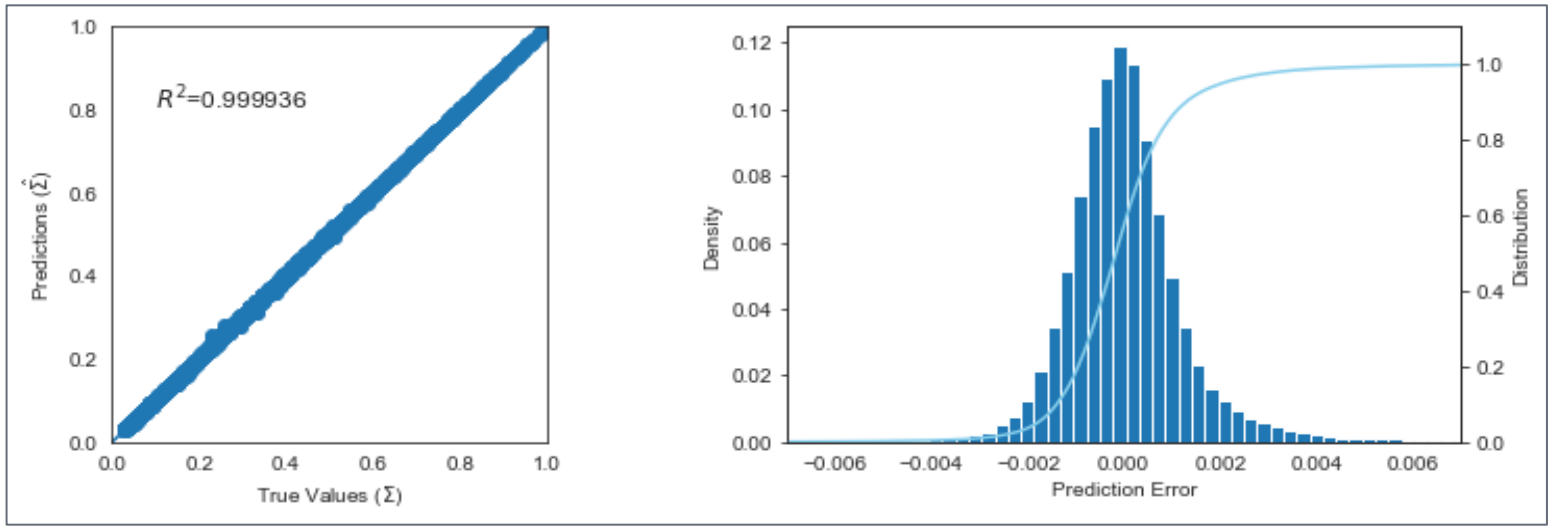
- One might be encouraged to argue that in cases like this the Softplus activation function should be implemented
- However, as cited by Glorot et al. (2011) and again in Goodfellow et al. (2016)

  The use of softplus is generally discouraged. ... one might expect it to have advantage over the rectifier due to being differentiable everywhere or due to saturating less completely, but empirically it does not

Why Not Use ELU (Tested On SABR)?

Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	2.00e-06	2.00e-06	1.73e-06	2096	10h

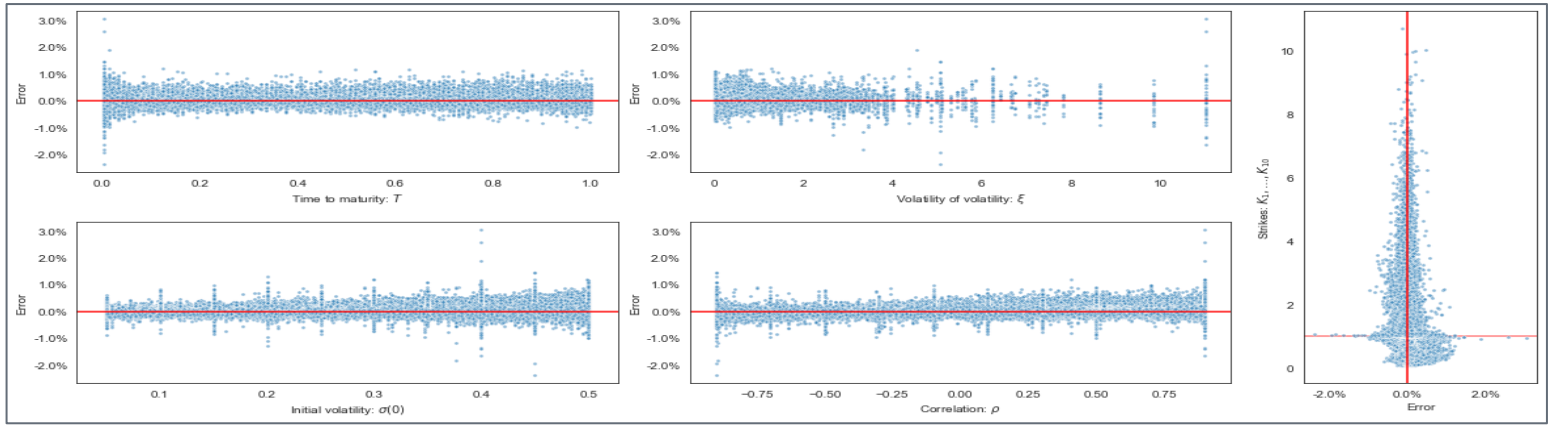




ELU activation function:

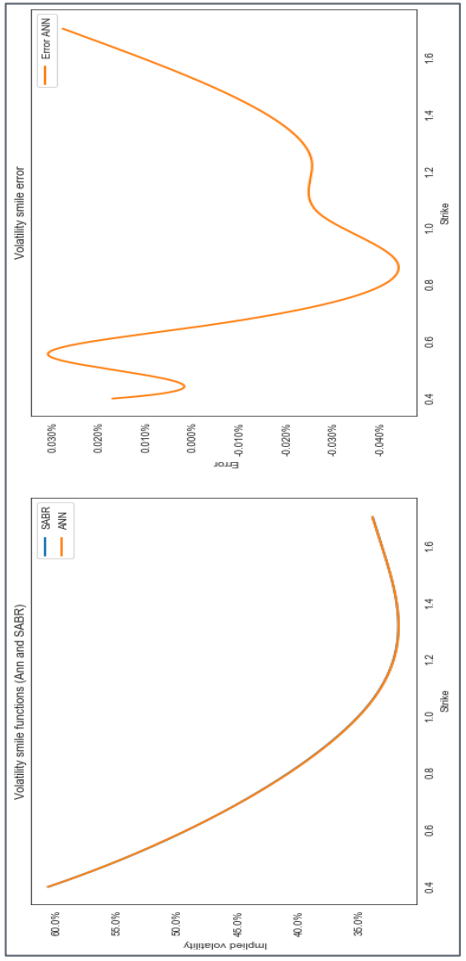
$$f(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$$

$$f'(x) = \begin{cases} 1 & x > 0 \\ \alpha e^x & x \leq 0 \end{cases}$$



Testing On SABR Integration Data

Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	4.82e-07	2.69e-07	2.56e-07	731	4h 48min 18s



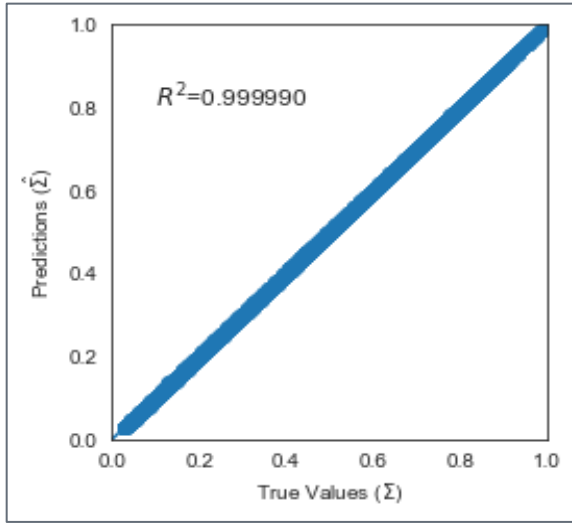
Left Picture:

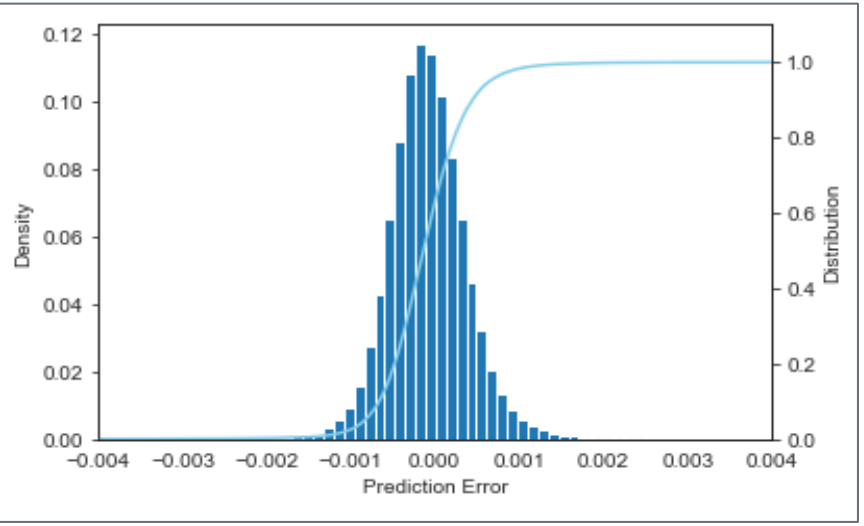
$T = 0.3049$
$\sigma_0 = 0.35$
$\xi = 0.9322$
$\rho = -0.5$
$K_1 = 0.4012$
$K_{10} = 1.7062$

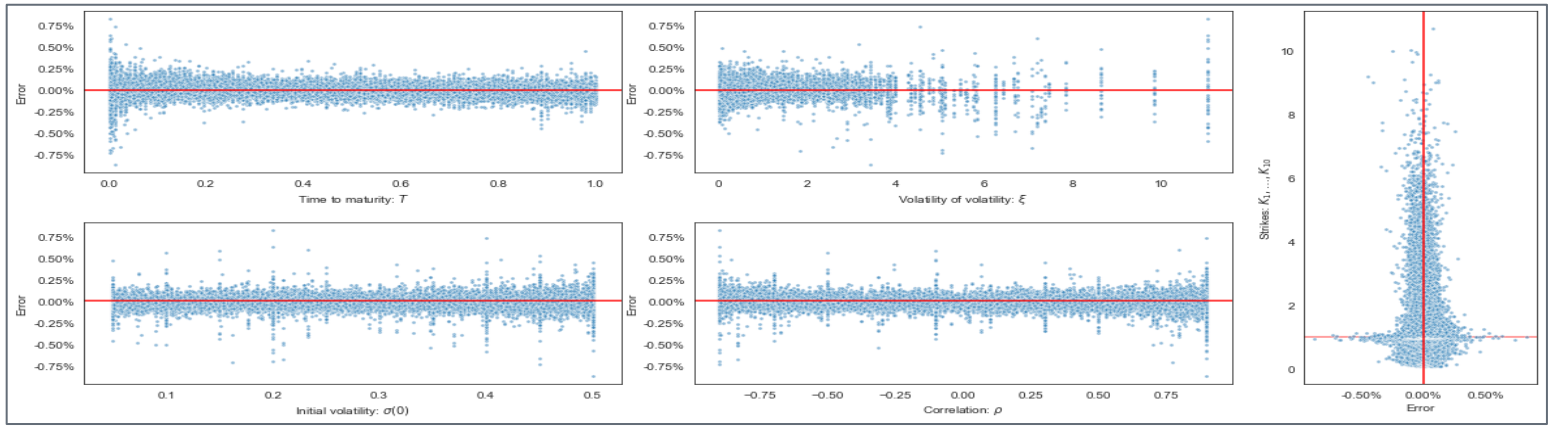
Error range:
[-0.042%,0.030%]

General:
Error range:
[-0.873%,0.826%]

Calculation time:
42.3ms

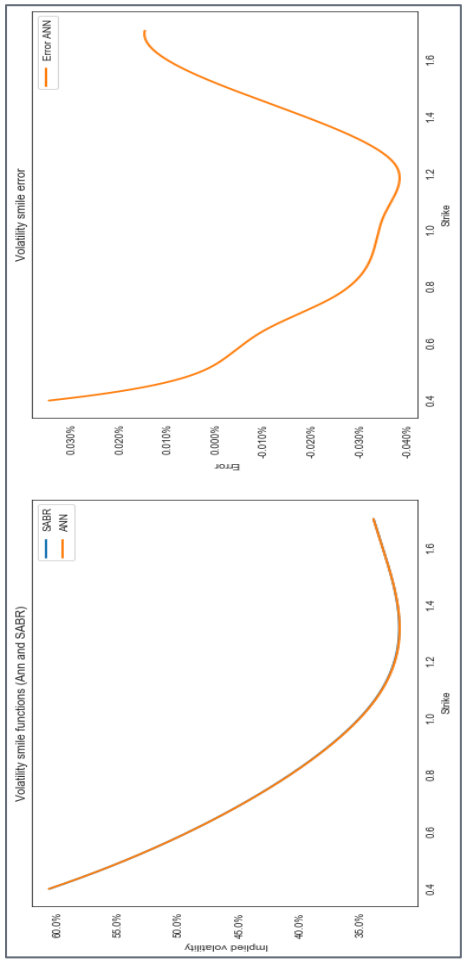






Testing On SABR Integration Data (DNN)

Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	1.5e-07	6.54e-08	6.50e-08	886	10h



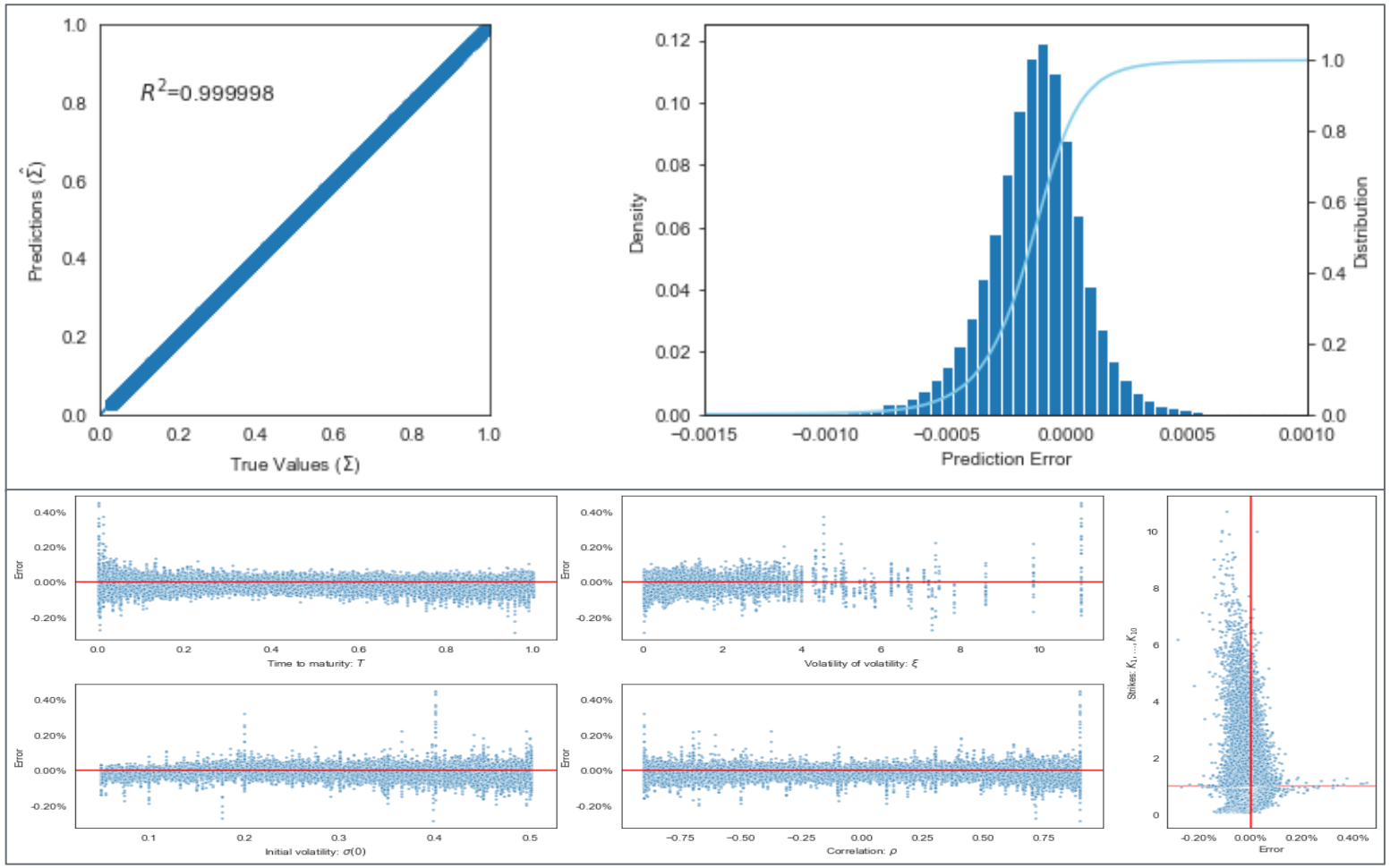
Left Picture:

$T = 0.3049$
$\sigma_0 = 0.35$
$\xi = 0.9322$
$\rho = -0.5$
$K_1 = 0.4012$
$K_{10} = 1.7062$

Error range:
[-0.039%,0.034%]

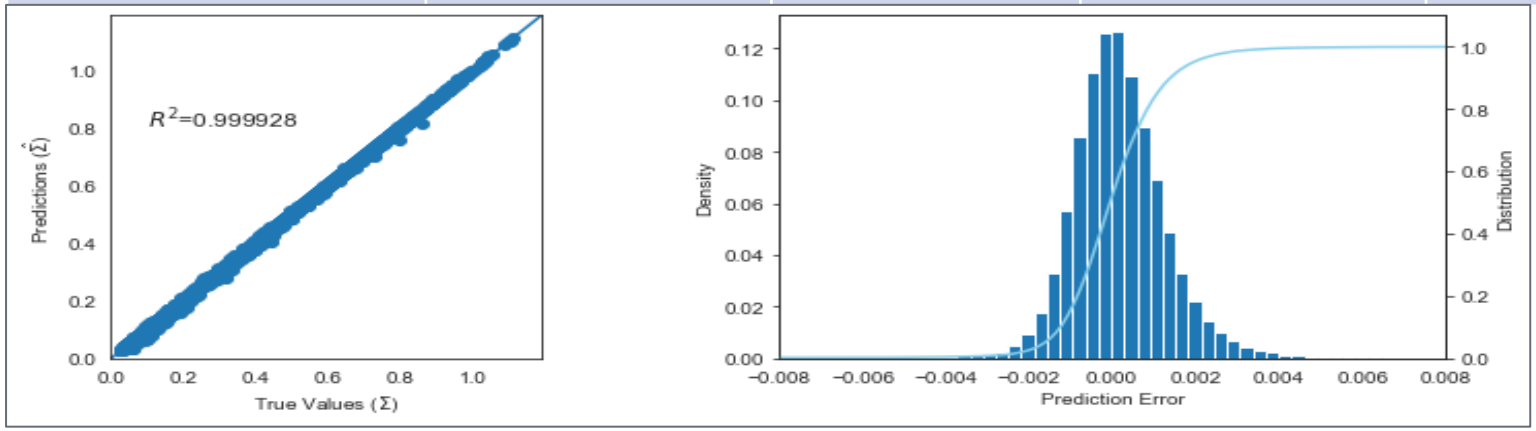
General:
Error range:
[-0.29%,0.45%]

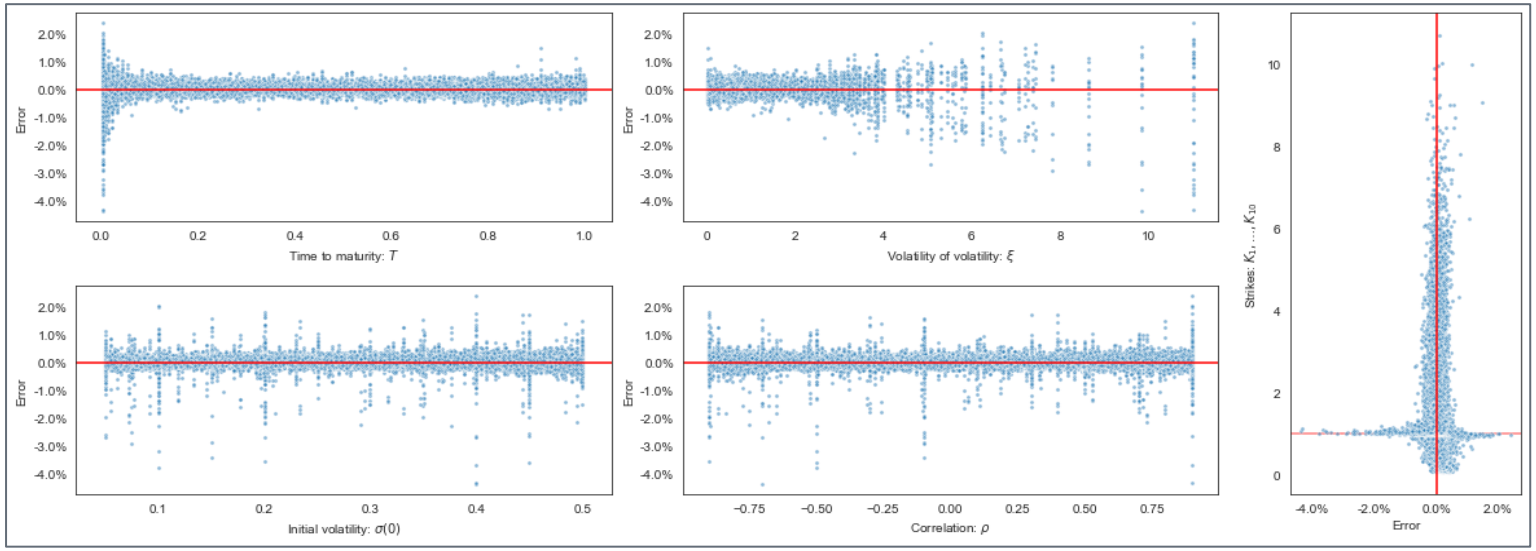
Calculation time:
123ms



Compromising The Accuracy For The Sake Of Speed (SABR)

Train/Val sample	Test samples	MSE train	MSE val	MSE test	Epochs	Time
290,000	10,000	1.00e-06	2.00e-06	2.22e-06	1324	7h 35min 16s

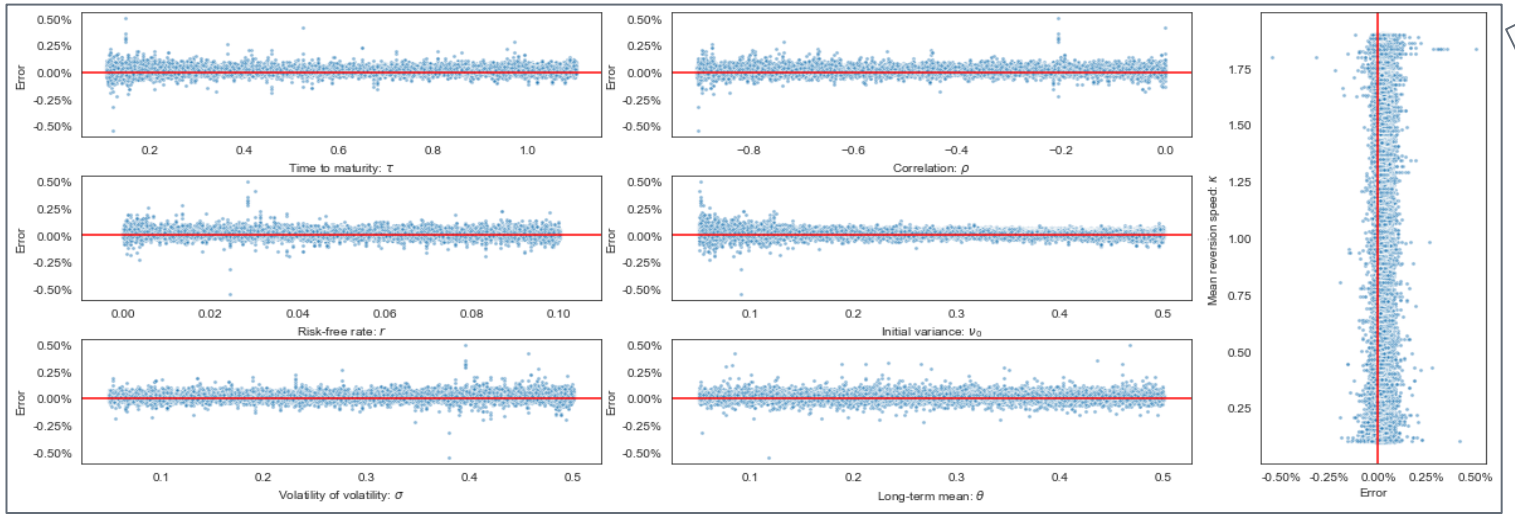




- What if we compromise the accuracy of the ANN purely for the sake of approximation speed?
- Hidden layer size = 250 neurons
- All errors are in the range of [-4.38%,2.41%]
- However, the SABR ANN now calculates 2 times faster than the SABR Approximation

Method	Number of smiles	Time
SABR	1,000	23.2ms
ANN	10,000	10.6ms
Method	**Time per smile**	**Time per volatility**
SABR	2.32e-06s	2.32e-07s
ANN	1.06e-06s	1.06e-07s

Heston And SABR Scatterplot



Heston ANN ↑ - SABR ANN ↓