A Neural Network Based Framework for Financial Models Summary

4.3 Backward Pass (Heston Model)

σ_{imp} | K, τ, S_0, r -> Heston-CaNN -> ρ, k, v_0, $\overline{v}$, γ

How does this relate to our research?

- We hope to be able to use historical asset prices and option prices to be able to calibrate our parameters. This mean, our input depends on the different parameters that make these prices/ returns, which would be strike prices, initial price, rate, time to maturity and the implicit volatility.
- We then send these input to the CaNN where it would train the model
- After we train the model, we can use our Joint objective function to return the calibrated parameters, where in our case, since we are just caring about the GARCH(1, 1) model would be ω, α, β.

Sampling Training Data

Found in Table 5 of the paper:

	Parameters	**Range**	**Samples**
Market data	Moneyness, $m = \frac{S_0}{K}$ Time to maturity, τ Risk free rate, r European call/put price, $\frac{V}{K}$	$[0.85, 1.15]$ $[0.5, 2.0]$(year) 0.03 $(0.0, 0.6)$	5 7 Fixed, •
Black-Scholes	Implied Volatility	$(0.2, 0.5)$	35

We can use this as a way to sample all our different input parameters, and be able to measure how more accurate the model becomes and its calibration.

During the calibration, they use the total squared error measure $J(\Theta)$:

$$J(\Theta) = \sum \omega \big(\sigma_{\text{imp}}^{\text{ANN}} - \sigma_{\text{imp}}^{*}\big)^2 + \overline{\lambda}|\Theta|$$

Averaged performance of the Backward pass of the CaNN:

- Need to list CPU and GPU spec
  - OS: Linux pop-os 6.6.10-76060610-generic
  - CPU: AMD Ryzen 5 5600G
  - GPU: Radeon 6600

Abosulute deviation from θ^*, Error measure and computational cost.

Error Measure: $J(\Theta)$, MJ and Data Points
Computational Cost: CPU, GPU time and Function evaluations

Summary of Joint Calibrarion Artificial Neural Networks

Goals & Designs

Our goal is to be able to use two forms of input data to be able to calibrate an optimum parameters of the GARCH(1, 1) model. This will be calibrated using joint calibration which will take into account the log likelihood of returns and option prices to be able to consider both physical and risk neutral measures.

The design will make use of a backward pass artificial neural network, where we will do the following:

Market Data (S, K, S_0, r, τ, sigma $| R_t$) $\longrightarrow$ Joint Calibration Neural Network $\longrightarrow$ GARCH Params (ω, α, β)

Where we consider the risk neutral parameters:

- S: Price
- K: Strike Price
- S_0: Initial Price
- r: Risk-free rate (fixed)
- σ: Volatility
- τ: Time to maturity

And we consider the following physical measure:

- R_t: Log return at time t

Our goal is to be able to use the Joint Calibration to have the most optimum calibrated GARCH parameters that takes into account both measures. This will come into the calibration phase, where the idea is to utilize the Joint Calibration formula as the objective function for minimization.

Input Data

So firstly we need to discuss how we get our input parameters?

Risk-Neutral Measure:

- Option Prices (European, American, Asian, etc.)

Physical Measure:

- Historical Asset Prices (change in difference to get log return)
  - Where: $R_t \equiv \ln\left(\frac{S_t}{S_{t-1}}\right)$
  - Can be easily done using `numpy.diff(numpy.log(S))`

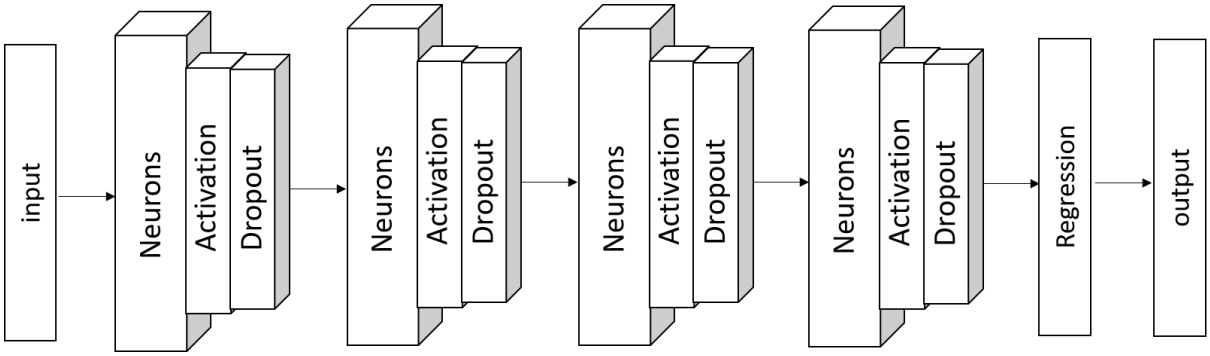
Joint Calibration Neural Network

This neural network will be written in Python with a minimum required version of 3.10, and will require the following dependencies:

```
[tool.poetry.dependencies]
python = "^3.10"
numpy = "^1.26.4"
pandas = "^2.2.2"
arch = "^7.0.0"
scikit-learn = "^1.5.0"
matplotlib = "^3.9.0"
scipy = "^1.13.1"
```

The architecture currently will follow simarly to the paper, as the following:

Parameters	Options
Hidden Layers	4
Neurons(each layer)	200
Activation	ReLu
Dropout rate	0.0
Batch-normalization	No
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024



Where we then hope in the regression phase, to be able to use the Joint Calibration formula in a batching method to be able to calibrate our parameters for the GARCH model.

Essential Equations

These are from GARCH Option Valuation Theory and Evidence:

Return Log Likelihood:

$$\ln L^R \propto -\frac{1}{2}\sum_{\mathrm{t}=1}^{T}\left\{\ln(h(t)) + \frac{\left(R(t)-\mu_t-\gamma_t\right)^2}{h(t)}\right\}$$

where:

- $h(t)$ is the conditional variance
- $R(t)$ represents the return at time t
- μ_t is the conditional mean of the returns at time t
- γ_t is an adjustment term

Options Log Likelihood:

$$\ln L^O \propto -\frac{1}{2}\sum_{\mathrm{i}=1}^{N}\left\{\ln(\mathrm{IVRMSE}^2) + \left(\frac{e_{\mathrm{i,t}}}{\mathrm{IVRMSE}}\right)^2\right\}$$

Implied Volatility root mean squared error (IVRMSE) loss function follows:

$$\mathrm{IVRMSE} \approx \sqrt{\frac{1}{N_T}\sum_{\mathrm{i,\ t}}^{N_T} e_{\mathrm{i,t}}^2}$$

where:

- N_T: total number of option prices in the sample

The verga weighted option error follows as:

$$e_{\mathrm{i,t}} = \frac{C_{\mathrm{i,t}} - C_{\mathrm{i,t}}(h_t(\xi^*))}{\text{Vega}_{\mathrm{i,t}}}$$

where:

- $\text{Vega}_{\mathrm{i,t}}$ is the Black-Scholes sensitivity of the option prices with respect to volatility
- ξ^* is the vector of risk-neutral parameters to be estimated
- $C_{\mathrm{i,t}}$ is the market option price
- $C_{\mathrm{i,t}}(h_t(\xi^*))$ is the model price

Joint Log Likelihood

$$L_{\text{joint}} = \frac{T + N_T}{2}\frac{L^R}{T} + \frac{T + N_T}{2}\frac{L^O}{N}$$

where:

- T is the number of days in the return sample
- N_T is the total number of option contracts

Creating Synthetic Data For European Options and Returns

```
import numpy as np
from scipy.stats import norm

# Parameters for GBM
S0 = 100    # Initial stock price
r = 0.1    # risk free rate
sigma = 0.2 # Volatility (annual)
T = 1.0     # Time period in years
dt = 1/252  # Time step (daily)
n_steps = int(T / dt)

# Generate random Brownian motion
np.random.seed(42)
W = np.random.standard_normal(size=n_steps)
W = np.cumsum(W) * np.sqrt(dt)

# Simulate stock prices
t = np.linspace(0, T, n_steps)
S = S0 * np.exp((r - 0.5 * sigma**2) * t + sigma * W)

# Calculate returns
returns = (S[1:] - S[:-1]) / S[:-1]
log_returns = np.diff(np.log(S))
```

The goal is to be able to train off of American options using the Willow Tree method to generate synthetic data.