Our model is based on the work of Molina[1] and Moody et al[2]. It uses Recurrent Reinforcement learning to maximize the Sharpe Ratio for a financial asset (Hang Seng Futures in our case). The model is defined as follows.

Variables:

N = number of iterations for the RL algo
Mu = max number shares per transaction
Delta = transaction costs in bps per trade
T = test set size
M = training set size

Input data:

Px = price array
rt = log(px_t / px_t-1)

Optimized variable:

$\theta = \{v,\ v_i,\ w\}$

Calculated variables:

Trader Function:

$F_t = tanh(w^T x_t) \in \{-1,\ 0,\ 1\}$

where $x_t = [1,\ r_t,\ ...,\ r_{t-M},\ F_{t-1}]$

$M = Number\ of\ time\ series\ inputs,\ i.e.\ length\ of\ return\ array$

$r_T = log\ return = ln\frac{p_t}{p_{t-1}}$

Trade Return (return from trading):

$R_t = \mu\left(F_{t-1}v_t - \delta\left|F_t - F_{t-1}\right|\right)$

where $\mu = maximum\ number\ of\ shares\ per\ transaction$

$\delta = transaction\ cost\ in\ bps$

Recurrent Reinforcement Learning:

$$\frac{dU_T(\theta)}{d\theta} = \sum_{t=1}^{T}\frac{dU_t}{dR_t}\left\{\frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta}\right\}$$

where $\frac{dR_t}{dF_t} = -1\mu\delta\left|F_t - F_{t-1}\right|$

$\frac{dR_t}{dF_{t-1}} = \mu r_t + \mu\delta\left|F_t - F_{t-1}\right|$

$\frac{dF_t}{d\theta} = \left(1 - tanh(x_t\theta)^2\right)x_t + w \cdot dF_{t-1}$

$\frac{dU_t}{dR_t} = Work\ in\ progress$

[1] "Stock Trading with Recurrent Reinforcement Learning (RRL)", Gabriel Molina
[2] "Learning to Trade via Direct Reinforcement", John Moody and Matthew Saffell, 2001

Gradient Ascent

$$\frac{AvgR_t}{\frac{\sqrt{\sum (R_{t<0}-\overline{R_{t<0}})^2}}{M-1}} = \frac{\frac{1}{T}\sum_{t=1}^{T} R_t}{\frac{\sqrt{\sum \left(R_{t<0}-\frac{1}{T}\sum_{t=1}^{T} R_{t<0}\right)^2}}{M-1}}$$

Pseudo Code:

1) Load prices -> z_t = [px_0, … px_T]
2) Calculate log price returns r_t
3) Calc optimal theta by minimizing the score function (Sortino ratio)
4) Update trader function -> F_t
    a) F_t = tanh(W^T x_t)
5) Calculate trades via rewardFunction -> R_t
    a) R_t = max_shares_per_transaction * (F_t-1 * r_t - transaction_cost *(F_t - F_t-1))
    b) Output:
        i) R_t trade return
6) Calculate Sortino ratio using R_t
   Sortino Ratio: $\frac{r_t}{std\left|r_t<0\right|}$ for t=1, …, T
7) Calculate cumulative return -> cumReturn

```
# iterate
Ret = zero_array
Ft = zero_array
For i in range(0, numIterations):
        If i > 0:
                Theta, cost = minimize_CostFunction

        // update trader function
        F_tt = updateF_t

        // calc cumulative return and sortino ratio
        tradeRet, Sortino = rewardFunction

        // calculate cumulative return
        cumReturn = getCumulReturn

        // update returns
        Append return estimates

        // Roll to next training period
        // update Ft based on Ftt
        Append Ftt to Ft
```

A preliminary version of the Recurrent Reinforcement Learning algorithm has been coded and tested. The git repository is available at https://gitlab.com/stanford_cs221/project.git.

An array of 5 minute open high low close (OHLC) prices for the Hang Seng Index future is loaded, z_t, and the log return is returned. The log return of prices, X, is the primary variable for the Recurrent Reinforcement Learning algorithm.

An optimization based on minimizing the negative Sortino ratio is calculated over the training set and the output is theta, an array containing weights for the previous returns and trading signal. There are two functions that the optimizer utilizes to find the optimal theta - 1) a cost function that returns the Sortino ratio and 2) a gradient function that returns the change in gradient.

Once theta is found, the trading decision function, Ft, is updated based on the latest weights. The function is updated over the test period. A trader return and score is then generated for the test period based on the trading decisions and the cumulative return is saved into an array. The trading decision array is also saved for the next rolling window.

The optimization algorithm is failing and needs some further adjustment before the entire algorithm is functional. Additional work is also needed to integrate the supervised and unsupervised learning signals with the reinforcement learning algorithm. Further work to clarify the rolling of the training and test periods is also planned as well.

We will compare the results of the recurrent reinforcement learning with a base case using supervised learning.

For the base case we have extracted features using the market data (order book depth, cancellations, trades, etc.) and run a random forest algorithm to get a slightly positive Sortino ratio.

We want to see if the recurrent reinforcement learning algorithm can generate better results.

We will additionally explore if we can use LSTMs to predict the next price and see if they perform better than our base case. We will also explore adding the predicted log return into our Reinforcement learning model as an additional parameter to compare results.