# Data Encoding

## Balance

```
_sum_balance = tokBalance * current_close + curBalance

cur_frac = curBalance / (_sum_balance + eps)
tok_frac = (tokBalance * current_close) / (_sum_balance + eps)
order    = sigmoid(logp(_sum_balance)-4) # min 0, so -4 making full use of sigmoid
```

## High-Low (in-context)

This encoding is meant to be used in combination with open or close. The rational is that a naive high+low+close encoding will result in uninformative data when put into a neural network as they all have similar trends. This encoding ensures high & low only give the relevant information (that being the 'error' from open or close).

```
H = meandev(ediff1d(high-close))
L = meandev(ediff1d(close-low))
```

**Critisism:**

After testing this it seems the meandev doesn't work well as it loses the absolute variance between iterations, which is required for situations where an non-class output is required. The revised version below:

```
L = ediff1d((close-low)/close)
H = ediff1d((highe-close)/close)
```

# Model Design

## Recurrence

While recurrence doesn't add a significant amount of pattern recognition compared to a standard deep Feed-Forward Neural Network (FFNN), it is kind of useful for behaviour which mimics strategies like Moving Average Crossover. As the Recurrent Neural Networks (RNN) essentially compute an exponential moving average at each layer with variable alpha (determined by the weight).

RNNs, more generally, have the benefit of 'knowing' what they have previously computed, at the cost of having to compute on both "memory" and "state" instead of just "state".

Currently the best RNNs are GRUs, however we might experiment with liGRUs.

### Transformers

While transformers are all the hype in the industry, for our application & framework it seems hard to find the benefits of using them.

The current issues:

- Transformers are computationally expensive compared to RNNs
- Transformers are typically applied in scenarios where all inputs are distinct
- Transformers for timeseries are still open research as far as we know
- Transformers have quadratic/O(N^2) complexity

We have attempted to use transformers as an embedding layers by creating a Transformer-Encoder model, which tries to predict the next OHLCV, and using the second-to-last layer output as embedding, however this didn't give good results.

It might be worth exploring in the future SAC/ESAC instead of ES, but currently it's not feasible.

### Deep Multi Layer Perceptrons

We have expiremented with Filter Networks, these are simple multi-layer networks using WaveNet inspired FilterBlocks. FilterBlocks are essentially the WaveNet Residual layers, but with Linear layers instead of Convolutional; and no skip-connections.

## TODOs

- **Compare FilterNetworks to standard MLP Networks**
- **Try a regular wavenet structure**
- **Add deque of trade_log as input**
- **Add ROI as input?**