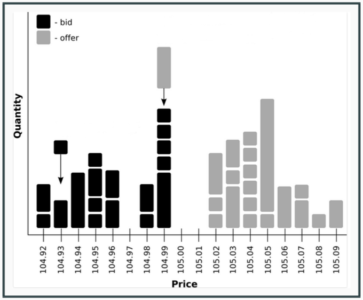
Deep Learning Financial Market Data

Steven Hutt steven.c.hutt@gmail.com

12 December, 2016

The Limit Order Book

A financial exchange provides a central location for electronically trading contracts. Trading is implemented on a Matching Engine, which matches buy and sell orders according to certain priority rules and queues orders which do not immediately match.



A. Booth©

A Marketable limit order is immediately matched and results in a trade.

A Passive limit order cannot be matched and joins the queue.

Data In: Client ID tagged order flow.

Data Out: LOB state (quantity at price).

Need to learn data sequences.

Representing the Limit Order Book

Level	Bid Px	Bid Sz	Ask Px	Ask Sz
1	23.0	6	24.0	2
2	22.5	13	24.5	5
3	22.0	76	25.0	17
4	21.5	132	25.5	14
5	21.0	88	26.0	8

5 Level Limit Order Book

$$a_t = \underbrace{(88, 132, 76, \ldots, 14, 8)}_{\text{10 elements}}$$

$$b_t = (23.0, 24.0)$$

At a given point in time, the LOB may be represented by:

Bid Side: The quantity of orders at each price level at or below the best bid.

Ask Side: The quantity of orders at each price level at or above the best ask.

A possible representation $x_t = a_t \sqcup b_t$ is given on the left.

The evolution of the LOB over time is represented by a sequence:

$$(x_1, x_2, x_3, \cdots, x_n, \cdots)$$

LOB Pattern Search for Prohibited Trading Patterns

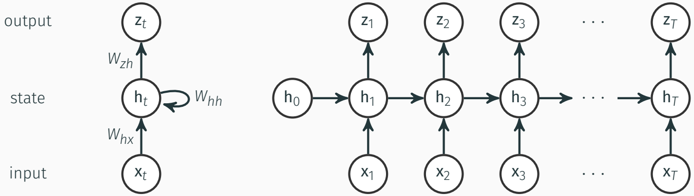
Motivation Regulators identify prohibited patterns of trading activity detrimental to orderly markets. Financial Exchanges are responsible for maintaining orderly markets. (e.g. Flash Crash and Hound of Hounslow.)

Challenge Identify prohibited trading patterns quickly and efficiently.

Goal Build a trading pattern search function using Deep Learning. Given a sample trading pattern identify similar patterns in historical LOB data.

Learning Sequences with RNNs

In a Recurrent Neural Network there are nodes which feedback on themselves. Below $x_t \in \mathbb{R}^{n_x}$, $h_t \in \mathbb{R}^{n_h}$ and $z_t \in \mathbb{R}^{n_z}$, for $t = 1, 2, \ldots, T$. Each node is a vector or 1-dim array.



output
state
input
z_t
W_{zh}
h_t
W_{hh}
W_{hx}
x_t
h_0
z_1 z_2 z_3 ... z_T
h_1 h_2 h_3 ... h_T
x_1 x_2 x_3 ... x_T

The RNN is defined by update equations:

$$h_{t+1} = f(W_{hh}h_t + W_{hx}x_t + b_h), \quad z_t = g(W_{yh}h_t + b_z).$$

where $W_{uv} \in \mathbb{R}^{n_u \times n_v}$ and $b_u \in \mathbb{R}^{n_u}$ are the RNN weights and biases respectively. The functions f and g are generally nonlinear.

Given parameters $\theta = \{W_{hh}, W_{hx}, W_{yh}, b_h, b_z\}$, an RNN defines a mapping of sequences

$$F_\theta : X = (x_1, x_2, \cdots, x_T) \mapsto Z = (z_1, z_2, \cdots, z_T).$$

Supervised Training for RNNs

Supervised Learning: Given sample input sequences $\{X^{(1)}, X^2, \cdots X^{(K)}\}$ with corresponding outputs $\{Z^{(1)}, Z^2, \cdots Z^{(K)}\}$, choose the RNN parameters θ so as to minimize

$$\mathcal{L}(\theta) = \sum_{t=k}^{K} \mathcal{L}_k(\theta) \quad \text{where} \quad \mathcal{L}_k(\theta) = \sum_{t=1}^{T} L(F_\theta(x_t^{(k)}), z_t^{(k)})$$

and $L_t(\theta) = L(F_\theta(x_t), z_t)$ is some loss function, e.g. least squares.

Gradient Descent: Fixing some time t above, we have

$$\frac{\partial L_t}{\partial \theta} = \sum_{i=1}^{t} \frac{\partial L_t}{\partial x_t} \frac{\partial x_t}{\partial x_i} \frac{\partial x_i}{\partial \theta} \quad \text{where} \quad \frac{\partial x_t}{\partial x_i} = \prod_{t \geq j > i} \frac{\partial x_j}{\partial x_{j-1}}$$

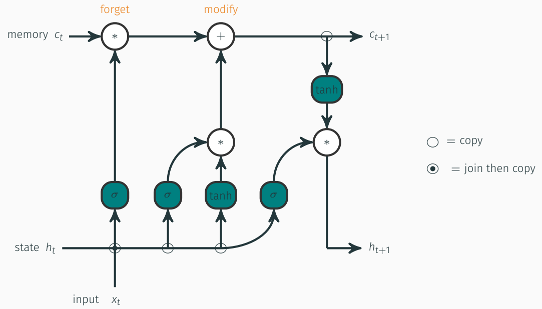
But this final product leads to vanishing / exploding gradients! Vanishing gradients limits the RNNs capacity to learn from the distant past. Exploding gradients destroy the Gradient Descent algorithm.

Problem: How to allow information in early states to influence later states? Note that trading patterns can extend over 100s of time steps.

Long Short Term Memory Units

We would like to create paths through time through which the gradient neither vanishes nor explodes. Sometimes need to forget information once it has been used.

An LSTM [3] learns long term dependencies by adding a memory cell and controlling what gets modified and what gets forgotten:



An LSTM can remember state dependencies further back than a standard RNN and has become one of the standard approaches to learning sequences.

Unitary RNNs

The behaviour of the LSTM can be difficult to analyse.

An alternative approach [1] is to ensure there are no contractive directions, that is require $W_{hh} \in U(n)$, the space of unitary matrices.

This preserves the simple form of an RNN but ensures the state dependencies go further back in the sequence.

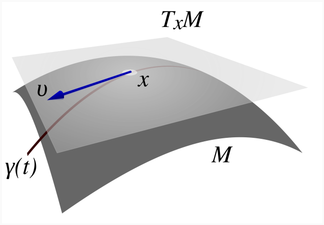
A key problem is that gradient descent does not preserve the unitary property:

$$x \in U(n) \mapsto x + \lambda v \notin U(n)$$

The solution in [1] is to choose W_{hh} from a subset generated by parametrized unitary matrices:

$$W_{hh} = D_3 R_2 \mathcal{F}^{-1} D_2 \Pi R_1 \mathcal{F} D_1.$$

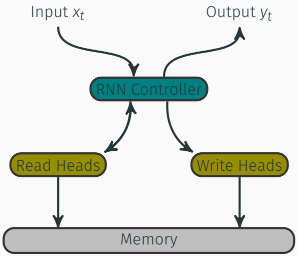
Here D is diagonal, R is a reflection, $\mathcal{F}$ is the Fourier transform and Π is a permutation.



RNNs as Programs addressing External Memory

RNNs are Turing complete but the theoretical capabilities are not matched in practice due to training inefficiencies.

The Neural Turing Machine emulates a differentiable Turing Machine by adding external addressable memory and using the RNN state as a memory controller [2].

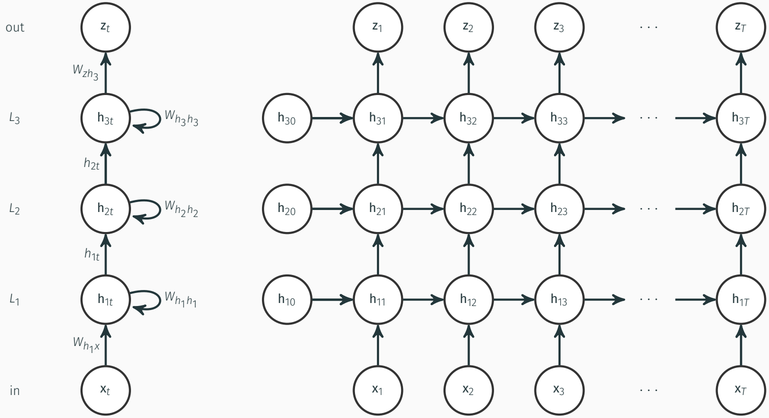


Vanila NTM Architecture

NTM equivalent 'code'

```
initialise: move head to start location
while input delimiter not seen do
   receive input vector
   write input to head location
   increment head location by 1
end while
return head to start location
while true do
   read output vector from head location
   emit output
   increment head location by 1
end while
```

Deep RNNs

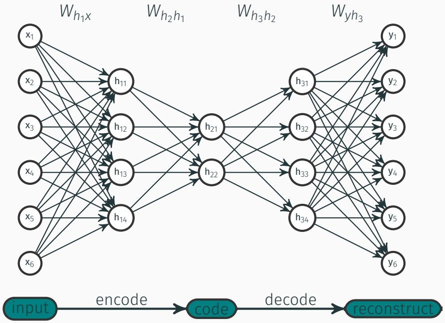


The input to hidden layer L_i is the state h_{i-1} of hidden layer L_{i-1}. Deep RNNs can learn at different time scales.

Unsupervised Learning: Autoencoders

The more regularities there are in data the more it can be compressed. The more we are able to compress the data, the more we have learned about the data.

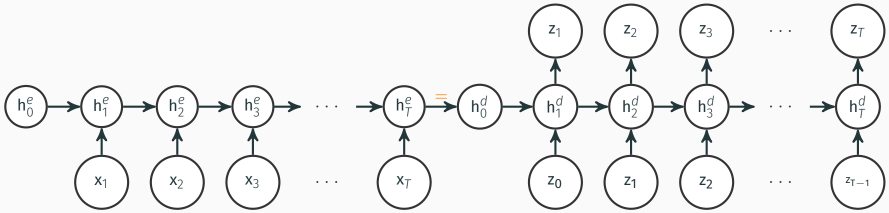
(Grünwald, 1998)



W_{h_1x}
$W_{h_2h_1}$
$W_{h_3h_2}$
W_{yh_3}
x_1 x_2 x_3 x_4 x_5 x_6
h_{11} h_{12} h_{13} h_{14}
h_{21} h_{22}
h_{31} h_{32} h_{33} h_{34}
y_1 y_2 y_3 y_4 y_5 y_6
input
encode
code
decode
reconstruct

Autoencoder

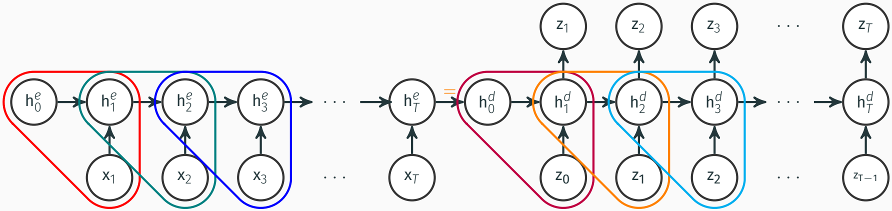
Recurrent Autoencoders



encoder decoder

A Recurrent Autoencoder is trained to output a reconstruction z of the the input sequence x. All data must pass through the narrow h_T^e. Compression between input and output forces pattern learning.

Recurrent Autoencoder Updates



encoder

decoder

$$\mathbf{h}_1^e = f(W_{hh}^e \mathbf{h}_0^e + W_{hx}^e \mathbf{x}_1 + b_h^e)$$
$$\mathbf{h}_2^e = f(W_{hh}^e \mathbf{h}_1^e + W_{hx}^e \mathbf{x}_2 + b_h^e)$$
$$\mathbf{h}_3^e = f(W_{hh}^e \mathbf{h}_2^e + W_{hx}^e \mathbf{x}_3 + b_h^e)$$
$$\dots$$

encoder updates

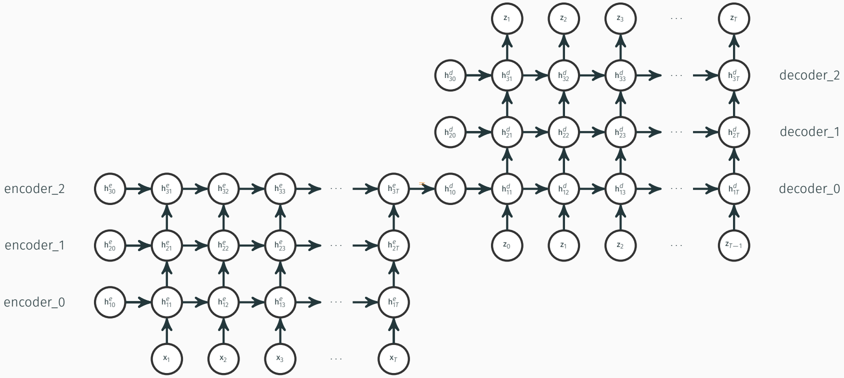
$$\mathbf{h}_0^d = \mathbf{h}_T^e$$

copy

$$\mathbf{h}_1^d = f(W_{hh}^d \mathbf{h}_0^d + W_{hx}^d \mathbf{z}_0 + b_h^d)$$
$$\mathbf{h}_2^d = f(W_{hh}^d \mathbf{h}_1^d + W_{hx}^d \mathbf{z}_1 + b_h^d)$$
$$\mathbf{h}_3^d = f(W_{hh}^d \mathbf{h}_2^d + W_{hx}^d \mathbf{z}_2 + b_h^d)$$
$$\dots$$

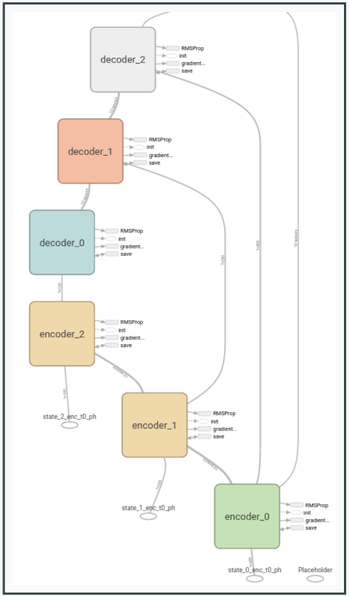
decoder updates

Deep Recurrent Autoencoder



A Deep Recurrent Autoencoder allows learning patterns on different time scales. This is important for trading patterns which may occur over a few or multiple time steps.

TensorFlow Deep Recurrent AutoEncoder



```python
with tf.variable_scope('encoder_0') as scope:
    for t in range(1, seq_len):
        # placeholder for input data
        xs_0_enc[t] = tf.placeholder(shape=[None, nx_enc])
        hs_0_enc[t] = lstm_cell_l0_enc(xs_0_enc[t], hs_0_enc[t-1])
        scope.reuse_variables()

with tf.variable_scope('encoder_1') as scope:
    for t in range(1, seq_len):
        # encoder_1 input is encoder_0 hidden state
        xs_1_enc[t] = hs_0_enc[t]
        hs_1_enc[t] = lstm_cell_l1_enc(xs_1_enc[t], hs_1_enc[t-1])
        scope.reuse_variables()

with tf.variable_scope('encoder_2') as scope:
    for t in range(1, seq_len):
        # encoder_2 input is encoder_1 hidden state
        xs_2_enc[t] = hs_1_enc[t]
        hs_2_enc[t] = lstm_cell_l2_enc(xs_2_enc[t], hs_2_enc[t-1])
        scope.reuse_variables()
```

Pattern Search

Goal: Build a trading pattern search function using Deep Learning. Given a sample trading pattern identify similar patterns in historical LOB data.

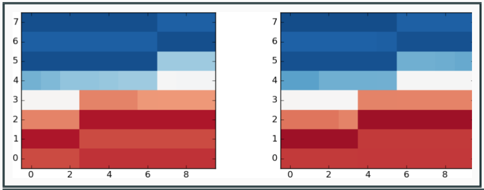
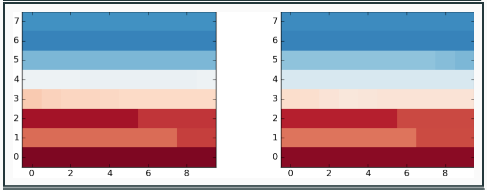
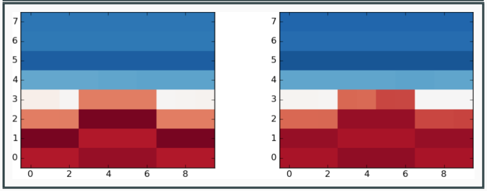
Methodology: Train the deep recurrent autoencoder on sequences of LOB data to define a mapping from LOB sequences to fixed length vector encoded LOB sequences:

$$X = \{x_1, x_2, \cdots, x_T\} \xrightarrow{encode} X_{enc} = h^e_{3T}$$

For all historical LOB data, $\{X^{(1)}, X^{(2)}, \cdots, X^{(K)}\}$ compute $\{X^{(1)}_{enc}, X^{(2)}_{enc}, \cdots, X^{(K)}_{enc}\}$.

Pattern Search: Define $\mathrm{dist}(X, Y) = d(X_{enc}, Y_{enc})$ where d is some metric on $\mathbb{R}^n$ with $n = \dim(h^e_{3T})$.

Sample Search Results (Small Data Set)

LHS: Search target
RHS: Search result

Time displacement
Large quantity bias

Generative Models I

It is useful to learn the distribution of LOB sequences and be able to generate samples from this distribution:

Backtesting: Generate large scale sample sets for systems testing

Transfer Learning: Augment historical data for illiquid contracts

Latent Variables: Learn driving variables for LOB markets

The recurrent autoencoder above does not learn data distributions and cannot generate samples of LOB sequences.

Instead:

Generative Models: generate examples like those already in the data

Generative Models II

We assume a probabalistic model for the data x in the form

$$p(x) = \int p_\theta(x \mid h) p(h) dh$$

for latent variables h and seek to choose parameters θ which maximize $p(x)$ over the data.

General Approach: Distribution $p(h)$ can be simple (e.g. Gaussian) so long as $p_\theta(x \mid h)$ is universal (e.g. neural network)

How to maximize p(x)?: Sample $h_k \sim p(h)$ and set $p(x) \simeq \frac{1}{n} \sum_k p_\theta(x \mid h_k)$

But: For most h, $p_\theta(x \mid h)$ will be close to zero so convergence is poor

Better: Sample h from $p_\theta(h \mid x)$ and set $p(x) = E_{h \sim p_\theta(z|x)}(p_\theta(x \mid h))$

Variational Autoencoder

We assume the market data $\{x_k\}$ is sampled from a probability distribution $p(x)$ with a 'small' number of latent variables h.

Assume $p_\theta(x, h) \quad = \quad p_\theta(x \mid h) p_\theta(h)$
where θ are weights of a NN/RNN. Then

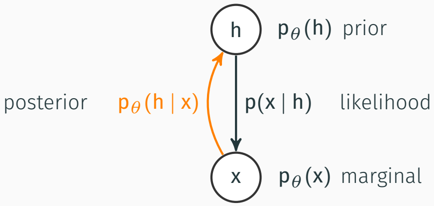
$$p_\theta(x) = \sum_h p_\theta(x \mid h) p_\theta(h)$$

Train network to minimize

$$\mathcal{L}_\theta(\{x_k\}) = \min_\theta \sum_k - \log(p_\theta(x_k)).$$

Compute $p_\theta(h \mid x)$ and cluster.

But $p_\theta(h \mid x)$ is intractable!



Variational Inference

Variational Inference learns an NN/RNN approximation $q_\phi(h \mid x)$ to $p_\theta(h \mid x)$ during training.

Think of q_ϕ as encoder and p_θ decoder networks: an autoencoder.

Then $\log p_\theta(x_k) \geq \mathcal{L}_{\theta,\phi}(x_k)$ where

$$\mathcal{L}_{\theta,\phi}(x_k) = \underbrace{-D_{\mathrm{KL}}(q_\phi(h \mid x_k) || p_\theta(h))}_{\text{regularization term}} + \underbrace{E_{q_\phi}(\log p_\theta(x_k \mid h))}_{\text{reconstruction term}}$$

is the variational lower bound.

Reconstruction term = log-likelihood wrt q_ϕ

Regularization term = target prior on encoder

$$\begin{tikzcd} \text{h} \arrow[d, "p_\theta(\text{x} \mid \text{h}) \ \text{likelihood}"] & p_\theta(\text{h}) \ \text{prior} \\ \text{x} \arrow[u, bend left, "\text{approx} \ q_\phi(\text{h} \mid \text{x})"] & p_\theta(\text{x}) \ \text{marginal} \end{tikzcd}$$

Questions?

References

M. Arjovsky, A. Shah, and Y. Bengio.
Unitary evolution recurrent neural networks.
arXiv:1511.06464v4, 2016.

A. Graves, G. Wayne, and I. Danihelka.
Neural turing machines.
arXiv:1410.5401v2, 2014.

S. Hochreiter and J. Schmidhuber.
Long short term memory.
Neural Computation, 9:1735--1780, 1997.