Algothon 2019

Khariton Gorbunov, Kevin Fung, James Baldock, Alex Capstick

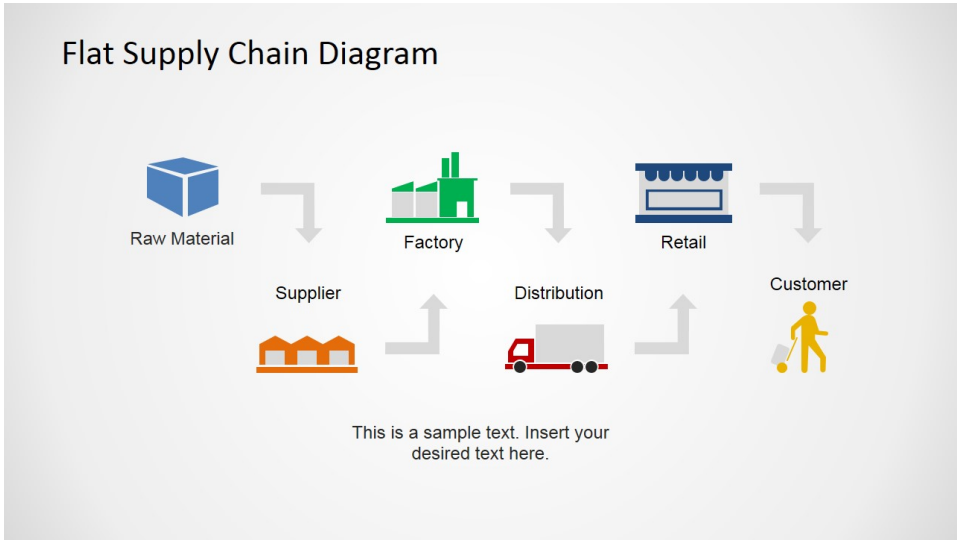
Mandatory information

Theme: Supply-side data leveraging
AWS used: No
Datasets used:

Outline

Rationale and motivation

Supply chain simplified



Supply prices are correlated with buyers' expenses, which in return influences their share prices.
Increasing silicon prices lead to increase in production costs of electronics companies, which then affects their revenue, driving the share prices down.

Real models have numerous layers of complexity

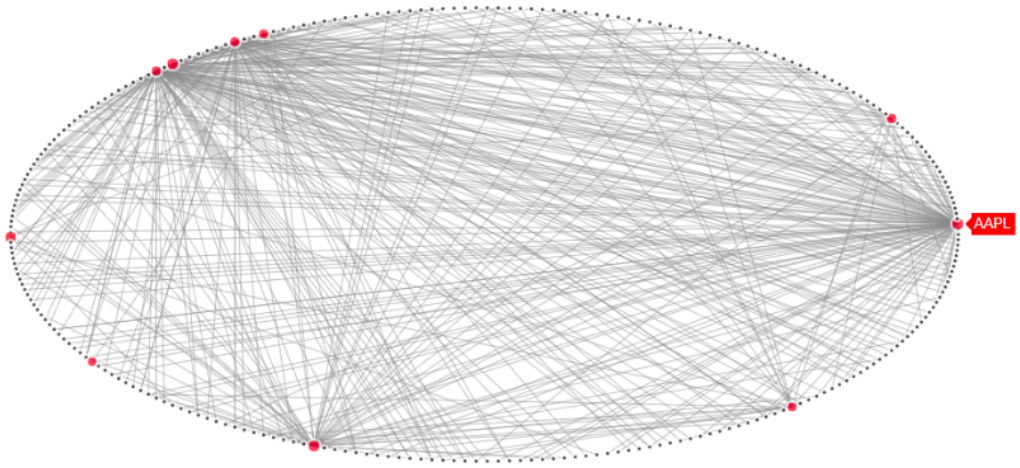


Figure: Quandl data

Trying to manually go along the chains is a noisy walk through a recursive mess.
Nevertheless, we believe useful data can be extracted to gain advantage in market.

Simplifying the model

Output gets more and more refined along production stage

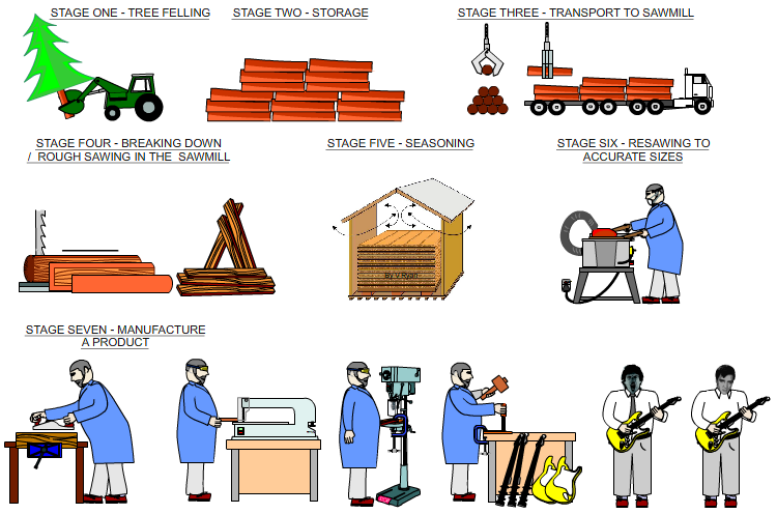


Figure: Manufacturing pipeline

The idea is to classify companies after crudeness of their goods, and build a model assuming that the financial complexity between two sectors is somewhat proportional to the crudeness difference of their products.

Model

Researchers and engineers have developed efficient tools to deal with high-dimensional, non-linear and complex data:

Artificial Neural Networks

with a slight modification...

Untangling the complexity

Neural networks can capture more and more complex features as one moves along the layers.
We believe that crude products do not belong in the same input layer as more refined products, for instance, logically there should be a higher complexity between prices of aluminium and laptops than CPUs and laptops, as those are further away from one another in the supply chain.

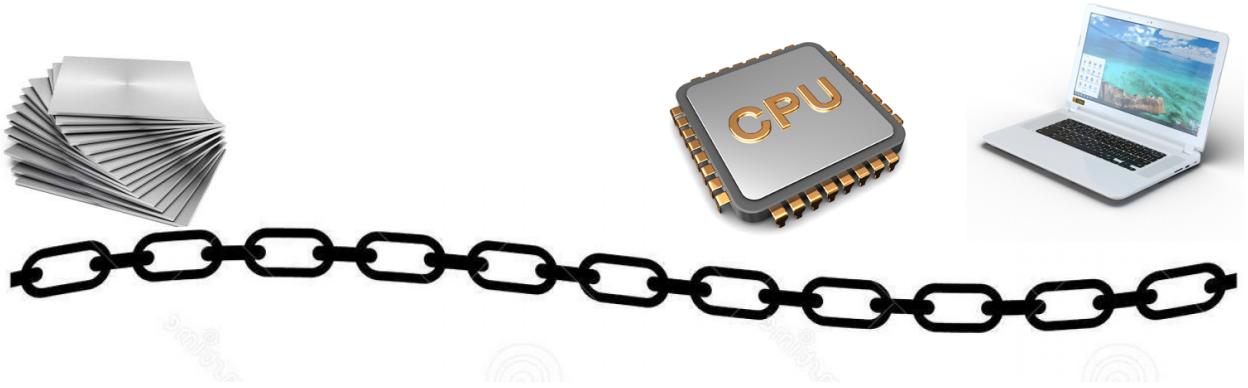
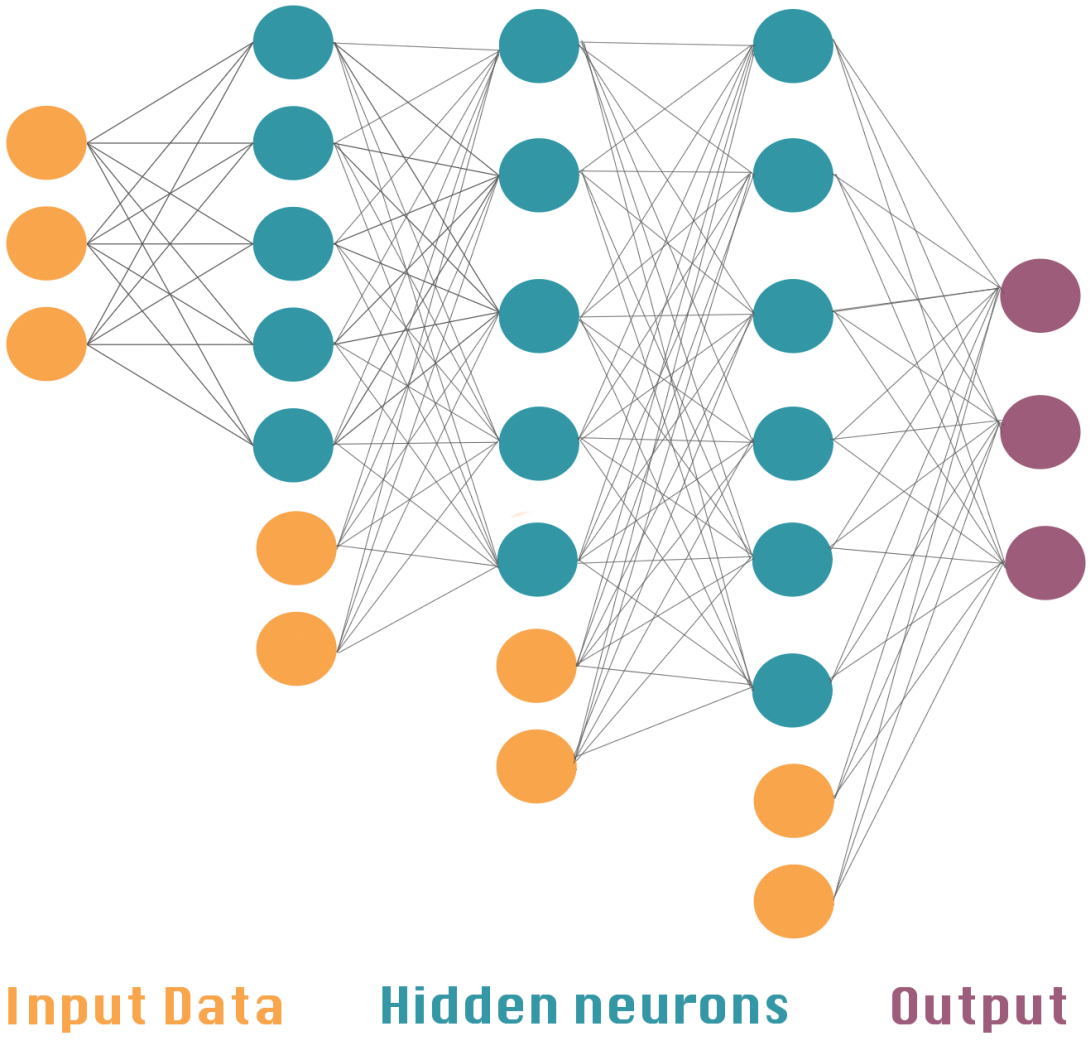


Figure: Laptop supply-chain

Proposed architecure

Hidden layers now have inputs



The inputs can be share prices, or any variation of those, where the neurons closer to the output take in data from companies whose output is more refined, allowing the model to capture the difference in complexity.
Define a crudeness index $C = \{x \in \mathbb{Z} | 0 \leq x \leq k\}$, and cluster the assets in descending order of crudeness, such that k is the cluster containing companies producing the most refined goods. Assets in cluster k will therefore be the ones we wish to add to our portfolio, which also means they will be used as the output of the neural net.

Since we have k clusters, it is sensible for the model to contain $k-3$ hidden layers, as the k-th cluster serves as output, 0-th cluster is the first input, keeping in mind the total number of clusters is $k-1$.
Inputs $\mathbf{x_i}$, $0 \leq i < k$ will go into i-th layer, where $\mathbf{x_k} = \mathbf{y}$, which is the output.
If we call the output from i-th layer $\mathbf{a_i}$, we can then write the general recursive equation connecting two adjacent layers:

$$\mathbf{a_{i+1}} = \sigma(W_i \mathbf{a_i} + \tilde{W}_i \mathbf{x_i} + \mathbf{b_i})$$

where $\sigma(x)$ is the activation function, W_i is the weight matrix connecting hidden neurons in layer i and $i+1$, $\tilde{W}_i$ is the weight matrix connecting $\mathbf{x_i}$ to hidden neurons in layer $i+1$, and $\mathbf{b_i}$ is the bias.

For the data, we have chosen to using prices, but we have made two big assumptions:

- Assume that there is a reaction (time) lag between two asset prices proportional to their difference in the cluster index.
- Assume (or ignore) any long term correlation between the prices of assets, generally for any time t greater than the biggest lag in the network.

Assumption justification

The reasoning behind the first assumption this is that price action moves with a finite speed through the supply chain. The second one has another underlying assumption, which is that any price-momentum is much smaller then the correlation between the assets we are looking at, which is unlikely to hold in most cases.
Using these assumptions, we must feed progressively more delayed data into the neural net, as we move down the crudeness number. So the data we are trying to predict the price of is the most recent data, while the inputs are progressively lagging.

Input function

Comparing absolute value of share prices throughout various industries and sizes is rather ambiguous. In addition, it does not allow us to capture the response to movement of one asset with respect to the other. Therefore, we chose a model that both senses the price change, as well as normalising it:

$$\partial P^i_{t,m} = \frac{{P_{t,m}}^i - {P_{t-1,m}}^i}{{P_{t-1,m}}^i}$$

Where superscript i is the crudeness number, t is a discrete subscript to index the time series and m indexes the particular asset in the i-th cluster.

The big question is how to determine the time-lag between each of the layers. For simplicity, we have chosen to take the previous candlestick in whatever time interval we are trading in, so if layer n is sampled at index t, layer $n+1$ is sampled at index $t+1$. Hence, we can write the inputs to the ANN as functions of price, crude number and time:

$$\mathbf{x_i} = \partial \mathbf{P^i_{t-k+i}}$$

Implementation

The team produced two implementations: one using Keras and Tensorflow, and the other one created manually.
The manual model just utilized basics of calculus and linear algebra, omitting back propagation and using numerical differentiation instead. Consider the following scalar field to be minimized with respect to all the weights and biases present:

$$J = \sum_i (\mathbf{y_i} - \tilde{\mathbf{y}}_\mathbf{i})^T (\mathbf{y_i} - \tilde{\mathbf{y}}_\mathbf{i})$$

Differentiating with respect to any weight component in any layer in the net yields: $\frac{\partial J}{\partial w_{ij}^L} = \sum_i 2(\mathbf{y_i} - \tilde{\mathbf{y}}_\mathbf{i})^T \frac{\partial \mathbf{y_i}}{\partial w_{ij}^L}$ Now approximate $\frac{\partial \mathbf{y_i}}{\partial w_{ij}^L}$ as following: $\frac{\partial \mathbf{y_i}}{\partial w_{ij}^L} \approx \frac{\mathbf{y_i}(w_{ij}^L + h) - \mathbf{y_i}(w_{ij}^L)}{h}$
The equation is the same for the biases. This proved to be a VERY inefficient method.

The addition of data to the hidden layers was easier to implement by just splicing the output$\mathbf{a_i}$ at layer i with $\mathbf{x_i}$, and making sure the matrix dimensions are correctly initialised, so input $\mathbf{z_{i+1}}$ to layer $i+1$ is:

$$\mathbf{z_{i+1}} = \begin{bmatrix} \mathbf{a_i} \\ .. \\ \mathbf{x_i} \end{bmatrix}$$

and the respective matrix is:

$$W_{i,mod} = \begin{bmatrix} W_i & \vdots & \tilde{W}_i \end{bmatrix}$$

Keras provides a framework for creating custom layers, so recurrent networks and auxiliary input are made possible. Using the concatenate function, we were able to create the model and optimise the mean square error cost function. The optimiser used was Adamax.
The algorithm was generalised to produce any sized network for flexibility and exploratory modelling. The network can be defined by two simple arrays, one describing the hidden neurons and the other one the hidden layer inputs.

Machine Learning

—

Data Wrangling

Mixing data sets was thornier than expected.
1. Company Names – Short forms and long forms (e.g. 'Toshiba' vs. 'Toshiba Corporation')
Solution(s): Tried using –

... WHERE LONG_FORM
LIKE '%' + SHORTFORM + '%'

Worked but was very slow, probably because it has to compile into a regex under the DBMS bonnet.

SELECT * FROM MyTable
WHERE CHARINDEX('word1', Column1) > 0

Would have thought query optimiser would have figured out to do this but we noticed a performance increase.

2. Data from alphavantage gave the needed TRBC to cluster the companies after the relevant industries, however that was a very slow process as the free API only allows 1 call per 13 seconds. When implementing this model, data costs should be considered a limitation.
3. Ticker symbols were not standardized across different exchanges. For instance, Toshiba is listed as TOSBF on Nasdaq and TYO:6502 on Tokoyo stock exchange. This example showcases the difference in both name and format, which makes combination of various datasets challenging.

4. Joining 3 tables was also very inefficient, yielding time complexities of order $O(n^3)$.
Should be done once and cached. Otherwise would like to use cluster indices and merged sort joins in order to very join quickly each time.

Evaluation