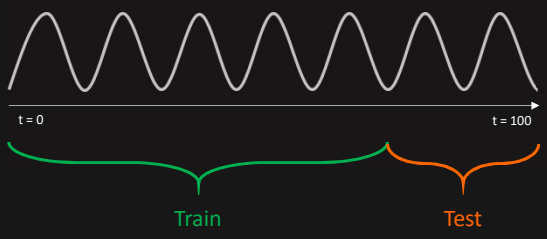
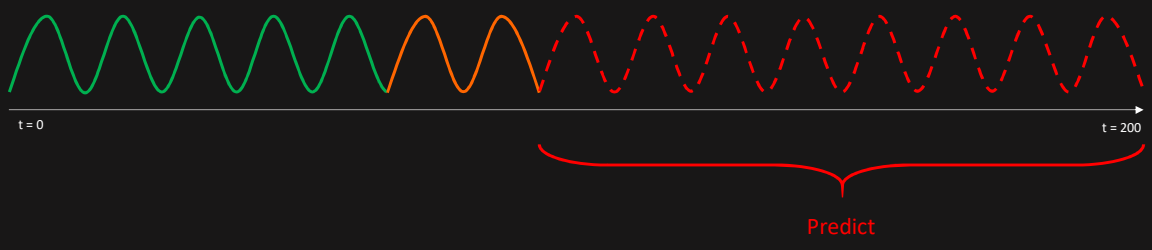
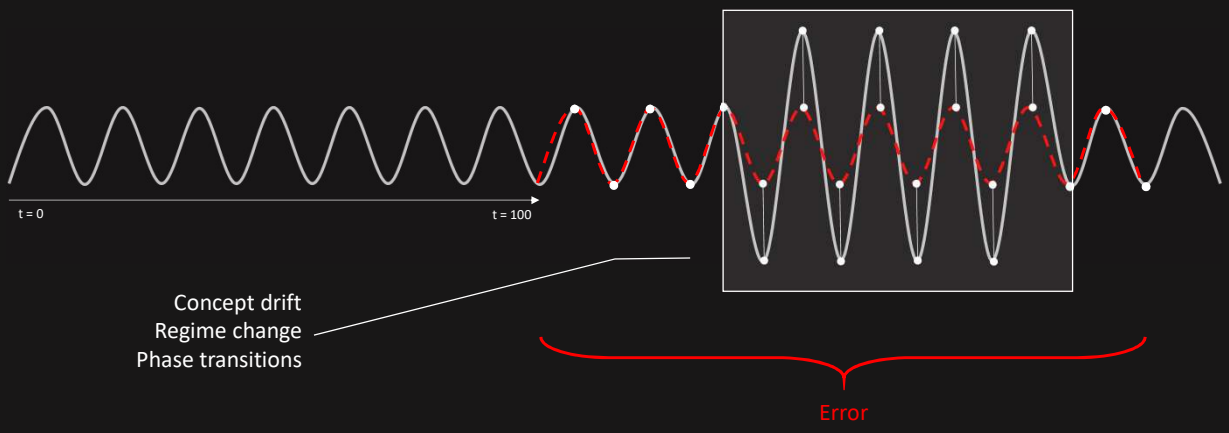
Some Tricks for Deep Learning in Complex Dynamical Systems

Stuart Gordon Reid
Chief Scientist
NMRQL Research

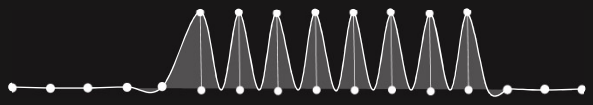


Perfect World





The Real World



Most machine learning methods assume that the data generating process is stationary …

This is quite often an unsound assumption

Sensor degradation

Sensor degradation caused by *normal wear and tear* or *damage* to mechanical equipment can result in significant changes to the distribution and quality of input and response data.

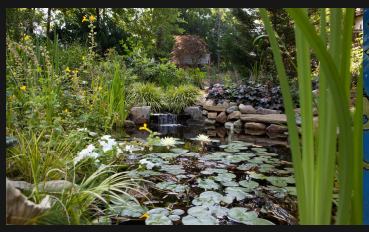
Brand new, shiny, latest tech with 100% working sensors

Aging, still shiny, yesterdays tech with ~85% okay sensors

Fixed up after 'minor' damage with ~70% okay sensors

Dynamical systems

Other systems are inherently nonstationary, these are called *dynamical systems*. They can be stochastic, adaptive, or just sensitive to initial conditions. Examples include, but are not limited to:



Habitats and ecosystems

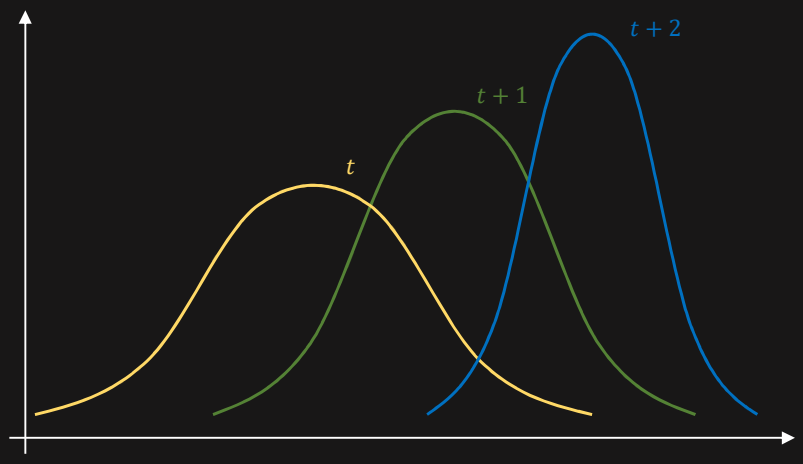
Weather systems

Financial markets

Data sampling

The first set of ‘tricks’ involve how to sample data to train our deep learning model on

Nonstationary (changing) environment

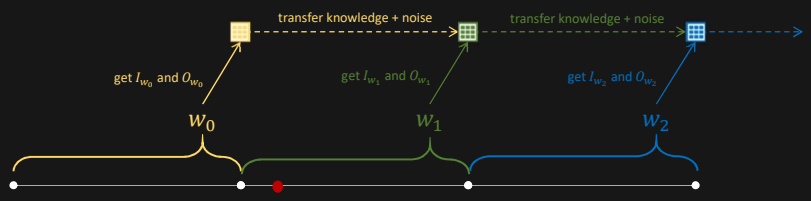


Supervised deep learning through time

Our model learns a function which can map some set of input patterns, $I_{t-w,t}$, to some set output responses, $O_{t-w,t}$.

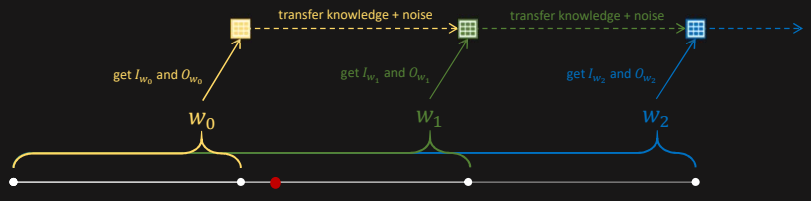
In this setting there is it is assumed that the relationship between I and O is stationary over the window so there is one f to approximate.

The choice of window size (how much historical data to train the model on) poses a problem as it can be either optimistic or pessimistic.



Fixed window size

- \- Pessimistic data sampling
- \+ Adapts to change quickly
- \- Less data to train on
- \+ Faster (less data)
- \- Inefficient data usage
- \- Hard with large models



Increasing window size

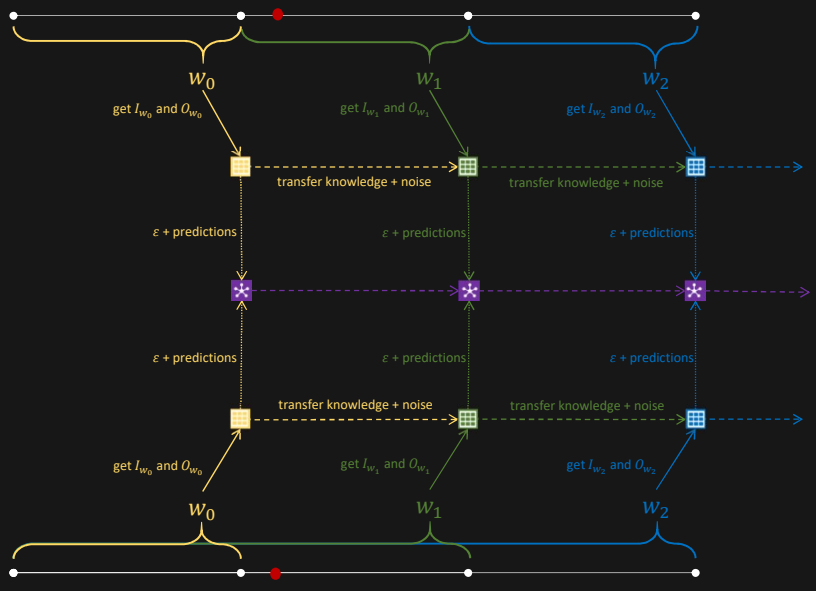
- \- Optimistic data sampling
- \- Adapts to change slowly
- \+ More data to learn from
- \- Slower (more data)
- \- Most data is *irrelevant* ~ poor model performance

Static ensemble over fixed window sizes

To remove the choice of window size, we could try to construct a performance-weighted ensemble of models with different windows.

The assumption here being that the model with the most relevant window size will perform the best out-of-sample. This is reasonable.

The challenge with this approach is that it increases computational complexity which, in some dynamic environments, is infeasible.



Static ensemble over different window sizes

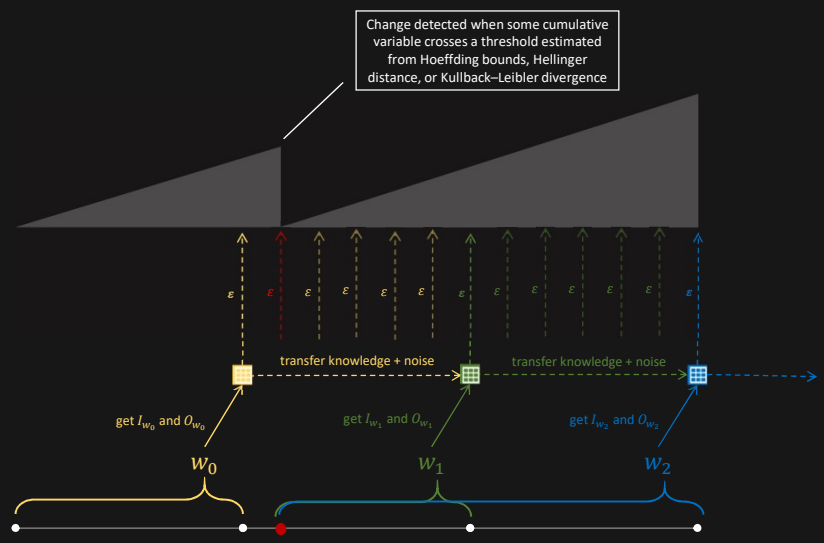
- ± Mixed and pessimistic and optimistic sampling
- ± Mixed fast and slow training times (more & less data)
- \+ Can adapt to change quickly
- \- Increased computational complexity and runtime

Change detection & Hypothesis tests

Another option is to try and determine the *optimal* window size using change detection tests and chained hypothesis tests.

Change detection tests are typically done on some cumulative random variable (means, variances, spectral densities, errors, etc.).

Hypothesis tests measure how likely two sequences of data have been sampled from the same distribution. It directly tests for stationarity.



Change detection tests

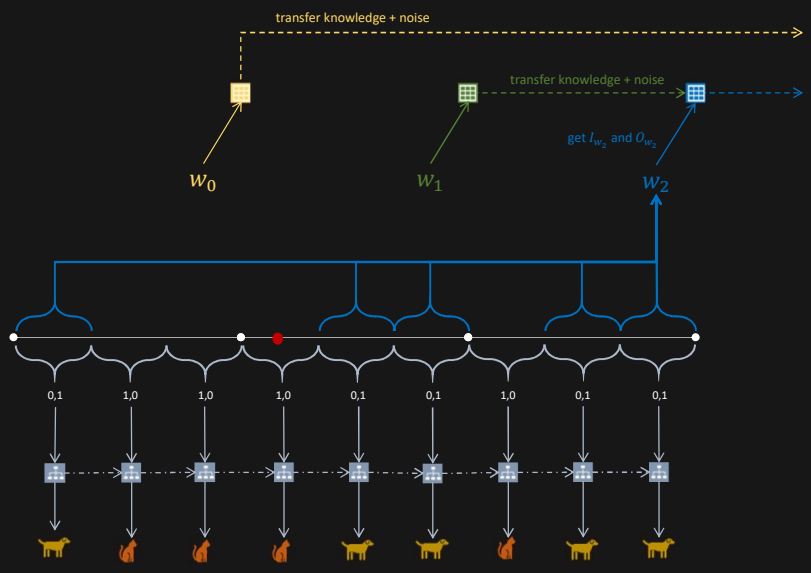
+ A first principled approach to optimal window sizing
+ Computationally efficient
+ Neither optimistic not pessimistic if CDT and HT results are correct
+ Adapts to change when change occurs, otherwise just improves decisions
- Sensitive to CDT threshold and p-values in HT

Time series subsequence clustering

Another approach is to see the entire history of a given time series as an unsupervised classification problem and to use clustering methods.

The goal is to cluster historical subsequence's (windows) into distinct clusters based on specific statistical characteristics of the sequences.

One good old fashioned approach is to use k nearest neighbours. The only challenge can be the assignment of labels through time.



Time series subsequence clustering model

+ Also a first principles approach to data sampling
+ Neither optimistic nor pessimistic if clustering is correct / good enough
+ Data need not be sampled contiguously (NB!).
- Sensitive to performance of the clustering algorithm
- Computationally expensive and practically difficult.

Data augmentation

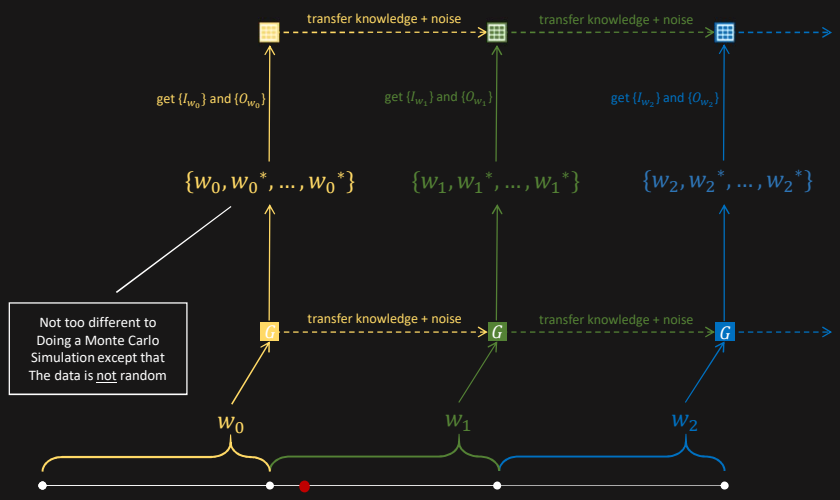
The second set of tricks involve how we can augment and
Improve the data we have sampled to improve learning

Data generation

A challenge of pessimistic fixed window sizes and optimal window size estimation is not having enough data in the selected window.

If the relevant window contains too few patterns then training large complex models becomes infeasible (they won't converge).

One approach to dealing with this problem is to train a simple generator and use it to amplify the training data for larger models.



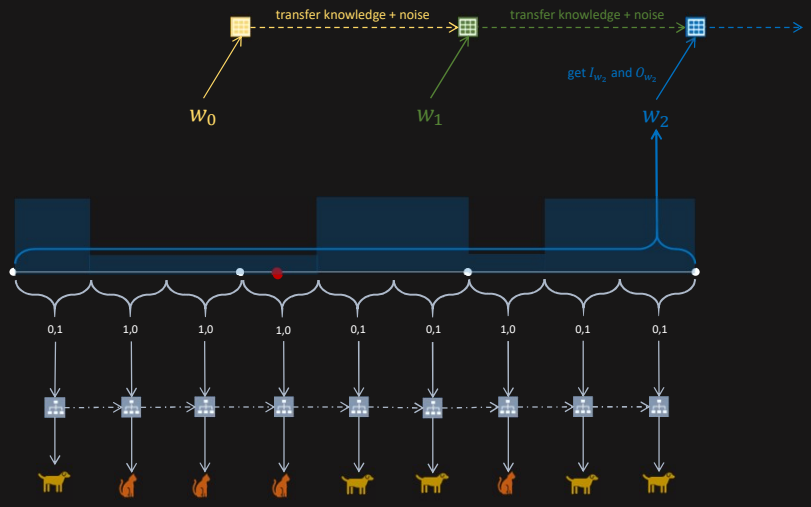
Data Generation

+ Enough data to fix complex models esp. RL agents
- Still pessimistic data usage (can be overcome with CDT & HT tests)
- Generator could easily produce nonsense if the data is too noisy or the generator is too simple.
+ It's pretty dope

Cluster (similarity) based sample weighting

The time series clustering approach is equivalent to setting the 'importance' or sample weight of patterns sampled from clusters other than the one we are in to 0 and those from the cluster we are in to 1.

This can be made more robust by measuring the *probability* of the most recent data points belonging to any given cluster and then weighting the patterns according those probabilities.



Sample weighted time series subsequence clustering

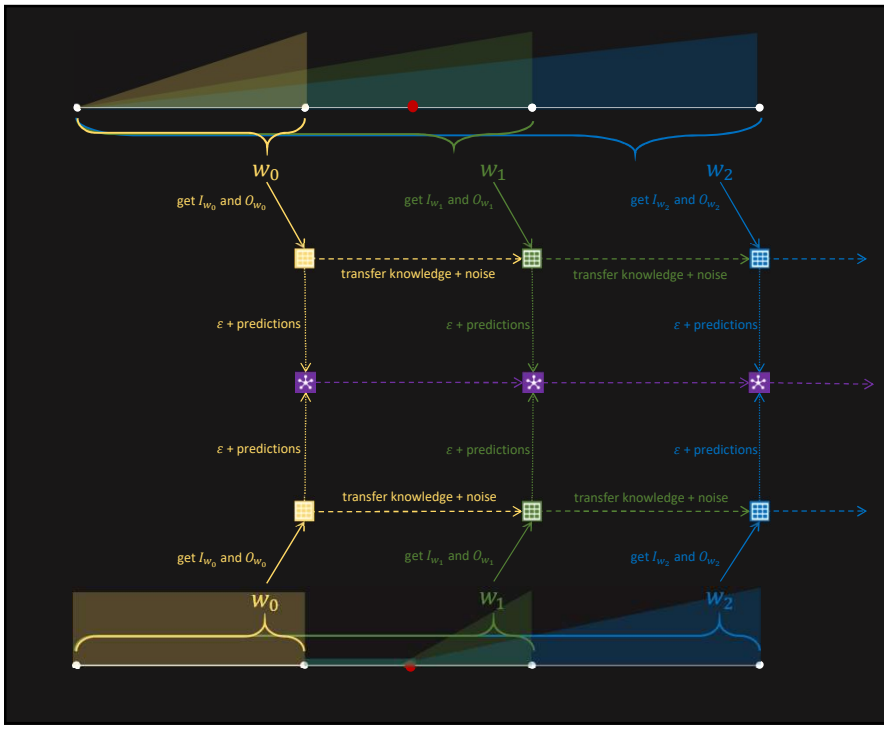
+ Also a first principles approach to data sampling
+ Neither optimistic not pessimistic if clustering is correct / good enough
+ Relevant needs not be sampled contiguously.
+ Less sensitive to performance of the clustering algorithm
+ No need to keep different models for different clusters just add more noise if the cluster changes (sees all)

Neural network boosting **

Once we have sampled our data, we can further improve upon our selection by weighting the importance of patterns in that data.

Boosting can be modified to 'work around' change points. This is done by initially weighting *recent* patterns higher than *older* patterns and then weighting *easier* patterns higher than *harder* patterns.

The assumption here is that the hardest patterns will be those which are *least similar* to *recent* patterns so they should be down weighted.



Neural network boosting

+ Neural network can learn where patterns start to differ from recent ones.
+ Delicate balance between weighting and difficulty
- Computationally expensive

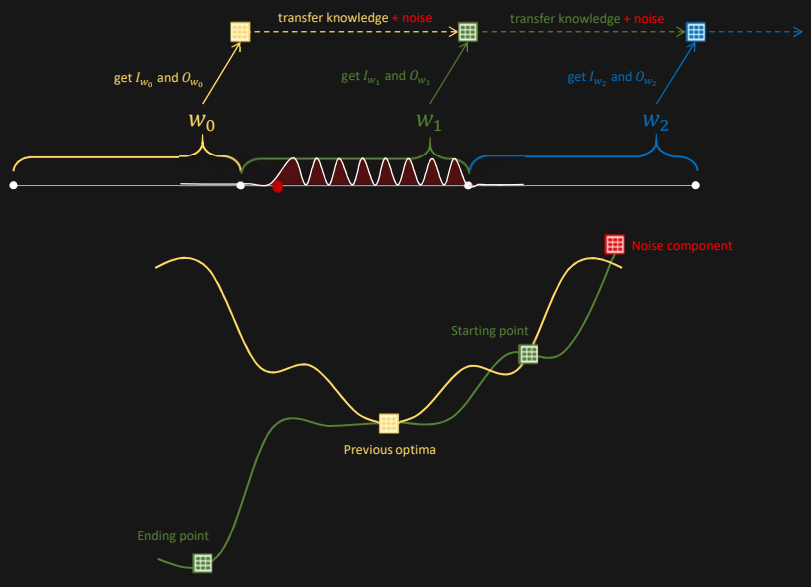
Model augmentation

The third set of tricks involve modifications to the model which allow it to adapt to change faster in

Error-based re-initialization

The first trick has already been shown in every slide so far. When the model moves from time t to some time $t + x$ the knowledge learnt at time t should be transferred but only to the extent that it is relevant.

This can be done by partially re-initializing the neural network based on the out-of-sample error observed between time t and time $t + x$.



Error-based re-initialization

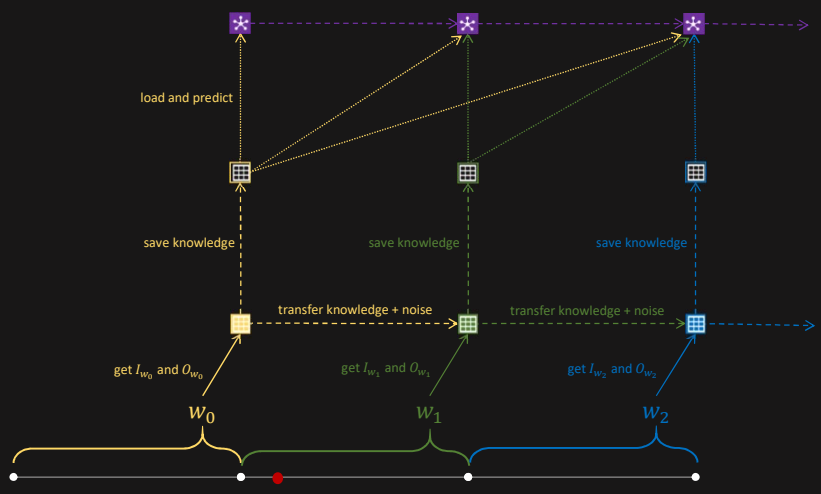
+ When regime changes occur, we end up re-initializing more.
+ A regime change is the same as the loss surface changing so previous optima are no longer optimal, but they may still exist on a plateau.
- Sensitive to error scale.
- If the noise is spurious (not a regime change, just a "rough patch") a lot of previously learnt knowledge may be destroyed!

Snapshot ensemble over time

In the previous example E_2 is effectively seeing w_2 twice (from the top and bottom NN) and w_0 and w_1 once (from the bottom NN).

Given a fixed window size, and snapshots of models trained on those windows this can be approximated in a computationally efficient way.

At each point in time the ensemble loads historic snapshots, generates predictions with them, and weights them inversely to how old they are.



Snapshot ensemble over time

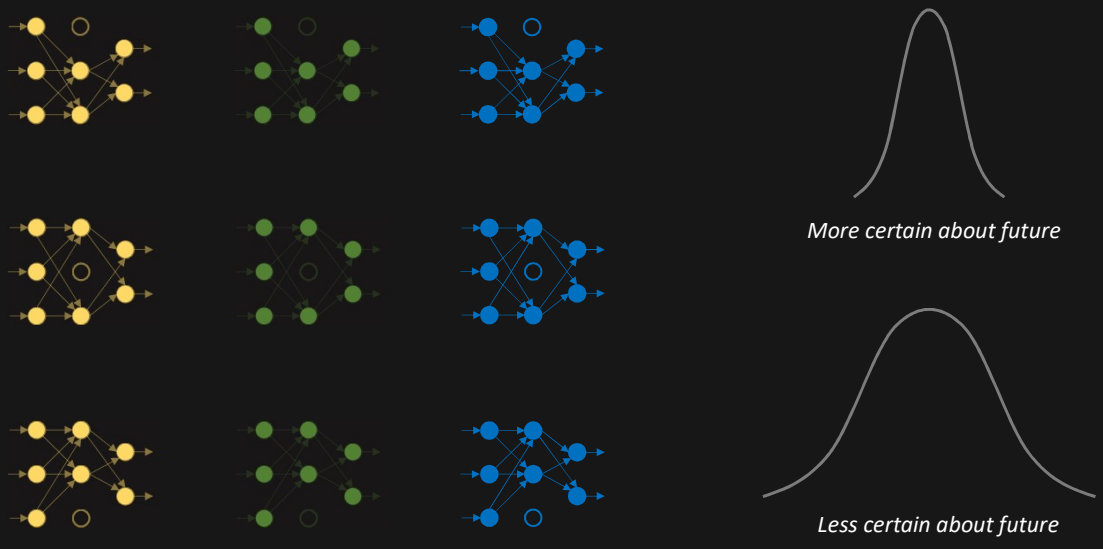
- ± Mixed and optimistic and pessimistic sampling
- \+ Only fast training since we always have less data
- \- Less data to train on
- \+ Can adapt to change quickly
- \+ Equivalent computational complexity and runtime
- \- Involves making some assumptions about relevance of old models

Approximate Bayesian Models using Stochastic Regularization Techniques

Another challenge is that the error terms which we are using the identify regime changes / phase transitions are lagged.

One trick for turning point estimate time series prediction models into Bayesian models is to perform inference with dropout enabled.

Different ‘paths’ through the neural network will be activated resulting in different predictions. This variance is a useful error metric.



A little bit of background

Some information about what we are up to at NMRQL Research

Current methodology

We developed a general-purpose, multivariate time series prediction platform which allows us to easily create thousands of collaborative supervised online learning agents which collectively encode self-organizing investment strategies which adapt alongside the market.

As of March 2018 we have about 1,200 deep learning algorithms deployed in production which collectively process 19,000 independent time series and produce 100's of GB of information a week. This will grow because innovation is the bedrock of our investment philosophy.

Research projects

Everything mentioned above is implemented to some degree in our framework so most day-to-day research focusses on incremental improvements - new models, better datasets, scaling up – as well as the application of our algorithms to new and exciting problems.

A major focus for us right now is the incorporation of deep reinforcement and active learning strategies into the framework to help improve individual algorithm and ensemble adaptiveness. We are also quite interested in deep learning interpretability research!