# Pre-Processing of Time-Series Data for a Neural Network

October 14, 2016

## 1 Overview

### What We Have

Let $x(t)$ denote the demand at time-step $t$. Let $T$ be the most recent time-step, and assume we have access to the historical demand data

$$x(1), x(2), \ldots, x(T-1).$$

### What We Want to Predict

Our first goal is to predict $x(T)$, but it is also a goal to predict $x(T+1)$ and in fact $x(T+s)$ for an arbitrary horizon $s$. There are three ways to do this[1]:

- train a network to predict $x(T+s)$
- train a network to predict all of $x(T), x(T+1), \ldots, x(T+s)$
- train a network to predict just $x(T)$ and iterate using previous predictions as historical data until we've found $x(T+1)$, then $x(T+2)$, and eventually $x(T+s)$.

### How We Predict

We want our forecasting to work as follows. We choose a vector $y$ that describes previous demand in some way, and we hope that when we input such a vector into our neural network, the output is a good indicator of demand in the next time-step (or the next $s$ time-step units, as discussed above).

To train the network, we will input vectors $y_1, y_2, \ldots$ that describe historical demand, meaning the demand in subsequent time-steps is known. Say a given input $y_i$ describes demand up to time-step $k$: we want the output of the NN to be approximately $x(k+1)$ (or $x(k+s)$, as discussed above).

[1] From the article *Time Series Sampling Using Neural Networks* at https://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/slides/timeseries.pdf

If the output from the neural network after inputting this $y_i$ does not match $x(k+1)$ (or $x(k+s)$), then we backpropogate (change the NN's weights) so that the NN will respond correctly to this input. If the output is correct, then we train on another data vector. A given vector may need to be inputted and backpropogated several times before the network responds correctly to all training vectors.

Once the NN responds correctly to all the input vectors describing historical data, it should be a good predictor for future data!

In practice, one must reserve some amount of historical data as "test" data, which is not part of the training but to which, one would hope, the network can respond correctly. If the NN doesn't respond correctly to the test data, the model is either overfitted or underfitted, and more training or more intelligent training is required. Including more nodes than necessary gives the potential for overfitting (and requires more computational work). Including too few nodes can lead to an under-fitted model.

## Crafting Input $y$

### Indexing $y_i$: to what does $i$ correspond?

The input vectors $y_1, y_2, \ldots$ each describe some period of historical data.

Say $y_i$ includes historical data up to time-step $k$, as above. Then $y_i$ is the best suited input vector for making predictions about time-step $k+1$, although it can also make predictions about future time-steps $(k+2), \ldots, (k+s)$.

It may be useful to associate $y_i$ with time-step $k$ by assuming that $i = k$, and that the index of the input vectors $y_1, y_2, \ldots$ all indicate the (chronological) first time-step that their output is able to predict. For example, the historical descriptor that incorporates the *most current* demand information will always be used to make actual (not training) predictions, and will be called $y_T$; the descriptor that informs predictions at time-step $t$ is $y_t$.

Now we can assert that we have exactly one input vector per time-step, which together make up

$$y_1, \ldots, y_T.$$

.

### Complexity of NN Algorithm vs. Dimension of $y_i$

In general, neural networks do $O(W^3)$ operations per iteration of backpropogation, where $W$ is the total number of nodes. The dimension of the inputs $y_i$ determines the number of input nodes (nodes in the first layer of the neural network). Input nodes may be a small minority of total nodes (most nodes are in "hidden layers"), so inputting more detailed data may not add much calculation per iteration, but it can certainly delay convergence by adding complexity to the hidden funtion that the neural network is approximating. As a result, more (possibly exponentially more) iterations are needed before the NN makes accurate predictions with higher-dimensional input.

## 2 Sampling Demand for NN Input

### Uniform Sampling: Most-Recent Samples

The previous $k$ samples of historical demand are $x(T-1), \ldots, x(T-k)$, and we can certainly create an input vector $y_T$ whose components are those values, based on which the NN would output a prediction for $x(T)$. In general, the input vector whose output approximates $x(t)$ can be called

$$y_t = \big(x(t-1), \ldots, x(t-k)\big).$$

This is the most straightforward neural network forecasting scheme, in which the neural network is trained to predict demand at a given time-step $t$ based on the demand during each of the previous $k$ time-steps. Here, $k$ is referred to as the *embedding dimension*.

### Uniform Sampling: Undersampling

Suppose demand follows yearly periodicity, meaning that the demand in previous years at a given date significantly correlates with demand in the current year on that date. It would be wise to choose an embedding dimension $k$ such that the previous $k$ time-steps stretch back more than one year; suppose $r$ time-steps stretch back exactly one year. In a uniformly-sampled input scheme, increasing $k$ to $r$ increases the input dimension by the same amount, which adds nodes and delays convergence of backpropogation.

Instead of simply setting $k = r$, suppose we sampled every other time-step, so that

$$y_t = \underbrace{\big(x(t-2), x(t-4), \ldots, x(t-(r-2)), x(t-r)\big)}_{\frac{r}{2} \text{ components}}.$$

To limit the input to exactly $k$ components, sampling could be distributed arbitrarily scarcely, yielding

$$y_t = \underbrace{\big(x(t-\frac{r}{k}), x(t-\frac{2r}{k}), \ldots, \overbrace{x(t-r)}^{}\overbrace{x(t-\frac{kr}{k})}\big)}_{k \text{ components}}.$$

Note that $k$ is no longer the "embedding dimension," but is rather the length of the "delay line," an arbitrary sequence of time-steps at which demand is being used as the input to a neural net whose output approximates subsequent demand.

### Non-Uniform Sampling: Delay Line

Let $(d_n)_{n=1}^{\infty}$ consist of strictly increasing integers. Then suppose the $k$-dimensional input for which a NN would approximate demand at time $t$ were

$$y_t = \big(x(t-d_1), x(t-d_2), \ldots, x(t-d_k)\big).$$

For example, suppose $(d_1, d_2, d_3, \ldots) = (1, 2, 3, \ldots)$, i.e. $d_i = i$. Then sampling with delay line $d_n$ restricted to $k$ components would be uniformly sampling most-recent samples, with an embedding dimension of $k$.

If, instead, $d_i = 2i$, meaning $(d_1, d_2, d_3, \ldots) = (2, 4, 6, \ldots)$, then the input would be under-sampled at every other time-step.

In fact, $d_i = d(i)$ constitutes a valid delay line for any strictly increasing function $d : \mathbb{Z} \to \mathbb{Z}$.

A non-linear increasing function $d(i)$ will include sparser samples further back in time, prioritizing more recent information in its predictions (e.g. $d(i) = e^i$ or $d(i) = i^2$).

# 3 General Demand Descriptors for NN Input

## Change of Notation

Regardless of the preferred scheme for sampling demand $x$, the input with which the NN approximates $x(t)$, which is called, $y_t$, will include some description of previous demand $\big(x(t-1), x(t-2), \ldots\big)$ and so on, in each of its components. In general, let the input $y_t$ consist of $k$ components, whose $i$'th component is called $x_i(t)$, so that

$$y_t = \big(x_1(t), \ldots, x_k(t)\big).$$

- Sampling uniformly from the most recent samples yields

  $$y_t = \big(x(t-1), x(t-2), \ldots, x(t-k)\big),$$

  so

  $$x_i(t) = x(t-i).$$

- Undersampling uniformly to consider a span stretching back $r$ time-steps yields

  $$y_t = \big(x(t-\frac{r}{k}), x(t-\frac{2r}{k}), \ldots, \overbrace{x(t-\frac{kr}{k})}^{x(t-r)}\big),$$

  so

  $$x_i(t) = x(t-\frac{ir}{k}).$$

- Sampling from a delay line $d_n$ yields

  $$y_t = \big(x(t-d_1), x(t-d_2), \ldots, x(t-d_k)\big),$$

  so

  $$x_i(t) = x(t-d_i).$$

## Periodic Memory Terms

There seems to be a trade-off between the dimension $k$ of the input $y_t$ and the historical "window" considered in a given prediction. However, this sense of trade-off is disrupted by the fact that each input can describe arbitrarily old demand by undersampling, either uniformly or on a delay line.

An idea seemed to be that any periodicity in demand could be incorporated into a prediction by including a time-step from the previous "period" in a delay-line. This idea was discussed when $r$ denoted the number of time-steps spanning one year, and the oldest demand sample included as a component of $y_t$ was $x(t-r)$.

Instead, periodicity at a scale/period $p$ can be measured by the inner product (denoted $\langle \cdot, \cdot \rangle$) of demand $x(t)$ and the $p$-periodic function $sin(\frac{2\pi}{p}t)$:

$$\left\langle x(t), sin(\frac{2\pi}{p}t) \right\rangle = \sum_{\text{oldest}_p \leq i \leq t} x(i) sin(\frac{2\pi}{p}i).$$

Note that the values of $i$ over which this summation is calculated determines the window considered relevant for analysis of periodicity at a given scale. At different scales, different spans of historical data may be relevant or irrelevant for analyzing periodicity. Let the oldest time-step worth considering for periodic trends at scale $p$ be called $\text{oldest}_p$.

For example, a shift in sales strategy 1.5 years ago may have yielded weekly patterns that didn't previously exist (a shift from wholesale to retail inducing higher demand on Fridays than Mondays, e.g.), even as monthly demand trend has stayed somewhat constant over the company's 10 year history (demand is always high on the first of the month, e.g.). If $p_{\text{year}}, p_{\text{month}}$, and $p_{\text{week}}$ denote the number of time-steps corresponding to one year, one month, and one week, respectively, then it may be wise to choose $\text{oldest}_{p_{\text{week}}} = 1.5p_{\text{year}}$, and $\text{oldest}_{p_{\text{month}}} = 10p_{\text{year}}$.[2]

If each component of input $y_t$ described demand's periodicity at a different scale, then let the scale reflected in the $i$'th component be called $p_i$, and

$$x_i(t) = \left\langle x(t), sin(\frac{2\pi}{p_i}t) \right\rangle.$$

In other words, the NN will be trained to approximate demand at $x(t)$ with the $k$-dimensional input vector $y_t$ defined as:

$$\begin{aligned} y_t &= \big(x_1(t), \ldots, x_k(t)\big) \\ &= \big( \left\langle x(t), sin(\frac{2\pi}{p_1}t) \right\rangle, \left\langle x(t), sin(\frac{2\pi}{p_2}t) \right\rangle, \ldots, \left\langle x(t), sin(\frac{2\pi}{p_k}t) \right\rangle \big), \end{aligned}$$

where each component $x_i(t)$ represents periodicity of demand at scale $p_i$.

[2]The term $\text{oldest}_p$ will be incorporated into the general trend bases $c_i$ such that $c_i(t) = 0 \; \forall \; t \leq \text{oldest}_{p_i}$

## General Convolutional Memory Terms: Not Strictly Periodic Trends

### Convolution Definition

For two functions $f$ and $g$, the convolution at time $t$, denoted $f(t) * g(t)$, is defined as

$$f(t) * g(t) = \sum_i f(t)g(t-i).$$

Convolution is very similar to the inner product, with a few differences. First, rather than implicitly starting summation at the "beginning" of the functions and ending at the "end" as in an inner product, convolution specifies the point from which summation moves "outward". Second, one of the functions is reversed in the convolution (symmetry makes it arbitrary which function is reversed); since the functions that will be convolved here are always a "signal" and a "basis" function, and the directionality of the bases can be chosen, this is not such a constraint; this happens to be convenient, given that analysis of a given time-series trend naturally works backwards from the point of most recent information.

As with inner product, the ideal "depth" of the convolution, meaning the range of values away from $t$ over which the summation is performed, may be subject to the trend being considered.

It's also worth noting that for any two function $f$ and $g$, convolution is commutative (and associative and distributive), and the order of terms may be changed without notice in this document.

### Change of Notation

The concept of an inner product is associated with the relationship between two geometrical objects, which, in this case, would be the demand time-series and the "trend" functions (like sinusoids). However, what we're really measuring is the correlation of demand with a trend function *at a given time point* (each of which has its own span during which the trend is being examined; namely, time-steps$[oldest, t]$). For this reason, it may be more convenient, when representing this particular data in terms of basis functions, to reference the data's convolution with those bases (rather than inner product).

For that reason, it's more appropriate to represent the correlation of demand $x(t)$ with a trend as a *convolution* of the demand time-series data with a trend function, which, in general, will be indexed as $c_i(t)$ (in the above example regarding periodic functions, $c_i(t) = sin(\frac{2\pi}{p_i}t)$).

For the $i$'th trend being analyzed in the data, then, define a trend function $c_i(t)$, and let

$$x_i(t) = c_i(t) * x(t) = \sum_\tau x(t-\tau)c_i(\tau).$$

As with periodic trend descriptors, let

$$\begin{aligned} y_t &= \big(x_1(t), \ldots, x_k(t)\big) \\ &= \big(c_1(t) * x(t), \ldots, c_k(t) * x(t)\big) \end{aligned}$$

**Examples of Convolutional Memory Terms**

- Periodic trend functions (sinusoids) are generally symmetric, so convolution and inner product yield the same result, meaning periodic memory terms can be determined as above by using

  $$c_i(t) = sin(\frac{2\pi}{p_i}t).$$

  Using this choice of memory function $c_i$, we can write the $i$'th component of $y_t$, which is the $i$'th descriptor of demand $x_i(t)$, as

  $$x_i(t) = \langle x(t), c_i(t) \rangle = c_i(t) * x(t).$$

- To simulate a "delay line" $(d_n)_{n=1}^{\infty}$, let

  $$c_i(t) = \begin{cases} 1 & \text{if } t = d_i \\ 0 & \text{otherwise} \end{cases}.$$

  In this case, it can be checked that

  $$\begin{aligned} x_i(t) &= c_i(t) * x(t) \\ &= \sum_{\tau} x(t-\tau) c_i(\tau) \\ &= c_i(d_i) x(t-d_i) \\ &= x(t-d_i). \end{aligned}$$

  Thus, as previously described in this case, the input $y_t$ becomes

  $$y_t = \left(x(t-d_1), \ldots, x(t-d_k)\right).$$

- To quantify "Exponential Trace Memory," we can use

  $$c_i(t) = (1-\mu_i)\mu_i^t$$

  for some $\mu_i \in (-1, 1)$. In this case, a component $x_i(t)$ of the NN input $y_t$ becomes a weighted sum of previous values of demand $x$

  $$\begin{aligned} x_i(t) &= \sum_{\tau} x(t-\tau) c_i(\tau) \\ &= \sum_{\tau} x(t-\tau)(1-\mu_i)\mu_i^{\tau}. \end{aligned}$$

The rate of decay $\mu_i$ determines how much relative importance is accorded to more recent terms: a smaller $\mu_i$ means that recent samples of demand contribute much more to $x_i(t)$ than older samples.

Each component $x_i(t)$ is a weighted moving average of the previous demand values, and the components together in $y_t = \big(x_1(t), \ldots, x_k(t)\big)$ each represent total historical demand with different biases for recent data.

Note that convolution need not be performed at each time-step: the convolution of a basis function and demand at time $t$ can be defined by the recurrence relation

$$x_i(t) = (1 - \mu_i)x(t) + \mu_i x_i(t - i),$$

and thus each of the components of the vectors $y_t = (x_1(t), \ldots, x_k(t))$ can be efficiently calculated recursively over time $t$ using a dynamic programming algorithm.

- Let

$$c_i(t) = \begin{cases} \binom{t}{d_i}(1 - \mu_i)^{d_i+1}\mu_i^{t-d_i} & \text{if } t \geq d_i \\ 0 & \text{otherwise} \end{cases}$$

where the delay $d_i$ is the "center" of the exponentially decaying bias, and $\mu_i$ represents the rate of decay. The type of memory being quantified by this convolution can be called "Gamma Memory".

This is a generalized exponential trace memory strategy, in which only demand samples older than a given delay $d_i$ are considered, and in which a binomial coefficient gives preference to terms surrounding the center, depending on how far displaced the center is from current data. That is, bases of this form have a "center" and a "radius" in which demand is considered; when the center is very old, the radius widens.

Note that if $d_i = 0$, then this family of bases becomes the "exponential trace memory" family of bases. Also note that if $\mu_i = 0$, then this becomes the "tapped delay line" family of bases (if we allow $0^0 = 1$).

**FACT CHECK THIS EVENTUALLY** The statistical assumptions surrounding this model, and its relationship to the binomial distribution, seem to be that "shit happens" (as it does, which is with a binomial distribution of likelihood), and if we specify a scale (likely conceivable as the expected value of time-between-occurrences), we can measure the contribution to demand of shit happening at that scale. If impulses effect the system, and their frequency is not periodic chronologically, but is rather periodic in a "how many rolls until I get a 5" type of way, then these memory terms will likely capture that behavior.

**MORE PRECISELY** This is the binomial distribution, which usually is plotted and conceptualized with respect to the variable that here is

$d_i$. In particular $c_i(t)$ is, exactly (time $(1-\mu_i)$ for some reason), the probability that an event, with independent probability of happening $\mu_i$ per sample, doesn't happen exactly $d_i$ times out of $t$ total samples. With respect to $t$, this function attains its maximum at $t \approx \frac{d_i}{(1-\mu)}$. That is, the solution to

$$\max_t c_i(t)$$

is

$$t \approx \frac{d_i}{(1-\mu)}.$$

Convoluting against $c_i(t)$ thus measures demand events $\frac{d_i}{(1-\mu)}$ time-steps ago, and the surrounding time-steps. It is a smoothing, delaying function.

The end-result is that, given the assumption that $d_i$ things have happened, and that their probability of happening in a given time-step is $\mu_i$, this basis function measures the demand on the time-steps when those things are most likely to have started happening.

**A note on Dimensionality**

The basis functions mentioned for any representation strategy (called $c_i(t)$), when taken together, make up an infinite set $\{c_1(t), c_2(t), \ldots\}$, or a "family" of functions. For each strategy, the corresponding family is such that, when taken together, there are "enough" functions that we can infer everything we need to know about the actual demand $x(t)$ just by observing its convolution with each function. In particular, there are "enough" functions in each family that we can infer *everything* about the actual demand $x(t)$.

That is, given every convolution

$$(x_1(t), x_2(t), \ldots) = (c_1(t) * x(t), c_2(t) * x(t), \ldots),$$

we should be able to reconstruct the demand $(x(1), x(2), \ldots)$ exactly.

In fact, since each family of functions is infinite, and the historical demand goes back arbitrarily but is in fact finite, we would hope to reconstruct demand with only finitely many convolutions. In practice, a careful choice of basis functions should allow a relatively accurate representation of demand with very few convolutional terms.

The number of terms necessary to accurately reconstruct the data *in the best possible choice of basis* has to do with the data's innate dimensionality. An assumption of forecasting is that the demand, which contains many historical data points, actually has a much lower innate dimension. That is, relating historical demand over $n$ time-steps would take an $n$-dimensional vector. If that data could be represented with $k$ convolutional memory terms, from which we can fairly accurately reconstruct it, then the innate dimension is approximately $k$, which is useful if $k << n$.

If the historical data is very "smooth," then it is easier to imagine data's low dimensionality, but even pathological-looking data may be low-dimensional, as long as it follows consistent trends.

Note that a given basis's ability to efficiently reconstruct a function (its ability to "compress" the data) is a good indicator of its descriptive usefulness as input to a forecasting NN, even though the NN will not technically be reconstructing the function.

The ideal basis need not all come from one straightforwardly-indexed family, but may be a hybrid from many different families of functions. Perhaps a function could be represented by many different families of functions, and then an analytic algorithm (like SVD) or a numerical one (like K-SVD) could select the bases from each family that most highly correlate with demand (in an analytical sense, or a numerical sense such as under the constraint of a sparse representation).