Machine Learning II
Week 11 Lecture – 8th January 2019
Neural networks and time series data

Contents

Introduction to time series

We have not yet considered time series data in ML1 or ML2.

The have been a few examples, where the predictor variable was time, such as the motorcycle impact data, but these were analysed as if time were a spatial variable. When smoothing the curve, the values to the right of a fixed point were treated exactly the same as values to the left.

There are three types of data which are first impressions very similar but have important differences.

A time series data are observed at specific time points and the series is ongoing. E.g. share prices.

A signal data were obtained in a specific time frame and is now considered as fixed data. E.g. spoken syllables.

An image is a signal observed in two dimensions.

In each case there is a concept of nearness/separation in time or space between two elements.

In a time series, when modelling the data at the current time t we can only use data values that correspond to the same or previous time points $t, t-1, t-2, \ldots$

The same types of NN models can be applied to all three data types and the neural network literature often blurs the difference between them. For example time delayed NNs can be applied to a image data.

We will consider **image data** in detail next week.

Types of time series

The simplest form of time series is a univariate time series.

$$x_1, x_2, \ldots, x_n$$

Usually the time interval between observations is constant.

Examples:		
financial	share prices, index prices (DAX, Dow Jones)	daily
economic	unemployment rates, interest rates	monthly
meteorological	temperatures	daily
	Global CO2 Levels	yearly
engineering	acoustic signals	millisecond

Multidimensional time series, here with 2 dimensions

$$(x_{11}, x_{12}), (x_{21}, x_{22}), \ldots, (x_{n1}, x_{n2})$$

Eg x_{t1}=DAX Index x_{t2}=Dow Jones Index.

In some cases x_{t1} may be used as a predictor for x_{t2} but not vice versa.
Eg. minimum x_{t1} and maximum x_{t2} temperatures at a weather station. The minimum temperature on day t is usually known about 10 hours before the maximum temperature.

A counter example is daily pollution measurements of NOx and CO at a given location. Usually both measurements are taken at the same time.

Multidimensional time series can have fixed covariates (fixed meaning not time dependent). E.g. Share prices for 500 different companies listed at 10 different stock exchanges. Fixed covariates could be stock exchange and the industry each company belongs to.

Repeated measurements are similar to a multidimensional time series with fixed covariates, but:

- the number of time observations is usually small
- the time interval between measurements does not have to be constant
- and the time at which the measurements are observed is not the same for each series.

Repeated measurements example: In a study, babies' weights are recorded at birth, 1, 3, 6 and 12 months.
Fixed covariates are age of mother, length of pregnancy, ...
The measurements will not be taken when the baby at exactly 3 months etc, but during an appointment with a nurse around that time.

Time Series and Supervised learning

A basic univariate time series will usually be its own outcome variable.
Using the data available today, what is the predicted value tomorrow?
For predictor variables $x_1, x_2, \ldots, x_t$ the outcome value is x_{t+1}. This is called one-step-ahead prediction.
If the aim is to predict $x_{t+1}, \ldots, x_{t+k}$, (or alternatively just x_{t+k}) then it is called k-step-ahead forecasting.

Time series prediction can be simplified to a classification model such as
$y_{t+1} = 1$ share price increases by more than 10 cents
$y_{t+1} = 0$ share price stays within $\pm$ 10 cents
$y_{t+1} = -1$ share price decreases by more than 10 cents.

An application which has had success with fitting neural networks to time series data is real time voice recognition: E.g. *Alexa* in stand-by mode continually processes the acoustic signal and responds when someone says “Alexa".

Forecasting, training & test data

You cannot just randomly choose a proportion of the data to be the test data, and fit using the training data! — You’ll end up using data from the future.

The training data will be the first epoch of data up to a time point T. You can fit the NN on all the data at once or using an “online” training method starting with x_1, then x_1 and x_2, then x_1, x_2 and x_3 etc.

The validation data is then a second epoch to asses and fit the hyperparameters. If a test data set is needed this will be the last epoch.

Usually the training data epoch will be much larger than the validation and test epochs.

Note that although this approach is very common in data analysis and model development, it will slightly undervalue the predictions of the NN time-series model applied to the test data set!

If you rely on a NN to predict daily share prices when working in a bank, you would develop a good NN model using training, validation and test data sets, but when using the NN you *would* update your NN-Parameters every day, and *would* update your hyperparameters regularly (maybe once a week).

Classical approach to time series

The subject of time series in statistics is quite old. The approach assumes there is some underlying random component ϵ_t, which is an infinite time series:

$$\ldots, \epsilon_{-t}, \epsilon_{-t+1}, \ldots, \epsilon_{-2}, \epsilon_{-1}, \epsilon_o, \epsilon_1, \epsilon_2, \ldots, \epsilon_t, \ldots$$

Each value is independent and identically distributed e.g. $N(0, \sigma^2)$.

The **autoregressive model AR(p)**

The data we observe today is a linear function of the recent past and the random component.

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \cdots + \alpha_p x_{t-p} + \epsilon_t$$

We assume that any constant c in the time series has been subtracted out.

Two very popular autoregressive models are the AR(1) model with $\alpha_1 = 1$, which is a *continuous random walk*,

$$x_t = x_{t-1} + c + \epsilon_t$$

and the AR(1) model with $|\alpha_1| < 1$,

$$x_t = \alpha x_{t-1} + \epsilon_t$$

This second model is called a stationary time series, and so it follows that the expected value of x_t with t long in the future is zero.

The moving average MA(q) Model is similar to the AR model, but now today's value is a linear function of the recent random component.

$$x_t = \epsilon_t + \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + \beta_p \epsilon_{t-p}$$

The **ARMA(p,q) Model** is a combination of an AR(p) and an MA(q) model.

The ARMA model fits a surprisingly wide class of time series data, once any long term components and the seasonal components are known and subtracted out of the time series.

Example is monthly rainfall r_t, where the long term trend is (probably) constant c and the seasonal effects for each month can be easily estimated, s_t then $r_t = c + s_t + x_t$.

x_t is the residual time series and is fitted well by an ARMA model.

Autocorrelation

Standard correlation is measured between two different data variables x and y.

Autocorrelation measures the correlation within a time series. The autocorrelation with lag k measures the correlation between x_t and x_{t-k} i.e. delayed by k time points. If for a fixed lag k the autocorrelation does not vary over time, then the time series is called *stationary*.

For a stationary time series the autocorrelation is often written as ρ_k and plotting ρ_k against k is an informative diagram.

If the autocorrelation structure is known, this can help, when training a NN.

Classical time series analysis is useful for analysing the structure of the data, but is poor at forecasting.

In the rainfall example, the accuracy in fitting the long term trend and the seasonal component, is far more important than forecasting the residual time series.

NNs are the exact opposite. We get very little understanding of the model structure but are good at predicting the data into the future.

Time delay neural networks TDNN

There is nothing wrong with predicting x_{t+1} using just x_t, in which case you restructure your training data from $x_1, x_2, \ldots, x_T$ to:

predictor	outcome
x_1	x_2
x_2	x_3
$\ldots$	$\ldots$
x_{T-1}	x_T

The neural network is then fitted using the standard feed-forward fully connected NN.

Usually we want to use more than one value at each time point. The simplest time delay model is of order 1, where the current and the previous value are used in the prediction. This can be done by restructuring the data and using two input nodes.

predictor		outcome	Diagram
x_1	x_2	x_3	
x_2	x_3	x_4	
$\ldots$	$\ldots$		
x_{T-2}	x_{T-1}	x_T	

Comments

The TDNN can easily be adapted to order q by having q delay operators.

The memory of the process is fixed at q steps. If an extreme outlier occurs at x_t then it will have no further influence on the predictors after time $t + q + 1$.

This model is similar to the Moving average MA(q) time series model.

TDNNs are often applied to signal data and images, because of the shift-invariance property of the model. This means that given a specific pattern in the data the same output is obtained regardless of when that pattern is encountered.

For time series this is important, because we do not want out output to be dependent on the starting time. We want to predict an upcoming stock market crash regardless of whether we started collecting data on 5 November 2012 or 6th September 2013

In signal data we might want to differentiate between a recorded voice saying “zwei”, “zwo” or “drei”, and this should be independent from the when the signal started recording.

For images we want to identify a cat regardless of where in the image it is.

Recurrent neural networks RNN

The TDNN assumes that the data is time shifted and input as a new input node.

A similar model is the fully recurrent NN. The time series data is input via one node but the output value of the hidden nodes are fed back into themselves

There are two standard methods to drawing this: the time folded representation, Diagram

and the time unfolded (or unrolled) representation:Diagram

The fully RNN model is very similar to the Autoregressive AR(1) time series model.

With the TDNN we need to specify in advance how long the memory in the system lasts.

The fully RNN model leaves a little bit of information from x_1 in the model for ever.

If the feedback loop has a high weight then the memory of x_1 is quite strong, if the feedback loop has a low weight then the memory of x_1 is quite weak. Importantly the NN training will find the best value for this weight.

Time delay NNs were the first type of **convolutional NN** to be developed.

You can define a time delay NN as a convolution, which very briefly means that the same weights are applied to the inputs but shifted along by one each time, so if:

$$z_1 = w_1 x_1 + w_2 x_2 + w_3 x_3,$$

then $$z_2 = w_1 x_2 + w_2 x_3 + w_3 x_4$$

and $$z_3 = w_1 x_3 + w_2 x_4 + w_3 x_5$$

etc.

More on this next week

Long short-term memory (LSTM) process

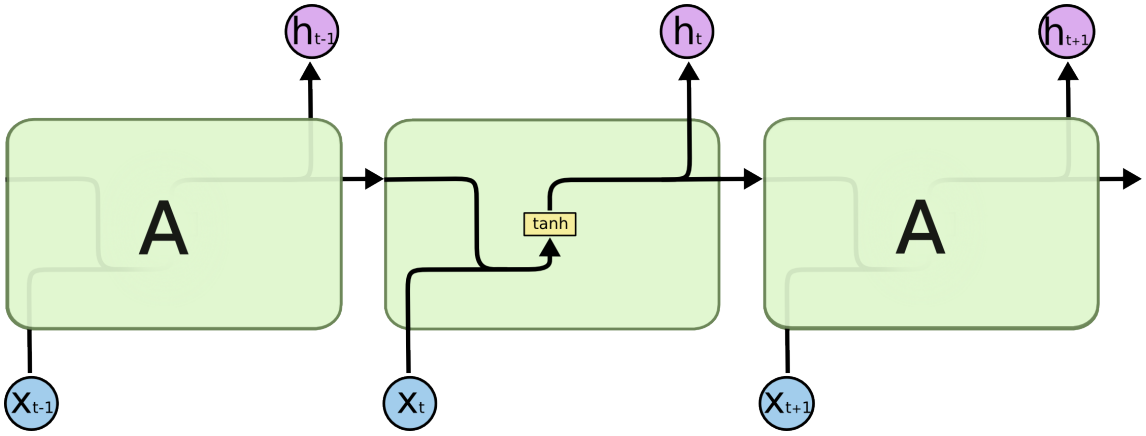
There are many extensions to time series NNs. The most important is the LSTM process. In principle when there is a long term dependency in the data a fully RNN can cope with this, but often the optimiser will not find this low cost region in the parameter space.

Example: firework sales in Berlin have an obvious high autocorrelation at a lag of 365 days. People know this, finite response neural networks don't and will struggle to find the dependency.

All of the NNs we have considered sofar in ML2 consist of basic nodes/neurons. Each neuron takes a weighted linear combination of inputs, applies an activation function and outputs the result.

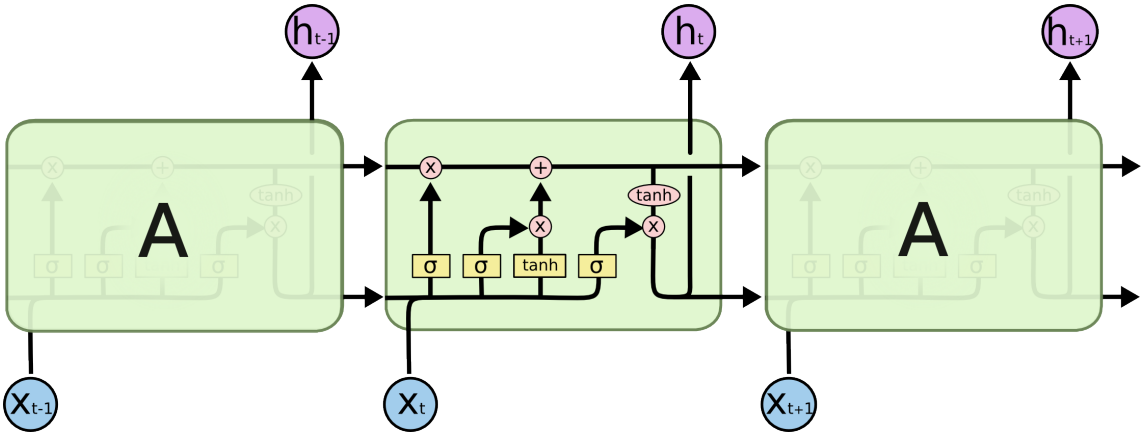
We will now replace the neurons with blocks, which can do multiple operations before outputting the result.

The fully RNN above can be represented with the following block A (repeated)



The notation is a bit different, the value fed to the next layer is h_t, A is the module at the 1st hidden layer, it takes the input from x_t and the previous module A, multiplies by weights and applies the $\tanh$ activation function.

The classic configuration for an LSTM module is



This is complex, but The important features are:
There are two channels the upper and lower channel.
Data in the upper channel can easily move on with minor changes.
The lower channel is highly processed before it either meets the fast channel or the output channel.
The fast lane allows dependencies to propagate a long way down the chain.

These NN diagrams were obtained from a website by Christopher Olah called *Understanding LSTM Networks* at
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
which explains this type of NN very clearly.

Please read this article as homework.