Predictive Analysis on Time Series

Lucas Bechberger*

Advisor: Anja Bachmann†

Karlsruhe Institute of Technology (KIT)
Pervasive Computing Systems – TECO
*Lucas.Bechberger@student.kit.edu
†Bachmann@teco.edu

Abstract. This paper gives an overview over different methods for predicting time series. General concepts regarding time series prediction are introduced. Then, both linear methods (with focus on ARIMA) and non-linear methods (with focus on Multi Layer Perceptrons) are presented. Approaches to combine different methods are discussed, including a hybridization of ARIMA and MLP and the more general approach of ensembles. Moreover, in addition to the widespread single-point prediction, the concept of prediction intervals is presented and practical approaches of estimating such prediction intervals (namely ensemble-based approaches and conformal predictors) are discussed.

Keywords: Predictive Analysis, Time Series, Time Series Prediction, Artificial Neural Network, ARIMA, Prediction Interval, Multi Layer Perceptron, Ensemble, Conformal Predictor

1 Introduction

Predicting the future is a task being carried out by humans every day, significantly influencing their behavior. For instance, the weather forecast tries to predict the weather conditions for the next hours and days. One might base the decision whether or not to take an umbrella or the decision whether or not to plan a picnic for the weekend on this forecast.
Hawkins even argues that predicting the future is not just an important task, but that

> "[p]rediction is not just one of the things your brain does. It is the *primary function* of the neocortex, and the foundation of intelligence." [11]

Besides this rather philosophical aspect, making accurate predictions is crucial in many applications – knowing something about the future usually is a major advantage. Wong [31] argues that every important business decision is based on some sort of forecast, and Silipo & Winters [24] note that a 1% increase in prediction accuracy can lead to operational cost savings of 10 million pounds. Therefore, accurate forecasts are of real business value.

Some forecasting examples include electricity prices, water flows, sunspot activity, tourist arriving patterns, exchange rates, stock prices or the number of airline passengers.
In all of these examples, the basic problem is to use past observations of a variable to predict its future values. For the exchange rate example, this can be formulated as “given the USD-EUR exchange rates of the last two weeks – what will be tomorrow’s USD-EUR exchange rate?”. If the forecast made in this scenario is sufficiently accurate, it can lead to considerable savings. For instance, an international company can optimize the scheduling of their international wire transfers based on this forecast, in order to minimize the conversion loss.
Also for the other examples, one can easily see that they use past values to predict future ones. Again, accurate predictions can be used for an optimization of some kind, may it be scheduling like in the exchange rate example or an airline’s decision whether or not to buy additional planes. Usually, these optimizations will help to lower costs or to increase profits.
The examples illustrate the need for accurate predictions and their importance to business decisions. This paper will explain in more detail how the general problem of time series prediction can be approached.

The remainder of the paper is organized as follows: Section 2 contains the problem statement and gives an overview over general concepts regarding time series prediction. Section 3 presents linear methods in general and the ARIMA approach in more detail. Section 4 covers nonlinear methods and focuses on Multi Layer Perceptrons. Section 5 discusses different approaches to combine multiple methods. Section 6 motivates the use of predictive intervals and shows different methods of approximating them. Section 7 summarizes the key insights of this paper and gives an outlook on open research questions.

2 Background

There exists a wide variety of models used for predicting time series. This section’s purpose is both to state the general problem of time series prediction and to give an overview of general concepts regarding time series prediction.

The mathematical elements dealt with in time series prediction are time series. A *time series* is defined as a sequence of chronologically ordered values of the same variable measured at different points in time. An example would be the temperature at a certain location, being measured every day at noon for the last 7 days, with values of 23.3 °C, 26.4 °C, 26.2 °C, 26.3 °C, 22.1 °C, 24.2 °C and 26.6 °C.
The problem of time series prediction can informally be stated as extrapolating the given time series one or multiple time steps into the future. For the given temperature example this could be stated as “given this time series of temperature data, what are the most likely temperature values to follow?”

Let Y be the time series variable to predict. Let Y_t denote the value of Y at time t. A general assumption is that the time interval between two measurements of Y is kept constant. Then, the problem of time series prediction can be defined as follows:

$$\text{Given } Y_{t-l}, ..., Y_t, \text{ predict the value of } Y_{t+k}.$$

Here, l is called the *lag* (or the *window size*) and denotes how many time steps into the past are considered, whereas k is called the *lead time* and denotes the number of time steps ahead of t to be forecast. $k = 1$ gives a *one-step ahead prediction* (or *short-term prediction*). Any greater value of k gives a so-called *multi-step ahead prediction* (or *long-term prediction*).
Although multi-step ahead prediction is a very interesting research topic, it tends to be more complicated than one-step ahead prediction. Therefore, this paper will put its focus on one-step ahead prediction.

Let $\hat{Y}_{t+k}$ be the prediction for Y_{t+k}. Then,

$$Y_{t+k} = \hat{Y}_{t+k} + \epsilon_{t+k}$$

with ϵ_{t+k} being the prediction error. Clearly, for a good prediction, the goal is to minimize ϵ_{t+k}. An alternative way of formulating the problem is the following: Find a function f such that

$$\hat{Y}_{t+k} = f(Y_{t-l}, ..., Y_t) \quad with\ |Y_{t+k} - \hat{Y}_{t+k}|\ being\ minimized.$$

How f will look like is mainly determined by the forecasting method being used.

A *forecasting method* is a general approach to solving the time series prediction problem. Most forecasting methods include the step of fitting a *forecasting model* against the data (i.e. optimizing some parameters). In [4], Chatfield points out the distinction between "forecasting method" and "forecasting model" and argues that the terms should not be used interchangeably: There are forecasting methods that involve fitting a forecasting model (e.g. ARIMA, ANNs), but there are also forecasting methods that do not make use of a forecasting model (e.g. the naive method, where $\hat{Y}_{t+1} = Y_t$, does not contain any parameters to be optimized).

Of course, if there already is a good model of the application domain and the underlying process generating the observed time series, this model can and should be used to predict future values of the time series. Amjady [2] calls this a *white box* approach.
If, however, no such model is available (which often means that the underlying process is unknown or poorly understood), the more general forecasting methods presented in this paper can be used [33]. This is called a *black box* approach.

So far, only *univariate* time series prediction has been discussed, where all parameters of f are past values of Y. In *multivariate* time series prediction, however, f can also take other parameters as input [5]:

$$\hat{Y}_{t+k} = f(Y_{t-l}, ..., Y_t, X^{(1)}, ..., X^{(n)})$$

The temperature example from the beginning of this section can be extended to a multivariate prediction problem, if in addition to previous temperature values also the current wind strength is used for the prediction.
Multivariate time series prediction tends to be much more complex than univariate time series prediction. Moreover, it is being less applied in practice [8]. Therefore, we will focus on univariate time series prediction for the scope of this paper.

Yadav & Toshniwal [32] and Wong [31] list four major components that can be used to characterize a time series:

– **Trend Component** T: The general movement of the time series mean over a long period of time. Often, a linear trend is assumed.
– **Cyclic Component** C: The long term oscillation of the time series. May be periodic or not.
– **Seasonal Component** S: A systematic, mostly calendar-related oscillation.
– **Random Component** R: Random fluctuation in the data. Mostly assumed to be normally distributed with zero mean.

The value of the time series Y at time t can then be expressed by multiplying these components:

$$Y_t = T_t \cdot C_t \cdot S_t \cdot R_t$$

According to Wong [31], the random component R can further be divided into two parts: a pseudo-random component created by a deterministic process, and a component entirely created by stochastic noise. Furthermore, Wong argues that it is possible to model and hence predict the pseudo-random component.

If there is no trend component, a time series can be called *stationary*. More precisely, a time series is called *stationary* if its mean and variance are not changing over time. For some forecasting methods (e.g. ARIMA) it is important that the time series under analysis is made stationary before applying the method itself.

An important concept when talking about time series is *autocorrelation*. It describes the correlation between different measurements of the observed variable, i.e. the correlation of Y_t and Y_{t-m} for some $0 \leq m \leq l$. This autocorrelation is usually high for small lags (i.e. small values of m) and for lags m corresponding to the length of a cycle or season [5].
Note that the whole field of time series prediction is based on the assumption that there is some autocorrelation in the data, i.e. that future values depend on past values to at least some extent.

To evaluate the performance of a forecasting method, different accuracy measures can be used. De Gooijer & Hyndman [8] provide a thorough list of such accuracy measures, including the most commonly used mean squared error. As it is common in the field of machine learning, the accuracy measure is computed on a test set separate from the training set which is exclusively used for training the model. Usually, the test set for time series prediction tasks will be the most recent historical data available, whereas the training set will consist of all older data.

Forecating methods can be subdivided into two classes: *linear* methods and *nonlinear* methods. Linear methods assume that the forecast $\hat{Y}_{t+k}$ can be calculated as a linear combination of the lag values and other values, if applicable. These methods will be further explored in Section 3. Nonlinear methods do not assume such a linear decomposition of the time series being analyzed. They will be covered in more detail in Section 4. A detailed overview over different linear and nonlinear methods is given by De Gooijer & Hyndman [8].

3 Linear Methods

3.1 General Idea

Linear methods assume that any value of the time series can be expressed using only linear relationships. Therefore, especially all linear regression methods are linear methods. Although the theoretical assumption of linearity in the data is not always satisfied, linear methods still might yield good performance in practice.
Ruta et al. [23] name two simple linear forecasting methods: the *naive method* where

$$\hat{Y}_{t+1} = Y_t$$

and the *simple moving average* where

$$\hat{Y}_{t+1} = \frac{1}{l+1} \sum_{i=0}^{l} Y_{t-i}$$

Exponential smoothing is another simple method, given by the following equation:

$$\hat{Y}_{t+1} = \alpha \cdot Y_t + (1-\alpha) \cdot \hat{Y}_t$$

where α is the *smoothing factor* which determines the weighting of the actual value in relation to the prediction.
Note that all three of these simple methods are special cases of the more general ARIMA method [8] which will be presented in the remainder of this section.

3.2 ARIMA

"ARIMA" is an acronym for **A**uto **R**egressive **I**ntegrated **M**oving **A**verage. The ARIMA method has been the most popular linear method of the last decades and is considered to be a well established time series prediction method. Basically, an ARIMA model consists of an autoregressive and a moving average component. ARIMA models were introduced by Box & Jenkins [3]. In order to fully motivate the ARIMA formulas, we will follow their step-by-step explanation.

Let B be the *backward shift operator*:

$$\begin{aligned} B \cdot Y_t &= Y_{t-1} \\ B^2 \cdot Y_t &= Y_{t-2} \\ &\vdots \end{aligned}$$

Then the *backward difference operator* ∇ can be defined as:

$$\nabla Y_t = Y_t - Y_{t-1} = (1 - B) \cdot Y_t$$

Furthermore, let a_t be a random variable at time t which is created by a white noise process.

Assuming that the time series Y is stationary with mean μ, it can be expressed with a *linear filter* model. Values of the time series are expected to be distributed around the mean, based on the white noise process:

$$Y_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \ldots$$

The assumption of stationarity of course requires $\sum_{j=0}^{\infty} |\psi_j| < \infty$. A more compact way of expressing the above formula is

$$\begin{aligned} Y_t &= \mu + \Psi(B) \cdot a_t \\ \textit{with} \quad \Psi(B) &= 1 + \psi_1 B + \psi_2 B^2 + \ldots \end{aligned}$$

An *Autoregressive (AR)* model tries to model the deviations from the mean, i.e.

$$\tilde{Y}_t = \mu - Y_t$$

It expresses the current deviation as linear combination of previous deviations plus the current white noise term and therefore can be written as

$$\tilde{Y}_t = \phi_1 \tilde{Y}_{t-1} + \ldots + \phi_p \tilde{Y}_{t-p} + a_t$$

with p being called the *order* of the AR model. This equation can be rewritten to:

$$\begin{aligned} \Phi(B) \cdot \tilde{Y}_t &= a_t \\ \textit{with} \quad \Phi(B) &= 1 - \phi_1 B - \ldots - \phi_p B^p \end{aligned}$$

Also a *Moving Average (MA)* model considers the deviations from the mean. It expresses the current deviation as linear combination of the current and previous white noise terms and can be written as

$$\tilde{Y}_t = a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}$$

with q being the order of the MA model. This equation can again be rewritten into a more compact form:

$$\begin{aligned} \tilde{Y}_t &= \Theta(B) \cdot a_t \\ \textit{with} \quad \Theta(B) &= 1 - \theta_1 B - \ldots - \theta_q B^q \end{aligned}$$

An *Autoregressive Moving Average (ARMA)* model is simply the combination of an AR and an MA model:

$$\tilde{Y}_t = \overbrace{\phi_1 \tilde{Y}_{t-1} + \ldots + \phi_p \tilde{Y}_{t-p} +}^{Autoregressive\ Model} \underbrace{a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}}_{Moving\ Average\ Model}$$

In a more compact form, it is also possible to write

$$\Phi(B) \cdot \tilde{Y}_t = \Theta(B) \cdot a_t \tag{1}$$

with $\Phi(B)$ and $\Theta(B)$ as defined above. For clearer distinction, p is now called *autoregressive order* whereas q is called *moving average order*. An ARMA model is usually specified as ARMA(p,q). ARMA models obviously include all AR and all MA models (for either p or q being set to zero). Note that it is possible to compute $\tilde{Y}_t$ (and therefore Y_t) by rearranging equation (1):

$$\tilde{Y}_t = \Phi(B)^{-1} \cdot \Theta(B) \cdot a_t$$

An *Autoregressive Integrated Moving Average (ARIMA)* model does not consider deviations from the mean, but the actual time series values Y_t. Moreover, the assumption of a stationary time series varying around a fixed mean is dropped. The general idea is, that even for nonsationary time series, there might still be some difference of the time series which is stationary. Box & Jenkins [3] provide a more thorough argumentation of why this assumption is feasible.
An ARIMA model is defined by the following equation:

$$\Phi(B) \cdot \nabla^d \cdot Y_t = \Theta(B) \cdot a_t \tag{2}$$

with $\Phi(B)$ and $\Theta(B)$ as defined above and d being the order of the first stationary difference of the time series.
By defining

$$w_t := \nabla^d Y_t$$

it is possible to write equation (2) like an ordinary ARMA equation:

$$\Phi(B) \cdot w_t = \Theta(B) \cdot a_t$$

Depending on the application, it is also possible to set

$$w_t := \nabla^d (Y_t - \mu)$$

The term *Integrated* in the ARIMA acronym stems from the following thought: Once the w_t has been determined, Y_t can be calculated as

$$Y_t = \nabla^{-d} w_t$$

which roughly corresponds to an integration operation on w_t. An ARIMA model is usually specified as ARIMA(p,d,q). Obviously, all ARMA models are also ARIMA models (for $d = 0$).

In the original approach described above, differencing (i.e. repeatedly using the backward difference operator ∇) is used to make a time series stationary. Makridakis & Hibon [18] provide an alternative approach: they simply try to remove the linear trend component of the time series. Their results indicate that with a short lead time the differencing approach performs better, whereas with longer lead time the removal of the linear trend outperforms the differencing approach. However, as they note, for most real world applications, the trend cannot be safely assumed to be linear, so more complex methods might be needed to remove also nonlinear (e.g. exponential) trend components.

After having defined the ARIMA model, there is still the open question how one can fit an ARIMA model to data of a time series. The general guidelines provided by Box & Jenkins [3] are now known as the *"Box-Jenkins methodology"*. Khashei & Bijari [14] describe this methodology on a high level, consisting of three main phases:

– **Model Identification**: Make the time series stationary (by identifying d) and use certain autocorrelation properties to determine the model's orders p and q.
– **Parameter Estimation**: Estimate the model parameters ϕ_i and θ_j, given some error measure to minimize.
– **Diagnostic Checking**: Check if the model assumptions about the remaining prediction errors are satisfied.

Depending on the results of the diagnostic checking phase, these steps might be repeated multiple times until a satisfactory model is selected.
The ARIMA method presented so far is univariate, but there are also multivariate generalizations [8].

The ARIMA method was explicitly created to model and analyze time series. Applying it to time series prediction is therefore very straightforward: First, the ARIMA model is fit to the data of the time series, e.g. by using the Box-Jenkins methodology. Then, the current and previous values are used as input to the formulas. Finally, the formulas are rearranged to retrieve the mathematical solution $\hat{Y}_{t+1}$ to the equations derived from the training data.

4 Nonlinear Methods

4.1 General Idea

In contrast to linear methods, nonlinear methods allow for more complex than only linear relationships within the data. Nonlinear methods include most machine learning techniques like support vector machines, hidden Markov models and artificial neural networks (ANNs). Especially ANNs are applied widely due to their inherent flexibility.

Nonlinear methods work especially well with nonlinear data (and most real-life systems are nonlinear in nature or at least contain a significant nonlinear component), but the results of applying them to linear data has yielded mixed results. Therefore, they cannot be considered a "silver bullet" and should not be blindly applied to every forecasting problem [14]. The following subsections will take a closer look at the Multi Layer Perceptron (MLP), which is the most commonly used ANN type, and at approaches based on pattern matching.

4.2 Multi Layer Perceptron

ANNs in general are neurobiologically motivated machine learning techniques which try to mimic the network of neurons found in human brains. This introduction into MLPs is necessarily brief. It is mainly based on Mitchell's work [21], which provides a very detailed description of the MLP approach.

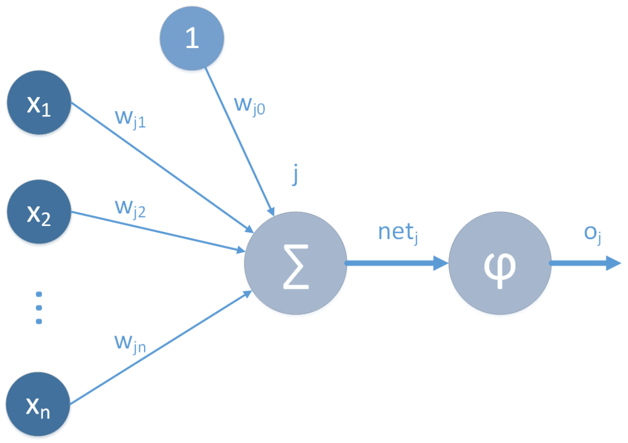

Fig. 4.2.1. Illustration of a simple perceptron with index j.

The basic unit of any ANN is an artificial neuron, which in Multi Layer Perceptrons is called a *perceptron*. Figure 4.2.1 shows such a perceptron with index j. It takes a vector of inputs $(x_1, \ldots, x_n)$ where each input x_i has an associated weight w_{ji}. The perceptron then takes the weighted sum of the inputs $\sum_{i=1}^{n} w_{ji}x_i$ and adds so-called bias term w_{j0}. In order to write this sum nicely, often a so-called "bias unit" $x_0 = 1$ is introduced, so one can write

$$net_j = \sum_{i=0}^{n} w_{ji}x_i$$

To determine the output of a perceptron, a so-called *transfer function* $\varphi(net_j)$ is used. Popular choices are the linear transfer function

$$id(x) = x$$

and the sigmoid function

$$sig(x) = \frac{1}{1+e^{-x}}$$

The output of a perceptron can be expressed like this:

$$o_j = \varphi(net_j) = \varphi\left(\sum_{i=0}^{n} w_{ji}x_i\right)$$

Depending on the transfer function φ, the output of a perceptron may be linear (e.g. when using the linear transfer function $id(x) = x$) or nonlinear (e.g. when using the sigmoid function).

In order to train a perceptron (i.e. determining the weights w_{ij}), an error function is needed. It quantifies the output error and will be minimized during training. The most commonly used error function in this context is defined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

where $\vec{w}$ is the weight vector of the perceptron, D is the training set and t_d and o_d are the desired and the real output for a training example d.
Minimizing this error function is done by *gradient descent*, i.e. by using the gradient

$$\nabla E(\vec{w}) = \left[\frac{\delta E}{\delta w_{j0}}, \ldots, \frac{\delta E}{\delta w_{jn}}\right]$$

The weight update Δw_{ji} for each weight is determined by

$$\Delta w_{ji} = -\eta\frac{\delta E}{\delta w_{ji}}$$

η is the so-called *learning rate* and determines the step size for the weight update, a negative sign is used to move the weights into the direction that decreases E.

$\frac{\delta E}{\delta w_{ji}}$ can be calculated by simply differentiating $E(\vec{w})$, which for a linear transfer function results in

$$\frac{\delta E}{\delta w_{ji}} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

where x_{id} is the value of input x_i for training example d.
Note that in order to compute the gradient, the transfer function must be differentiable. This requirement is fulfilled for both the linear transfer function and the sigmoid function.
The weights will updated according to the following rule:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

These weight updates will be computed in an iterative manner until a minimum of the error function is reached.

Gradient descent basically minimizes the error function by following its gradient (i.e. its "derivation") into a minimum. It can be shown that for a single perceptron, the error function always has only one minimum, hence gradient descent is guaranteed to always find a global minimum (as there are no local minima). Note that the learning rate η has a significant influence on the speed of convergence. In some cases, when η is chosen too large, gradient descent might not converge and end up oscillating around the minimum of E.

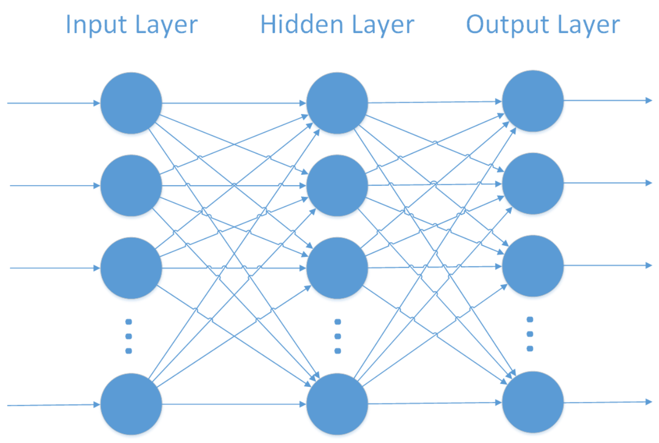

Fig. 4.2.2. An exemplary three-layer MLP with one single layer.

As a single perceptron is not very powerful – it can only model linearly separable classes, which e.g. is not sufficient for solving the XOR problem. Therefore, in practice a network of such perceptrons is used, with the perceptrons being devided into multiple layers – hence the name "Multi Layer Perceptron". See Figure 4.2.2 for an illustration.

There are three types of layers:

- **Input Layer**: First layer in the network, represents the input vector; consists of the input vector values, not of perceptrons.
- **Hidden Layer**: Layer somewhere between the first and the last layer of the network.
- **Output Layer**: Last layer in the network, the output of this layer is the overall output of the network.

Each perceptron in layer k will receive the output of each perceptron in layer $k-1$ as input and its output will be forwarded to all perceptrons in layer $k+1$. That is, the x_i of the perceptrons are actually the o_i of the perceptrons in the preceding layer. There are no other links in the network than the ones described which implies that the network is acyclic. This is the reason why networks of this type are called *feed-forward networks*.

According to Khashei & Bijari [14], MLPs are the most commonly used neural networks in time series prediction. For most applications in this area, only one hidden layer is used and the output layer contains only one perceptron. For such three-layer MLPs with a single output perceptron j one can write its output as

$$o_{MLP} = \varphi_{out}\left(\sum_{i=0}^{Q} w_{ij} \cdot \varphi_{hidden}\left(\sum_{h=0}^{P} w_{hi} x_i\right)\right)$$

with P being the number of inputs and Q being the number of perceptrons in the hidden layer. In most cases, the sigmoid function is used in the hidden layer (therefore $\varphi_{hidden} = sig$), whereas for the output layer, a linear transfer function is used (therefore $\varphi_{out} = id$). It can be easily seen that under these circumstances, due to the use of the sigmoid function, the output of the MLP is nonlinear.

Due to its increased complexity compared to a single perceptron, training an MLP is more complicated and computationally more expensive than training a single perceptron: because of the larger number of weights the hypothesis space (i.e. the space of possible weight combinations) is much larger.
The algorithm most commonly used for training MLPs is called *backpropagation* and can be thought of as a variant of gradient descent. The basic idea of backpropagation is that the error measured at the output layer is propagated backwards through all previous layers in order to update the network's weights (i.e. the weights of all perceptrons in the MLP). In the following only the big picture of backpropagation will be given.
Let δ_j be the error term associated with perceptron j. This error term will be used to compute the weight update:

$$\Delta w_{ji} = \eta \cdot \delta_j \cdot x_i$$

For the output layer neurons, the error term is calculated as

$$\delta_j = \varphi'(o_j) \cdot (t_j - o_j)$$

where φ' denotes the first derivation of φ. In this case, the weight updates are basically computed in the same way as for a single perceptron.
However, for hidden layer neurons, the error term is determined as

$$\delta_j = \varphi'(o_j) \cdot \sum_{k \in layer_{m+1}} w_{jk}\delta_k$$

where m denotes the layer of perceptron j. Hence, the sum is aggregated over all perceptrons in the subsequent MLP layer, i.e. all perceptrons that receive o_j as an input. This means, that the update depends on the weighted sum of all error terms of subsequent perceptrons. This can be intuitively understood as the error being propagated backward through the network of perceptrons. Mitchell [21] explains in greater detail the mathematical foundations of this algorithm and why it is guaranteed to converge.

As the backpropagation algorithm seems to be biologically not plausible, Aitkenhead et al. [1] propose an alternative way of training neural networks. Their "Local Interaction Method" refrains from propagating the global error from the output layer back through the complete network. Instead, the weight updates are determined only based on local criteria, i.e. only depending on a perceptron's immediate neighborhood. They show in two small experiments, that their approach can perform better than the traditional backpropagation approach. However, backpropagation is still the most widely applied training algorithm for MLPs and can be thought of as an accepted standard.

MLPs are a very flexible machine learning technique because they do not make any assumptions about the function to be approximated. However, this adaptability does not come without any cost:
First of all, the overall network structure (the number of layers to use and the number of perceptrons in each of these layers) usually must be determined before training the network via backpropagation. Although there exist some approaches to automatically determine these parameters (e.g. the use of genetic algorithms as in the work of Flores et al. [10]), in practice this is mostly done by expert judgement or trial and error.
Moreover, it is often necessary to train the network for a long time until good performance is reached. However, if a network is trained too long or if it contains too many perceptrons, it will start to *overfit* the data. This means, that there will be a very low error rate on the training data, but large errors for previously unseen data, i.e. the network does not generalize. Overfitting is a well known issue in machine learning and there exist techniques to deal with it (Mitchell [21] suggests e.g. the use of validation sets in addition to training and test sets).
Another problem is the existence of local minima: because backpropagation can be viewed as variant of gradient descent, it will find a minimum, but the error surface of an MLP usually contains multiple local minima. Both the learning rate η and the initial values of the weights can influence whether the algorithm will find a local or a global minimum. There are also some advanced techniques to prevent backpropagation to get stuck in local minima in order to find a global

minimum (e.g. adding a momentum term to the weight update). In most real-world applications, however, local minima are not a big concern.

ANNs in general and therefore also MLPs were not explicitly created to model, analyze or predict time series. In fact, the original intentions of use were classification and function approximation.
However, time series prediction can be seen as a special case of the more general function approximation problem. Therefore, the MLP can be trained to approximate the function

$$f(Y_t, \cdots, Y_{t-l}) := Y_{t+1}$$

This can be done by creating a training set from the time series data:

$$\begin{aligned}(x_1, \ldots, x_n) &:= (Y_t, \ldots, Y_{t-l}) \quad \textit{and} \\ t_{net} &:= Y_{t+1}\end{aligned}$$

Here, t_{net} denotes the desired output of the network.
Moreover, since ANNs do not make any assumptions about the input data or about the function to approximate, one can simply add additional variables as input to make the prediction multivariate.

There are, of course, also other types of ANNs that can be used for time series prediction. So-called *Time Delayed Neural Networks (TDNNs)* are capable of recognizing input patterns independent of their position in time and were originally used for phoneme recognition [30]. They are interesting in the context of time series analysis because of their ability to robustly handle inputs shifted in time. This property is called *time invariance.*
The class of *Recurrent Neural Networks* (the counterpart of feed forward networks) describes neural networks that have feedback connections, e.g. the output of a neuron being fed back as one of its inputs for the next time step. Due to this property, RNNs are capable of maintaining some inner state through time. This can be very valuable when dealing with time-dependent data like time series.
Kim and Shin [15] present the application of both TDNNs and RNNs to time series prediction. Although these types of ANNs might be better suited for dealing with time series, the MLP can still be considered to be the "standard ANN" in time series prediction. Many papers in this area even use the general term ANN to refer to an MLP.

4.3 Pattern-Matching approaches

A group of nonlinear approaches different from ANNs is based on pattern matching on sequences of labels. Figure 4.3.1 illustrates the general architecture of these approaches: In a first step, the input is transformed from a sequence of real values $Y_t, \ldots, Y_{t-l}$ into a sequence of labels $L_t, \ldots, L_{t-l}$. Then, this sequence of labels is matched to the knowledge base (consisting of other label-sequences

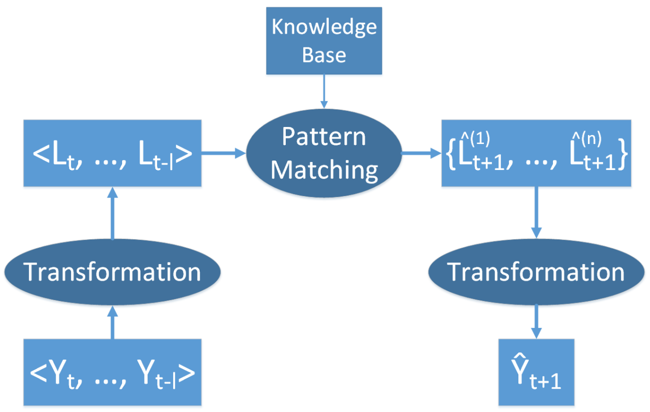

Fig. 4.3.1. Overview over the general architecture of pattern-matching approaches.

or consisting of a set of rules) in order to map it to one or more output labels $\hat{L}_{t+1}^{(1)}, \ldots, \hat{L}_{t+1}^{(n)}$. These output labels are then in a third step converted back into a real-valued prediction $\hat{Y}_{t+1}$.

Martinez et al. [19] use an approach called “Pattern Sequence-based Forecasting” on an energy consumption data set. This data set contains the energy consumption per hour. In a first step, they divide their data set into days and cluster these days based on their energy consumption curve using the k-means algorithm. For each day, the according cluster ID is stored as label.
When predicting the energy consumption curve for the next 24 hours, the sequence of the last l labels is taken into account. The history of label sequences is searched for the current sequence and for each match, the day immediately following this match is recorded. The prediction is then generated by averaging over the energy consumption rate of these days. If the sequence of the last l labels has never occurred in the history, the sequence of the last $l-1$ days is used and so on, until a match is found. Note that only exact matches are taken into account.

Also approaches using fuzzy sets fit into the category of pattern-matching approaches. Song & Chissom [25, 26] propose so-called *fuzzy time series* in this context. The training data input (consisting of real numbers) is first mapped to fuzzy sets, then fuzzy relationships between sequences of fuzzy sets are learned. These relationships have the form of "IF ... THEN ..." rules which are much easier to interpret by a human than e.g. the weights of an MLP.
In the forecasting step, the input sequence is mapped to a sequence of fuzzy set memberships. The learned fuzzy rules are applied and the result (also expressed with fuzzy set memberships) is “defuzzified” into a real-valued prediction. This “defuzzification” can be done e.g. by computing a weighted sum of the set means

with the set memberships being used as weights. Due to the inherent fuzziness of this whole process, there is no exact matching required. Hence, this approach seems to be more flexible than the one of Martinez et al.
Li et al. [16] apply the idea of fuzzy time series to the prediction of electricity prices. However, in general this approach is not yet used widely in practice.

5 Combination of Methods

5.1 General Idea

Virtually every individual forecasting method has its advantages, but also its limitations. Although the combination of different methods does not necessarily lead to better performance, it reduces the risk of using a completely inappropriate method and therefore the risk of total failure [12, 14].

Khashei & Bijari [14] distinguish hybrid architectures in two dimensions: They are either *homogeneous* (e.g. a set of differently configured MLPs) or *heterogeneous* (e.g. a combination of an ARIMA and a MLP). Moreover, one can distinguish *competitive* and *cooperative* architectures: in a competitive architecture, the "best" method will alone determine the output. In a cooperative architecture, the overall prediction is computed by combining the individual methods' forecasts. Mostly, the different methods model different aspects of the time series and are used sequentially.

In the following, two approaches towards hybrid architectures are presented in more detail.

5.2 Cooperative Combination of ARIMA and MLP

As described in Sections 3 and 4, ARIMA is a linear method, whereas MLP is a nonlinear method. Although they are quite different in nature, Zhang [33] also notes some similarities, e.g. both methods include a rich variety of models, need a large training set and are prone to overfitting.
Both Zhang [33] and Khashei & Bijari [14] use a cooperative approach of combining the ARIMA and MLP methods by dividing the time series Y_t into a linear part L_t (modeled by the ARIMA) and a nonlinear part N_t (modeled by the MLP). Both only consider univariate one-step ahead forecasts.

Zhang [33] assumes that the linear and the nonlinear components are additive, therefore

$$Y_t = L_t + N_t$$

His hybrid architecture (see Figure 5.2.1) is constructed as follows:
First, the ARIMA is used to fit the linear component L_t yielding predictions

$$\hat{L}_{t+1} = Y_{t+1} + e_{t+1}$$

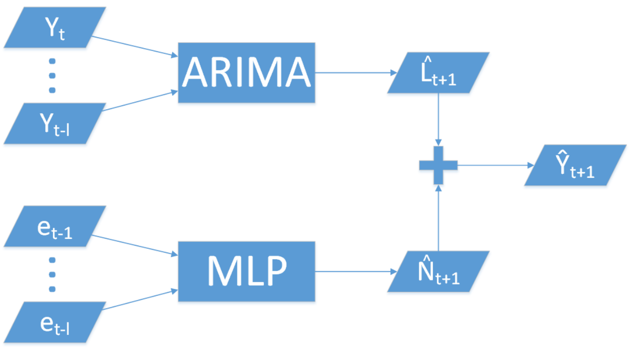

Fig. 5.2.1. Visualization of Zhang's [33] hybrid model for time series prediction which uses an additive combination of ARIMA and MLP predictions.

of the linear component. The residuals e_{t+1} only contain nonlinear relationships, which are modeled using an MLP with a single hidden layer, such that

$$e_{t+1} = f(e_{t-1}, ..., e_{t-l}) + \epsilon_{t+1}$$

with f being the function realized by the MLP and ϵ_{t+1} being the remaining error term. The forecast of the MLP is called $\hat{N}_{t+1}$. The overall forecast of the model is determined by adding the two components:

$$\hat{Y}_{t+1} = \hat{L}_{t+1} + \hat{N}_{t+1}$$

As Zhang showed by carrying out an experiment with three different data sets, this hybrid approach was able to outperform both individual ARIMA and ANN models.

Khashei & Bijari [14], however, argue that one can not safely assume the additive decomposition of a time series into a linear and a nonlinear part. Their approach is depicted in Figure 5.2.2. They describe the time series as general function of a linear and a nonlinear component:

$$Y_t = f(L_t, N_t)$$

Again, an ARIMA is used to fit the linear component. The nonlinear component is split into two subcomponents $N_t^{(1)}$ (based on the residuals) and $N_t^{(2)}$ (based on the w_j used in the ARIMA model) which are both approximated by an MLP:

$$\hat{N}_{t+1}^{(1)} = f^{(1)}(e_t, \ldots, e_{t-n}) \quad \textit{and}$$
$$\hat{N}_{t+1}^{(2)} = f^{(2)}(w_t, \ldots, w_{t-m})$$

with $f^{(1)}$, $f^{(2)}$ being the nonlinear functions represented by the respective MLP. The overall prediction can then be represented as

$$\hat{Y}_{t+1} = f(\hat{N}_{t+1}^{(1)}, \hat{L}_{t+1}, \hat{N}_{t+1}^{(2)})$$

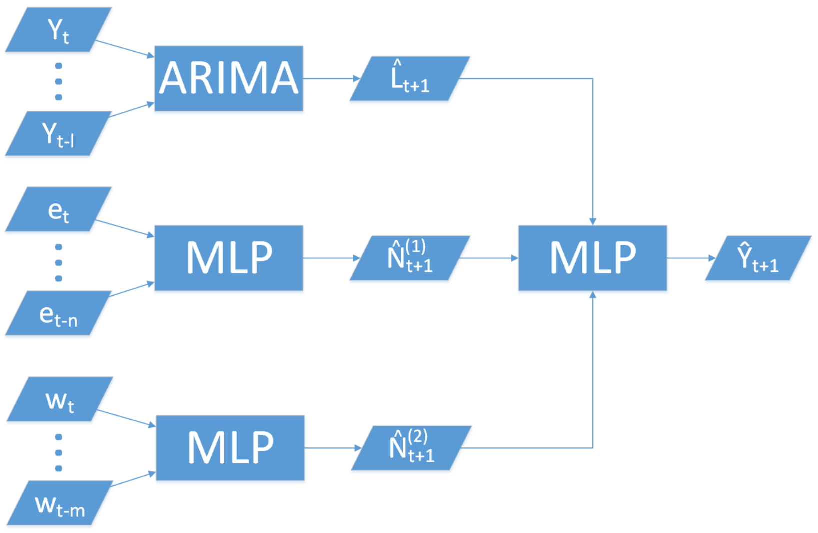

Fig. 5.2.2. Visualization of the hybrid model by Khashei & Bijari [14] which uses multiple MLPs in combination with an ARIMA model for time series prediction.

with f being the overall function represented by the resulting MLP.
Although three different MLPs are used, they can easily be combined into a single MLP receiving inputs $e_t, \ldots, e_{t-n}, w_t, \ldots, w_{t-m}, \hat{L}_{t+1}$.
When comparing the performance of their hybrid method to individual ANN and ARIMA models and to the hybrid method of Zhang, they arrived at the following results: The individual ANN tended to be slightly better than the individual ARIMA. Moreover, both hybrid methods outperformed the individual methods. These two results support the ones of Zhang mentioned aboved. Khashei & Bijari also found that their model yielded higher performance than Zhang's model. They conclude that their approach of combining linear and nonlinear forecast with an arbitrary function is more appropriate than Zhang's approach of simply adding them up.

5.3 Ensemble-based approaches

Another approach of combining different methods is based on so-called *ensembles*. An ensemble is a group of individual forecasting methods that use the same functional approach but different models (i.e. different parameter sets). The key idea is that the methods within an ensemble should be diverse in order to get the most performance gain. This model diversification can be reached by training the models differently.
Ruta et al. [23] suggest the use of different subsets of the training data, different features, different noise levels and a different initialization of the model parameters. They also list different approaches to combine the individual forecasts,

e.g. weighted average with weights determined by the prior accuracy of the individual forecasting models. For their practical evaluation, they use a two-stage approach with k groups: within each group, a single best predictor is selected ("winner takes it all") and then in a second step, the forecasts of the groups are combined by computing a simple average.
The work of Ruta et al. does not only cover ensembles, but proposes a complete architecture for time series prediction which cannot be treated in more detail here.

The approach of Martínez-Rego et al. [20] also makes use of ensembles, although in a somewhat different setting. For their "Distributed Commitees of Local Experts", they use clustering algorithms to create clusters in the input space. Each cluster is represented by one representative data point, called a *cluster node*. Then, for each cluster, a MLP is trained on the data belonging to this cluster and on the data belonging to neighboring clusters. After this, ensembles (called "committees" in their paper) are created: For each cluster, the corresponding ensemble contains all predictors that have been trained on the data points belonging to this cluster. That means, the ensemble of a cluster contains the MLP of that cluster itself and the MLPs of the neighboring clusters.
In the prediction step, the input is mapped to a cluster by picking the cluster node being closest to the input in the input space. The ensemble of this cluster will then determine the overall prediction. Martínez-Rego et al. use another MLP for the combination of the predictions made by the predictors within the ensemble. They call this a *trainable fusion-rule* in contrast to *fixed fusion-rules* like the average operation used by Ruta et al.

The approach taken by Silipo and Winters [24] can also be seen as a ensemble-based approach in the wider sense as it is also based on cluster-wise prediction. They predict the total electricity usage of about 6000 private and commercial consumers by first clustering them, then fitting a model for each of the clusters, and finally, calculating the overall prediction by summing up the individual predictions for all clusters.

6 Predictive Intervals

6.1 General Idea

Every prediction about the future necessarily contains come uncertainty. This uncertainty can have different sources [13]:

- **Input Uncertainty**: Errors in measuring and sampling the data.
- **Parametric Uncertainty**: Inability to identify the best parameter set when fitting the model.
- **Model Structure Uncertainty**: Information loss due to the conversion of real world problems into abstract mathematical formulas.

Of course, also the random component R of the time series (mentioned in Section 2) contributes to the overall uncertainty.

For multi-step ahead predictions the uncertainty is in general larger than for one-step ahead predictions: the larger the lead time, the greater the uncertainty [5]. Therefore, the forecasting accuracy will decrease with increasing lead time [31]. There are different reasons for this effect:
Li et al. [17] point out that historical data used for training the forecasting models are often incomplete, noisy and ambiguous and do not necessarily contain the necessary patterns for long-term forecasts.
As Sovilj et al. [29] add, long term prediction requires a long history of observations for training purposes. This results in the widely known "curse of dimensionality", which states that the amount of training data needed grows exponentially with the number of input dimensions being analyzed. In this context, the number of input dimensions corresponds to the length of the observation history.
Moreover, for multivariate time series, the prediction $\hat{Y}_{t+k}$ might depend on some $X_{t+j}^{(i)}$ $(1 \leq j \leq k)$. This introduces the additional uncertainty attached to forecasting $\hat{X}_{t+j}^{(i)}$ [23].
Another reason for this increased uncertainty is based on the approach used for making multi-step ahead predictions. In general, one can distinguish two approaches to multi-step ahead prediction [17, 23, 27]:
On the one hand, there is the *recursive* approach: the one-step ahead prediction is recursively applied with predictions $\hat{Y}_{t+1}, ...\hat{Y}_{t+k-1}$ being fed back into the prediction method:

$$\begin{aligned}
\hat{Y}_{t+1} &= f(Y_t, Y_{t-1}, \ldots, Y_{t-l}) \\
\hat{Y}_{t+2} &= f(\hat{Y}_{t+1}, Y_t, \ldots, Y_{t-l+1}) \\
&\vdots
\end{aligned}$$

This approach is very straightforward to implement, but has one major disadvantage: feeding back the predictions as inputs leads to an accumulation of prediction errors and therefore decreases the prediction accuracy [27, 28].
On the other hand, there is the *direct* approach, where k different models are trained for a k-step ahead prediction:

$$\begin{aligned}
\hat{Y}_{t+1} &= f_1(Y_t, Y_{t-1}, \ldots, Y_{t-l}) \\
\hat{Y}_{t+2} &= f_2(Y_t, Y_{t-1}, \ldots, Y_{t-l}) \\
&\vdots
\end{aligned}$$

This approach is computationally more expensive (because multiple models need to be fit), but does not suffer from accumulated errors like the recursive approach. However, there is still some problem with this approach: the autocorrelation of the time series for large lag values will probably be rather low, which means that historical data is not suitable for explaining data points far in the future.

By combining both approaches into the *DirRec* approach, Sorjamaa & Lendasse [28] try to overcome the weaknesses of the single approaches.

Most forecasting methods do not explicitly take into account the uncertainty attached to their forecast. They usually output exactly one value: their forecast for the future value of the time series. However, this gives no information about the confidence with which this forecast is made. To better assess this prediction confidence, so-called *predictive intervals* (or *prediction intervals*) can be used.
A predictive interval is an interval that will contain the true value of Y_{t+k} with a specified probability p (most commonly $p = 0.95$). Therefore, the output of a forecasting method using predictive intervals is no longer a single value $\hat{Y}_{t+k}$, but rather an interval $\Gamma_{t+k} = [LOW_{t+k}, HIGH_{t+k}]$. De Gooijer & Hyndman [8] argue that the term *prediction interval* should be used for predictions, whereas the term *confidence interval* should be reserved for model parameters to avoid confusion.

For evaluating predictive intervals, two measures are usually used, which are both computed on the test data set [13]: the probability of coverage POC which represents the percentage of test data points lying in the predictive interval, and the average width AW of the predictive interval. The general goal is to maximize POC while minimizing AW. As Dashevskiy & Luo [6] point out, a narrow AW indicates an efficient prediction interval, whereas a large POC close to the desired value of p indicates that the prediction interval is valid.

If some sort of Bayesian method is used, calculating a predictive interval is relatively straightforward as probabilities and distributions are already used throughout the method. The more interesting questions is how to determine predictive intervals for other methods which usually would only make a single-point forecast.
Chatfield [4] provides a thorough introduction into the mathematical problem of constructing prediction intervals and gives an overview of different approaches and common pitfalls. Here, we will focus on two simple techniques to construct approximations of predictive intervals: ensembles and conformal predictors. Although they might be mathematically inappropriate, they are are fairly easy to implement and in practice yield satisfactory results.

6.2 Ensembles

The general idea of ensembles has already been introduced in Section 5. Ensembles not only can be used for achieving a more robust prediction accuracy, but they also can be used to estimate predictive intervals. This can be achieved by calculating mean and variance of the predictions made by the ensemble members.
Kasiviswanathan et al. [13] used this approach to approximate predictive intervals for rainfall runoff data: in their experiment, they trained an MLP on the available data, and then created an ensemble by making copies of the trained

MLP. For each of these copies, they perturbed some randomly selected weights to diversify the ensemble.
The overall prediction of the ensemble was computed as simple arithmetic mean and the variance of the predictions made by the individual MLPs was used to compute the width of the predictive interval. For multi-step ahead forecasts, the direct approach was used. Figure 6.2.1 shows an illustration of their results, where the red bands indicate the predictive interval and the black dots denote the observed values. It nicely illustrates that for larger lead times the uncertainty tends to grow larger. This can be seen by the tendency that the predictive intervals grow larger and contain less of the observed data points as the lead time increases.

6.3 Conformal Predictors

Conformal predictors are a general approach of approximating a confidence interval. They can be used on top of virtually any classification or prediction algorithm (called the *underlying algorithm*). Dashevskiy & Luo [6, 7] provide an introduction into conformal predictors in general and their application to time series prediction. This section is largely based on their work.

Conformal predictors are a machine learning technique that makes predictions based on how well a new example will fit into the set of known training examples. A so-called *nonconformity measure* α is used to represent the dissimilarity between examples. Each example z_i consists of an object x_i and a label y_i, therefore

$$z_i = (x_i, y_i)$$

The *p-value* for a new example $z_n = (x_n, y)$ is defined as:

$$p(y) = \frac{\#\{i = 1, \ldots, n : \alpha_i(y) \geq \alpha_n(y)\}}{n} \tag{3}$$

where $\alpha_i(y)$ denotes the nonconformity score of object x_i if the value y is assigned to the new object x_n.
A large value of $p(y)$ indicates that the new example x_n with label y is very typical for the training set. This is because equation (3) computes the fraction of elements in the training set that have a greater dissimilarity to the training set than the new example. If this fraction is large, obviously the new example fits in very well. Therefore, the general goal is to find the label y with the largest p-value. Assuming that the order of the examples is irrelevant, it is even possible to prove that for any given error probability ϵ one can construct a prediction interval that will contain the actual observation with a probability of $1 - \epsilon$:

$$\mathit{\Gamma} = \{y : p(y) \geq \epsilon\} \tag{4}$$

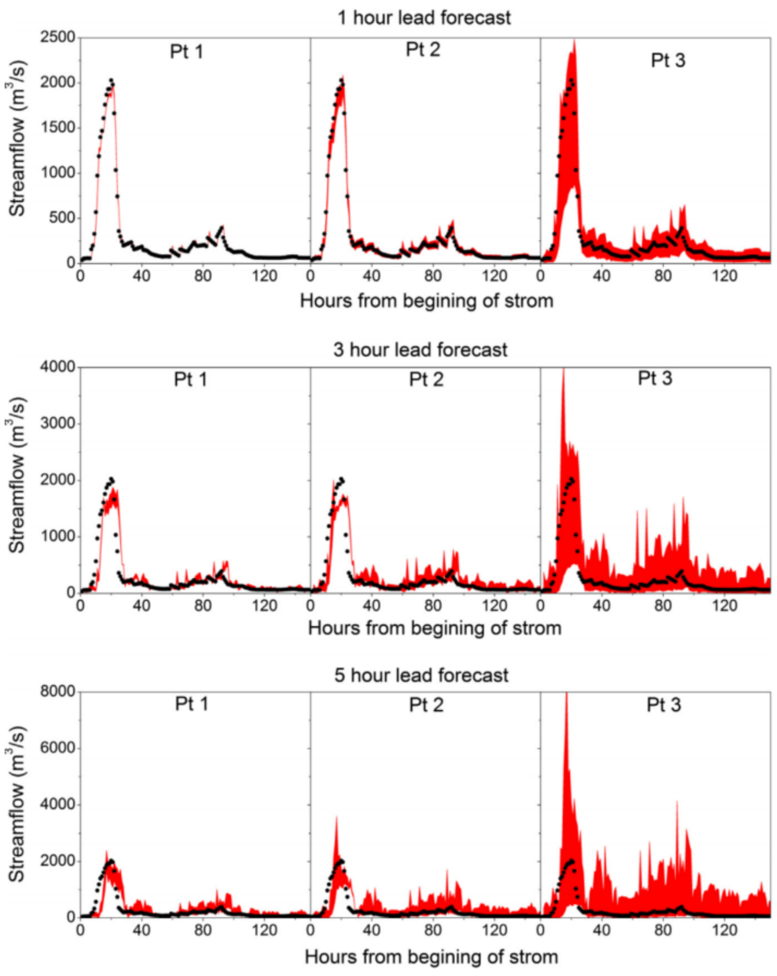

Fig. 6.2.1. Diagram by Kasiviswanathan et al. [13], showing the resulting predictive intervals for different examples and different lead times.

Due to the use of labels, conformal predictors are rather a classifier approach than a regression approach. However, it is possible to define the z_i in a way that conformal predictors can be also used for time series prediction:

$$z_i = (x_i, y_i) := ((Y_{t-l}, \dots, Y_t), Y_{t+1})$$

Let $\hat{Y}_{t+1}$ be the prediction of the underlying algorithm. Let the error measure for the new object be defined as

$$\alpha_n = |\hat{Y}_{t+1} - Y_{t+1}|$$

Then, for an error probability ϵ, construct the predictive interval as

$$\Gamma_{t+1} = \left[\hat{Y}_{t+1} - r_{max}, \hat{Y}_{t+1} + r_{max}\right] \tag{5}$$

with some r_{max}. The true value Y_{t+1} lies in this interval if and only if:

$$\alpha_n = |\hat{Y}_{t+1} - Y_{t+1}| \leq r_{max} \tag{6}$$

Therefore, r_{max} gives an upper bound on α_n. For given error probability ϵ, equation (4) shall hold true. With equations (5) and (6), r_{max} can then be calculated like this:

$$r_{max} = max_{r \in \mathbb{R}} \left\{ \frac{\#\{i = 1, \dots, n : \alpha_i(y) \geq r\}}{n} \geq \epsilon \right\}$$

By computing r_{max} in this way, the predictive interval Γ_{t+1} can be calculated based on the prediction $\hat{Y}_{t+1}$ of the underlying algorithm.
The assumption of the order of examples being irrelevant clearly does not hold for time series data, therefore no provable guarantees about the predictive interval can be made anymore. Despite this theoretical lack of valididy, Dashevskiy & Luo report good practical results for applying conformal predictors to time series.

Papadopoulos & Haralambous [22] propose a variant of conformal prediction they call "Inductive Conformal Prediction". It yields improved performance when used with ANNs as underlying algorithm. Figure 6.3.1 shows their results for a sunspot activity data set.

7 Conclusion

This paper gave an introduction into the topic of time series prediction which is a relevant problem in many application areas.

Due to space limitations, this paper can only serve as a brief overview over the topic of time series prediction. With ARIMA and MLP, the two most popular methods were introduced, but there are many other approaches that could not be discussed in this paper, e.g. support vector machines (SVMs) or classification and regression trees (CARTs). De Gooijer & Hyndman [8] provide a very thorough overview over a variety of approaches.

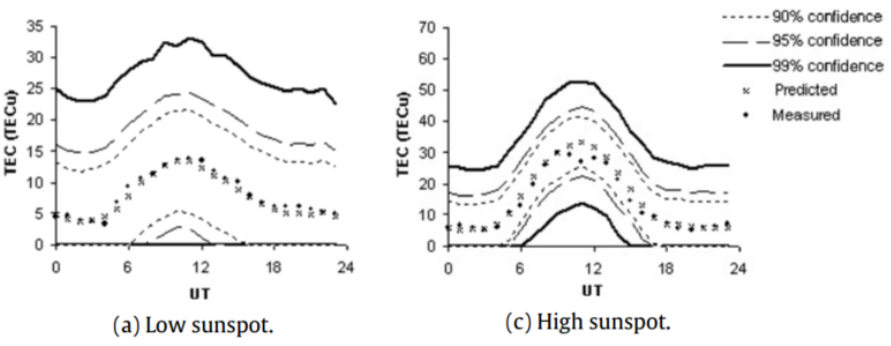

Fig. 6.3.1. Results from the paper of Papadopoulos & Haralambous [22], showing the predictive intervals for two different sunspot activity time series.

In recent years, the use of MLPs has become more and more popular in the field of time series prediction. But although there are other neural networks like TDNNs and RNNs that might be better suited to deal with time series data, they are not yet widely applied in practice.

All predictive methods are based on the assumption that the future is somehow correlated to the past. While this may be true most of the time, there are unforeseeable events like natural disasters, which in general cannot be predicted and therefore cannot be taken into account. It is not quite clear how this limitation can be overcome. Change point analysis [9] tries to detect points in the time series when the underlying process changes (i.e. another model needs to be used) and might be one possible approach to overcome this limitation.

Although most prediction methods work quite well in practice, there is currently no approach that performs equally well for all kinds of input data. As every approach has its individual strengths and weaknesses, the combination of different approaches and the use of predictive intervals instead of single-point forecasts seem to be reasonable steps towards better forecasts. There is still a lot of work to be done until the problem of time series prediction can be considered to be solved.

The original intention of this paper was to analyze the application of time series prediction to activity recognition on mobile devices. However, it turned out that there seems to be no research on the question how time series prediction can be used in this context. Therefore, this seems to be a worthwhile research direction for future work.

References

1. Aitkenhead, M., McDonald, A., Dawson, J., Couper, G., Smart, R., Billett, M., Hope, D., Palmer, S.: A novel method for training neural networks for time-series prediction in environmental systems. Ecological Modelling 162(1), 87–95 (2003)
2. Amjady, N.: Day-ahead price forecasting of electricity markets by a new fuzzy neural network. Power Systems, IEEE Transactions on 21(2), 887–896 (2006)
3. Box, G., Jenkins, G.M., Reinsel, G.: Time series analysis: Forecasting & control (1994)
4. Chatfield, C.: Calculating interval forecasts. Journal of Business & Economic Statistics 11(2), 121–135 (1993)
5. Chatfield, C.: Time-series forecasting. Significance 2(3), 131–133 (2005)
6. Dashevskiy, M., Luo, Z.: Time series prediction with performance guarantee. Communications, IET 5(8), 1044–1051 (2011)
7. Dashevskiy, M., Luo, Z.: Network traffic demand prediction with confidence. In: GLOBECOM. pp. 1453–1457 (2008)
8. De Gooijer, J.G., Hyndman, R.J.: 25 years of time series forecasting. International journal of forecasting 22(3), 443–473 (2006)
9. Denison, D.G., Mallick, B.K., Smith, A.F.: A bayesian cart algorithm. Biometrika 85(2), 363–377 (1998)
10. Flores, J.J., Graff, M., Rodriguez, H.: Evolutive design of arma and ann models for time series forecasting. Renewable Energy 44, 225–230 (2012)
11. Hawkins, J., Blakeslee, S.: On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines. Henry Holt & Company, New York, NY (2004)
12. Hibon, M., Evgeniou, T.: To combine or not to combine: selecting among forecasts and their combinations. International Journal of Forecasting 21(1), 15–24 (2005)
13. Kasiviswanathan, K., Cibin, R., Sudheer, K., Chaubey, I.: Constructing prediction interval for artificial neural network rainfall runoff models based on ensemble simulations. Journal of Hydrology 499, 275–288 (2013)
14. Khashei, M., Bijari, M.: A novel hybridization of artificial neural networks and arima models for time series forecasting. Applied Soft Computing 11(2), 2664–2675 (2011)
15. Kim, H.j., Shin, K.s.: A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. Applied Soft Computing 7(2), 569–576 (2007)
16. Li, G., Liu, C.C., Mattson, C., Lawarree, J.: Day-ahead electricity price forecasting in a grid environment. Power Systems, IEEE Transactions on 22(1), 266–274 (2007)
17. Li, S.T., Kuo, S.C., Cheng, Y.C., Chen, C.C.: Deterministic vector long-term forecasting for fuzzy time series. Fuzzy Sets and Systems 161(13), 1852–1870 (2010)
18. Makridakis, S., Hibon, M.: Arma models and the box–jenkins methodology. Journal of Forecasting 16(3), 147–163 (1997)
19. Martinez Alvarez, F., Troncoso, A., Riquelme, J.C., Aguilar Ruiz, J.S.: Energy time series forecasting based on pattern sequence similarity. Knowledge and Data Engineering, IEEE Transactions on 23(8), 1230–1243 (2011)
20. Martínez-Rego, D., Fontenla-Romero, O., Alonso-Betanzos, A.: Efficiency of local models ensembles for time series prediction. Expert Systems with Applications 38(6), 6884–6894 (2011)
21. Mitchell, T.M.: Machine learning. 1997. Burr Ridge, IL: McGraw Hill 45 (1997)

22. Papadopoulos, H., Haralambous, H.: Reliable prediction intervals with regression neural networks. Neural Networks 24(8), 842–851 (2011)
23. Ruta, D., Gabrys, B., Lemke, C.: A generic multilevel architecture for time series prediction. Knowledge and Data Engineering, IEEE Transactions on 23(3), 350–359 (2011)
24. Silipo, R., Winters, P.: Big data, smart energy, and predictive analysis (2013)
25. Song, Q., Chissom, B.S.: Forecasting enrollments with fuzzy time series—part i. Fuzzy sets and systems 54(1), 1–9 (1993)
26. Song, Q., Chissom, B.S.: Fuzzy time series and its models. Fuzzy sets and systems 54(3), 269–277 (1993)
27. Sorjamaa, A., Hao, J., Reyhani, N., Ji, Y., Lendasse, A.: Methodology for long-term prediction of time series. Neurocomputing 70(16), 2861–2869 (2007)
28. Sorjamaa, A., Lendasse, A.: Time series prediction using dirrec strategy (2006)
29. Sovilj, D., Sorjamaa, A., Yu, Q., Miche, Y., Séverin, E.: Opelm and opknn in long-term prediction of time series using projected input data. Neurocomputing 73(10), 1976–1986 (2010)
30. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.J.: Phoneme recognition using time-delay neural networks. Acoustics, Speech and Signal Processing, IEEE Transactions on 37(3), 328–339 (1989)
31. Wong, F.S.: Time series forecasting using backpropagation neural networks. Neurocomputing 2(4), 147–159 (1991)
32. Yadav, V., Toshniwal, D.: Graph based framework for time series prediction. Trends in Information Management 7(2) (2011)
33. Zhang, G.P.: Time series forecasting using a hybrid arima and neural network model. Neurocomputing 50, 159–175 (2003)