Technical University of Košice

Faculty of Mining, Ecology, Process Control and Geotechnologies

Institute of Control and Informatization of Production Processes

Process Theory

Ing. Ales Jandera

10. May 2024

Contents

1 Introduction **3**

2 Methods **4**

2.1 Partial Derivation 4

2.2 Volterra Series 7

2.3 Backporpagation 9

3 Processes with Memeory **11**

3.1 Linear prediction 12

3.1.1 Short-term linear prediction 12

3.1.2 Parameters estimation 13

3.1.3 Long-term linear prediction 15

3.2 Reccurent Neural Network 16

3.3 Long short-term memory 18

1 Introduction

Process theory from a cybernetics viewpoint involves the study of systems, especially looking at how systems process information and maintain control through feedback mechanisms. Cybernetics, founded by Norbert Wiener, is fundamentally concerned with communication and control in animals, humans, and machines. It emphasizes the essential roles of feedback, regulation, and dynamics in systems.

Core Concepts of Process Theory in Cybernetics

Feedback Loops divided to **Negative Feedback** is used to reduce discrepancies between the actual state and desired state, stabilizing the system. It is prevalent in both biological systems (like homeostasis in human bodies) and mechanical systems (such as thermostats) and **Positive Feedback** amplifies deviations or changes, potentially leading to exponential growth or runaway effects. It's often seen in processes like population growth or economic bubbles but needs to be carefully managed to prevent instability.

Homeostasis refers to the ability of a system to maintain internal stability despite external changes. In cybernetics, this concept is crucial for understanding how biological organisms and even social systems preserve their integrity and function.

Cybernetics studies how systems adapt to their environments and learn from interactions. This involves adjusting internal mechanisms based on external changes, a principle applicable to both living organisms and artificial intelligence systems.

Systems can develop structures and patterns internally without specific, external commands or controls. This emergent behavior is central to understanding complex systems in cybernetics, from neural networks to social systems.

Control Theory a branch closely related to cybernetics, focusing on how to direct systems toward desired states. It involves designing control systems that can measure the current state, compare it to a target or set point, and act to minimize any deviations.

System Dynamics involves the study of how systems evolve over time, considering the interdependencies and feedback loops that affect their behavior. This includes looking at both linear and non-linear dynamics, which can exhibit very different behaviors under changing conditions.

Applications and Implications

- Robotics and Automation understanding process theory in cybernetics helps in designing more effective and autonomous robots that can interact dynamically with their environments.
- Biological Sciences offers insights into physiological processes and ecosystems' dynamics.
- Management and Organization helps in designing better organizational structures and management practices that can adapt and thrive in changing environments.
- Artificial Intelligence influences the development of algorithms that can learn and adapt, enhancing their decision-making capabilities.

2 Methods

Studying cybernetic processes involves a multidisciplinary approach that merges concepts from engineering, mathematics, biology, and social sciences to understand control and communication in systems. The methods used to study cybernetics are diverse, reflecting the broad scope of the field which ranges from theoretical frameworks to practical implementations.

2.1 Partial Derivation

Partial derivatives are a fundamental concept in calculus, particularly in the field of multivariable calculus. They describe how a function changes as one of its variables changes, holding the other variables constant. This is essential in many applications across physics, engineering, economics, and more, where functions often depend on more than one variable.

Basic Concept

Suppose you have a function f that depends on two variables, x and y. The partial derivative of f with respect to x measures how f changes as x changes, while keeping y constant. Similarly, the partial derivative of f with respect to y measures how f changes as y changes, while keeping x constant.

Notation

Partial derivatives can be denoted in several ways. For a function $f(x, y)$, the partial derivative with respect to x is denoted as:

$$\frac{\partial f}{\partial x} \text{ or } f_x \tag{1}$$

Similarly, the partial derivative with respect to y is denoted as:

$$\frac{\partial f}{\partial y} \text{ or } f_y \tag{2}$$

Calculating Partial Derivatives

If you have a function $f(x, y) = x^2y + 3xy^3$, the partial derivatives would be calculated as follows:

1. Partial Derivative with respect to x

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(x^2y + 3xy^3) = 2xy + 3y^3 \tag{3}$$

   Here, y is treated as a constant, and you differentiate x^2y and $3xy^3$ with respect to x.

2. Partial Derivative with respect to y

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}(x^2y + 3xy^3) = x^2 + 9xy^2 \tag{4}$$

   Here, x is treated as a constant, and you differentiate x^2y and $3xy^3$ with respect to y.

Higher-Order Partial Derivatives

Partial derivatives can also be extended to higher orders. For example, the second-order partial derivatives of f are:
The second partial derivative of f with respect to x twice:

$$\frac{\partial^2 f}{\partial x^2} \tag{5}$$

The second partial derivative of f with respect to y twice:

$$\frac{\partial^2 f}{\partial y^2} \tag{6}$$

The mixed partial derivative first with respect to x and then with respect to y:

$$\frac{\partial^2 f}{\partial x \partial y} \tag{7}$$

The mixed partial derivative first with respect to y and then with respect to x:

$$\frac{\partial^2 f}{\partial y \partial x} \tag{8}$$

Applications

Partial derivatives are used in various ways, such as:

Gradient vectors, which are vectors of partial derivatives used in optimization problems.

Jacobian matrices in multivariable transformation, which consist of first-order partial derivatives.

Hessian matrices in optimization, which are matrices of second-order partial derivatives used to assess the curvature of multivariable functions.

Understanding and calculating partial derivatives is crucial for analyzing and predicting the behavior of systems that depend on multiple variables, providing a foundation for further studies and applications in more complex mathematical, physical, and economic models.

2.2 Volterra Series

The Volterra series [10] models the output of a nonlinear system as a functional series, where the output depends not only on the current input but also on past inputs, capturing the history-dependent behavior of the system. This is crucial for accurately describing systems where the output is not merely a direct proportional response to the input.

Components of the Volterra Series:

1. Linear and Nonlinear Kernels of the Volterra series is built up of kernels of increasing order that correspond to the system's linear, quadratic, cubic, and higher-order responses. Each kernel integrates the product of input values at multiple time points, reflecting the combined effects of these inputs over time.

2. Integration over Time Delays involve the kernels integration over different combinations of time delays, representing how past inputs influence the current output. This integration makes the model particularly useful for analyzing systems with memory.

3. Expansion Terms series includes terms of increasing complexity, with each higher-order term accounting for more complex interactions between input values at different times. The first-order kernel represents the linear response, the second-order kernel captures pairwise interactions (quadratic nonlinearity), and higher-order kernels describe more complex dynamics.

Applications of Voltera Series models are in the **Signal Processing** where Volterra series are used to model and predict nonlinear distortions in signals, such as in audio processing or telecommunications, where equipment nonlinearity affects the signal integrity then in **Control Systems** in control theory helps in designing controllers that can handle nonlinear dynamics effectively, ensuring stability and performance in nonlinear control systems. In the **Biological Systems** are applied in modeling the nonlinear responses of biological systems, such as the neural response functions in sensory systems where the output response to stimuli is inherently nonlinear.

As the order of interaction increases, the computational complexity of the model can grow significantly, making higher-order models difficult to manage. Accurately estimating the higher-order kernels requires extensive data, and the model may suffer from overfitting if not properly regularized or if the data is not sufficient.

The Volterra series provides a powerful framework for understanding and predicting the behavior of nonlinear systems, though its practical application requires careful consideration of its complexity and the computational resources available.

Volterra Series Expansion

The output $y(t)$ of a nonlinear system in response to an input $x(t)$ can be expressed as:

$$y(t) = h_0 + \sum_{n=1}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} h_n(\tau_1, \tau_2, \ldots, \tau_n) x(t-\tau_1) x(t-\tau_2) \cdots x(t-\tau_n) \, d\tau_1 \, d\tau_2 \cdots d\tau_n, \tag{9}$$

where, $h_n(\tau_1, \tau_2, \ldots, \tau_n)$ are the Volterra kernels, which are functions characterizing the system's response at different orders of interaction between time-shifted inputs. Each kernel h_n corresponds to the n-th order of nonlinearity in the system's response:

h_0 is a constant term, often representing the system's output at zero input. $h_1(\tau_1)$ is the first-order kernel or linear response of the system. $h_2(\tau_1, \tau_2)$ is the second-order kernel, capturing the pairwise nonlinear interactions. $h_3(\tau_1, \tau_2, \tau_3)$* and higher-order kernels account for increasingly complex interactions.

First and Second-Order Kernels

For practical applications, the Volterra series is often truncated after a few terms. Here's how the first and second-order terms are typically expressed:

First-order (linear) response:

$$y_1(t) = \int_{-\infty}^{\infty} h_1(\tau_1)x(t-\tau_1)\,d\tau_1 \tag{10}$$

Second-order (quadratic) response:

$$y_2(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)\,d\tau_1\,d\tau_2 \tag{11}$$

2.3 Backporpagation

Backpropagation is a method commonly used in artificial neural networks to train the network by adjusting the weights of the synapses. This adjustment is done by comparing the network's actual output to the desired output and updating the weights accordingly, allowing the network to learn and improve its performanceover time. Backpropagation is a fundamental algorithm in the field of artificial neural networks [6]. Backpropagation is a fundamental algorithm in the field of artificial neural networks, allowing for the training and improvement of the network by adjusting the weights of thesynapses based on the error between the actual output and the desired output. The algorithm works by propagating the error backwards through the network, hence its name "backpropagation" [2]. The backpropagation algorithm is a crucial tool in training artificial neural networks. It allows for the optimization of network performance by adjusting synapse weights based on the error between actual and desired outputs. Furthermore, backpropagation has demonstrated its power in optimizing the performance of neural networks by allowing psychologists to design networks capable of performing complex computations in unexpected ways and uncovering insights from data.

1. Forward Pass
   During the forward pass, the input data is fed through the neural network, and the output is computed through successive activations of the network layers. For each layer l, the input $z^{(l)}$ and output $a^{(l)}$ are computed as follows:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}, \tag{12}$$

$$a^{(l)} = \sigma(z^{(l)}), \tag{13}$$

   where:
   $W^{(l)}$ is the weight matrix for layer l,
   $b^{(l)}$ is the bias vector for layer l,
   $a^{(l-1)}$ is the output of the previous layer,
   σ is the activation function applied element-wise to the input.

2. Backward Pass (Error Backpropagation)
   During the backward pass, the gradients of the loss function with respect to the weights and biases of the network are computed using the chain rule of calculus.
   Compute the output layer error $\delta^{(L)}$ as the derivative of the loss function with respect to the output activations:

$$\delta^{(L)} = \nabla_a L \odot \sigma'(z^{(L)}) \tag{14}$$

   Backpropagate the error through the network layers to compute the errors $\delta^{(l)}$ for each hidden layer:

$$\delta^{(l)} = ((W^{(l+1)})^T \cdot \delta^{(l+1)}) \odot \sigma'(z^{(l)}) \tag{15}$$

3. Gradient Descent Update
   Use the computed errors to update the weights and biases of the network using gradient descent or its variants. Update the weights and biases of each layer using the gradients of the loss function with respect to the weights and biases:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} \cdot (a^{(l-1)})^T, \tag{16}$$

$$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}, \tag{17}$$

where:
L is the loss function,
$\odot$ denotes element-wise multiplication,
$\nabla_a L$ represents the gradient of the loss function with respect to the output activations,
σ' denotes the derivative of the activation function,
$\frac{\partial L}{\partial W^{(l)}}$ and $\frac{\partial L}{\partial b^{(l)}}$ represent the
gradients of the loss function with respect to the weights and biases of layer l, respectively.

These equations describe the key steps involved in the backpropagation algorithm for training neural networks [6].

3 Processes with Memeory

Processes with memory, often referred to in scientific and engineering disciplines as systems with memory or memory-dependent processes, are those in which the future state of the system is dependent not only on its current state but also on its past states. This dependency reflects the historical influence of previous inputs or conditions on the present behavior of the system. Such processes are inherently more complex than memoryless processes, where the future state depends solely on the current state.

Characteristics of Processes with Memory
Memory Effects: These are the primary characteristic of such systems, where past inputs or actions continue to influence the system's behavior after they have occurred. This can be seen in materials with hysteresis, charge in capacitors, or the psychological state of an individual influenced by past experiences.
Time-Dependence system's output is not only a function of the current input but also of inputs at previous times. This dependency can often be described using integral equations or differential equations with after-effects.

State Variables in processes with memory, the state of the system is described by variables that accumulate information about past interactions, which can include the history of various physical or non-physical quantities.

Types of Memory
Short-Term Memory in systems like electrical circuits (capacitors, inductors) or certain mechanical systems (elastic aftereffect), where the memory of the system only persists for a short duration relative to the time scale of interest.

Long-Term Memory seen in systems where past states significantly influence future states over extended periods, such as in climate systems, economic models, or in certain biological processes.

3.1 Linear prediction

Linear prediction is a method, that is extensively employed in signal processing to forecast future values of a time series by analyzing past observations. This technique is based on the premise that the current sample of a signal can be described by a linear combination of previous values complemented by a noise term [9, 4]. Long-term linear prediction extends this methodology using not only samples from the current past but also long-term instances, a longer period away, such as months or years. Although requiring a greater volume of data and increasing the complexity in comparison to short-term prediction, its applicability finds use in diverse fields such as weather forecasting, finance, economics and marketing to make long-term projections about variables such as sales, demand, or stock prices, with the goal to provide a comprehensive and accurate forecast that can be used to make strategic decisions in the long term [7].

3.1.1 Short-term linear prediction

Short-term linear prediction focuses on forecasting the immediate future values in a sequence. The model representation involves approximating a time series as a linear combination of its past values, providing a foundation for estimating the next value [8]. Is a technique commonly used in signal processing and time-series analysis to predict future values in a sequence based on a linear model. The basic idea is to use past observations in a sequence to estimate the next value. Linear prediction is a statistical technique used to forecast future values based on past observations. It is a method for modeling the relationship between a dependent variable and one or more independent variables in a linear form. The goal of linear prediction is to find the best linear approximation of the relationship between the variables, which can then be used to make predictions about future values of the dependent variable.

3.1.2 Parameters estimation

Parameter estimation in linear prediction involves determining the coefficients of a linear model that best represents a given dataset. In the context of linear prediction, this typically refers to estimating the coefficients of a linear filter that minimizes prediction error.

Least Squares Estimation
Minimize the sum of squared prediction errors between the model predictions and the actual observations.

Maximum Likelihood Estimation
Find the parameters that maximize the likelihood of observing the given data under the assumed model distribution.

Yule-Walker Equations
For AR models, the Yule-Walker equations provide a set of linear equations that can be solved to directly estimate the AR coefficients.

Burg Algorithm
An iterative algorithm that estimates AR coefficients by minimizing a reflection coefficient criterion.

Levinson-Durbin recursion
specific algorithm used for parameter estimation in linear prediction, particularly for autoregressive (AR) models

Assume we have a time series $x[n]$ that we want to model using an autoregressive model of order p, which can be represented as:

$$x[n] = \sum_{i=1}^{p} a_i x[n-i] + w[n], \tag{18}$$

where
a_i are the coefficients of the AR model,
$w[n]$ is white noise, and p is the order of the model.

Toeplitz Matrix Formation arrange the autocorrelation values of the time series into a Toeplitz matrix R of size $(p+1) \times (p+1)$. The i-th row and j-th column of this matrix represent the autocorrelation value at lag $|i-j|$.

The Levinson-Durbin algorithm is a recursive method that efficiently computes the AR coefficients a_i using the autocorrelation values stored in the Toeplitz matrix R. The recursion starts with an initial guess for the first coefficient a_1 and iteratively computes the subsequent coefficients using the following steps:

- set $k = 1$ and $E_0 = R[0,0]$.
- compute the forward prediction error ϵ_k and reflection coefficient k_k using the formula:

$$\epsilon_k = R[k,k] - \sum_{j=1}^{k-1} a_j R[k-j,k] \tag{19}$$

$$k_k = \frac{\epsilon_k}{E_{k-1}} \tag{20}$$

- update the AR coefficients a_k using the reflection coefficient k_k:

$$a_k = k_k \tag{21}$$

- update the prediction error E_k:

$$E_k = (1 - k_k^2)E_{k-1} \tag{22}$$

- increment k and repeat the process until all coefficients are computed.

Once all p coefficients are computed, you obtain the AR model parameters $a_1, a_2, ..., a_p$. The Levinson-Durbin recursion provides an efficient way to estimate the parameters of an autoregressive model by recursively solving linear equations, making it widely used in signal processing, time series analysis, and speech processing applications.

Overall, parameter estimation in linear prediction involves a systematic approach to finding the coefficients of a linear model that best fits the data and minimizes prediction error. It requires careful consideration of the model assumptions, choice of estimation method, and evaluation of model performance.

3.1.3 Long-term linear prediction

Long-term linear prediction (LTLP) refers to the use of linear prediction techniques to make predictions about the future values of a time series over an extended period of time [3], typically several months to several years. Unlike short-term linear prediction, which focuses on forecasting the near-term future, long-term linear prediction aims to provide a more comprehensive and accurate forecast of future values. Combining these two components, short-term and long-term, one obtains the relation [9, 1]:

$$\hat{x}(n) = \sum_{i=1}^{p} a_i x(n-i) + \sum_{i=-q}^{q} b_i x(n-i-Q), \tag{23}$$

where $\hat{x}(n)$ is the predicted value of the signal at instance n, p is the order of the short-term predictor, that uses p past samples, q is the order of the long-term predictor, that uses $2q+1$ samples a pitch period (Q) away, and a_i, b_i, are the short-term predictor coefficients, the long-term predcitor coefficients, respectively (see Fig. 1).

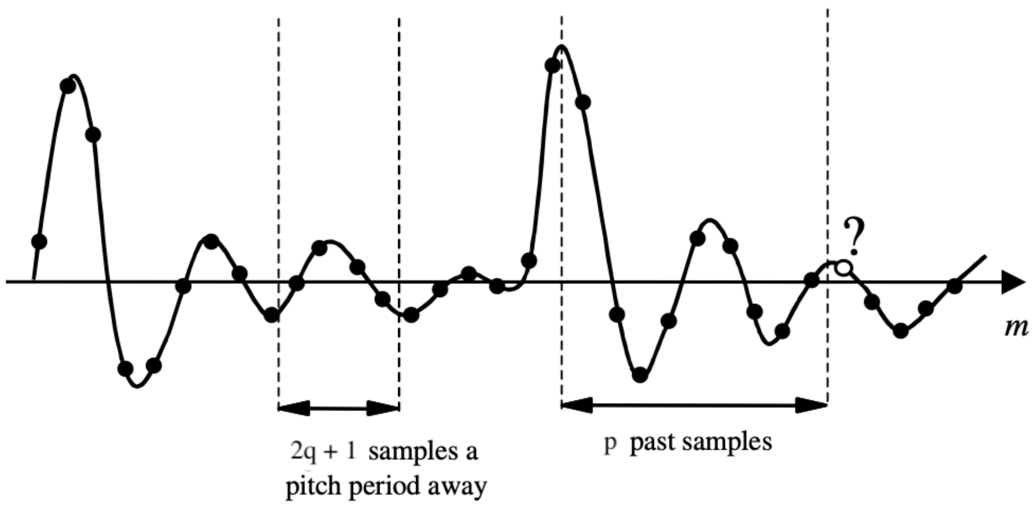

Figure 1: Pitch-lag in long-term prediction [3].

3.2 Reccurent Neural Network

In the realm of artificial intelligence and machine learning, recurrent neural networks (RNNs) have emerged as powerful tools for modeling and understanding sequential data. Unlike traditional feedforward neural networks, RNNs possess the unique ability to capture temporal dependencies and dynamic patterns inherent in sequential data, making them well-suited for a wide range of applications across various scientific domains.
The ability to comprehend and analyze sequential data is fundamental to understanding the dynamics of complex systems in various scientific disciplines. Traditional machine learning techniques often struggle to capture the temporal dependencies present in such data, hindering their effectiveness in modeling and prediction tasks. Recurrent neural networks (RNNs) have emerged as a transformative paradigm for addressing this challenge, offering a powerful framework for modeling sequential data with dynamic temporal patterns. By introducing feedback loops that enable information to persist over time, RNNs excel at processing sequences of data points, making them invaluable tools for scientific research across diverse domains.

Architecture and Mechanisms
At the core of RNNs lies a recurrent structure that allows them to maintain

an internal state or memory, enabling them to capture long-range dependencies in sequential data. This architecture facilitates the processing of input sequences of varying lengths and the generation of output sequences with flexible lengths. The key mechanism driving RNNs is the recurrent connection, which enables information to flow through the network over multiple time steps. Through the use of specialized activation functions such as the long short-term memory (LSTM) and gated recurrent unit (GRU), RNNs can effectively manage and update their internal states, mitigating the vanishing gradient problem and enhancing their ability to capture long-term dependencies.

The equations of a basic Recurrent Neural Network (RNN) model:

1. Hidden State Update At each time step t, the hidden state $h^{(t)}$ of the RNN is updated based on the input at the current time step $x^{(t)}$ and the previous hidden state $h^{(t-1)}$. This update can be represented by the following equation:

   $$h^{(t)} = \sigma(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h), \tag{24}$$

   where:
   $h^{(t)}$ is the hidden state at time step t,
   $x^{(t)}$ is the input at time step t,
   W_{hx} is the weight matrix connecting the input to the hidden state,
   W_{hh} is the weight matrix connecting the hidden state to itself (recurrent weights),
   b_h is the bias vector,
   σ is the activation function applied element-wise to the input.

2. Output Calculation The output $y^{(t)}$ of the RNN at each time step can be computed based on the current hidden state $h^{(t)}$. This can be represented by the following equation:

   $$y^{(t)} = \sigma(W_{hy}h^{(t)} + b_y), \tag{25}$$

   where:
   $y^{(t)}$ is the output at time step t,
   W_{hy} is the weight matrix connecting the hidden state to the output,
   b_y is the bias vector.

3. Initial Hidden State The initial hidden state $h^{(0)}$ is typically set to a vector of zeros or initialized randomly.

4. Activation Function The activation function σ is usually a nonlinear function such as the hyperbolic tangent (tanh) or the rectified linear unit (ReLU).

   These equations describe the basic forward pass of a recurrent neural network. During training, the parameters (weights and biases) of the network, W_{hx}, W_{hh}, W_{hy}, b_h, and b_y, are adjusted using backpropagation through time (BPTT) to minimize a specified loss function. This enables the RNN to learn to capture temporal dependencies and make predictions based on sequential data.

3.3 Long short-term memory

Long Short-Term Memory (LSTM) networks [5] are a special kind of Recurrent Neural Network (RNN) capable of learning long-term dependencies in sequence prediction problems. This is crucial in many deep learning tasks where classical RNNs could fail due to the vanishing gradient problem—as gradients are propagated back through the network, they can become vanishingly small, effectively preventing the network from learning long-range dependencies.

LSTMs were introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997 to specifically address this issue. They incorporate mechanisms called gates that regulate the flow of information [5].

Structure An LSTM unit typically consists of a cell (which contains the state of the network) and three gates:

- Forget gate decides what information is discarded from the cell state.
- Input gate decides which values will be updated.
- Output gate decides what the next hidden state should be.

Mathematical Formulations

Let's define the following:
x_t: input vector at time step t
h_{t-1}: hidden state from the previous time step
C_{t-1}: cell state from the previous time step
σ: sigmoid activation function
tanh: hyperbolic tangent activation function
W, U and b: parameter matrices and vector biases that are learned during training

The equations governing the forward pass in an LSTM cell are:

Forget Gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{26}$$

This gate decides which information to throw away from the cell state.

Input Gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{27}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{28}$$

The input gate layer decides which values will be updated, and a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.

Cell State Update

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{29}$$

The old cell state, C_{t-1}, is updated to the new cell state C_t. The forget gate's output, multiplied by the previous cell state, decides what to forget, and the input gate's output, multiplied by the candidate values, decides what new information to add.

Output Gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{30}$$

$$h_t = o_t * \tanh(C_t) \tag{31}$$

The output gate decides what the next hidden state, h_t, should be. The hidden state contains information on previous inputs. The contents of the cell state are shaped by a tanh function to ensure they stay between -1 and 1 and are multiplied by the output of the output gate, so that only the necessary information is passed on. LSTMs are designed to avoid the long-term dependency problem, allowing them to remember information for extended periods. Due to their effectiveness, they are widely used in tasks that involve sequences, such as speech recognition, language modeling, and text generation. The unique architecture of LSTMs makes them suited for these kinds of tasks where context from the distant past is crucial for understanding the current element in the sequence.

References

[1] DN BAKER et al. “LINEAR PREDICTION FILTER ANALYSIS OF RELATIVISTIC ELECTRON PROPERTIES AT 6.6 RE”. English. In: *JOURNAL OF GEOPHYSICAL RESEARCH-SPACE PHYSICS* 95.A9 (Sept. 1990), pp. 15133–15140. ISSN: 2169-9380. DOI: `10.1029/JA095iA09p15133`.

[2] Daewon Chung and Insoo Sohn. “Neural Network Optimization Based on Complex Network Theory: A Survey”. English. In: *MATHEMATICS* 11.2 (Jan. 2023). DOI: `10.3390/math11020321`.

[3] Jonathan D. Cryer and Kung-Sik Chan. “Time Series Analysis With Applications in R Second Edition INTRODUCTION”. English. In: *TIME SERIES ANALYSIS: WITH APPLICATIONS IN R, SECOND EDITION*. Springer Texts in Statistics. 233 SPRING STREET, NEW YORK, NY 10013, UNITED STATES: SPRINGER, 2008, pp. 1+. ISBN: 978-0-387-75958-6.

[4] Livio Fenga. “A wavelet threshold denoising procedure for multimodel predictions: An application to economic time series”. English. In: *STATISTICAL ANALYSIS AND DATA MINING* 10.6 (Dec. 2017), pp. 410–421. ISSN: 1932-1864. DOI: `10.1002/sam.11351`.

[5] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.1735-1780 (1997).

[6] Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. “Predictive Coding Approximates Backprop Along Arbitrary Computation Graphs”. English. In: *NEURAL COMPUTATION* 34.6 (May 2022), pp. 1329–1368. ISSN: 0899-7667. DOI: `10.1162/neco\_a\_01497`.

[7] MH PESARAN, RG PIERSE, and MS KUMAR. “ECONOMETRIC-ANALYSIS OF AGGREGATION IN THE CONTEXT OF LINEAR PREDICTION MODELS”. English. In: *ECONOMETRICA* 57.4 (July 1989), pp. 861–888. ISSN: 0012-9682. DOI: `10.2307/1913775`.

[8] Fotios Petropoulos et al. “Forecasting: theory and practice”. English. In: *INTERNATIONAL JOURNAL OF FORECASTING* 38.3 (July 2022), pp. 705–871. ISSN: 0169-2070. DOI: `10.1016/j.ijforecast.2021.11.001`.

[9] P.P. Vaidyanathan. *The Theory of Linear Prediction*. Synthesis Lectures on Signal Processing. Morgan & Claypool Publishers, 2007. ISBN: 9781598295764. URL: `https://books.google.sk/books?id=6bJeAQAAQBAJ`.

[10] N. Wiener. *Nonlinear Problems in Random Theory*. Technology Press of MIT and John Wiley & Sons, 1958.