FERNUNIVERSITÄT IN HAGEN

APPLIED STATISTICS

Bachelor Thesis

Advanced Architectures in LSTM Networks for Effective Management of Temporal Data

Luca Zangari

" Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world. Science is the highest personification of the nation because that nation will remain the first which carries the furthest the works of thought and intelligence."

Louis Pasteur

Acknowledgments

Abstract

Contents

1. Introduction

Accurate temporal data forecasting has broad applications across various sectors, from energy to finance and from healthcare to advertisement, and effective management of this data is essential for optimizing operations in those scenarios. In reality, predictions of long sequential time-series data present challenges due to inherent complexities and the dynamic nature of its information. Dependencies can span from short-term daily fluctuations to long-term seasonal trends, complicating the forecasting process as capturing all relevant information becomes difficult. Non-stationarity, where statistical properties such as mean and variance change over time, is an additional hurdle, necessitating models that are adaptable to temporal changes and must address irregular events and noise, to maintain accuracy. The complexity is further increased by interdependencies among variables, creating a polymorph and multidimensional problem. With advancements in machine learning, the ability to forecast temporal data has seen significant improvements, despite the aforementioned challenges and limitations. Historically, the first applicable research efforts of time-series forecasting, have been autoregressive integrated moving average (ARIMA) models which assumed linear relationships within the data that simplified the prediction process. However, ARIMA and its variants, such as Seasonal ARIMA (SARIMA), often fell short when dealing with complex, nonlinear relationships naturally found in many real-world temporal datasets. To this front, Artificial Neural Networks (ANNs) have become increasingly applicable, chiefly because the underlying assumption here, is the one of non-linearity. These networks have proven particularly useful in extracting information when handed large amounts of data in fields like image and speech recognition, natural language processing, and financial forecasting. Among neural networks, RNN stand out due to their added benefit of backpropagation through time. RNNs are able to handle variable-length sequences where the output at a given time step depends not just on the current input but also on previous information. With

their memory capability derived from the feedback loop, RNN have found applications in sequential time series prediction tasks where order of output and context is of non-negligible importance. However, RNNs tend to struggle with learning dependencies on a long term, an issue which has been mitigated by the introductions of extension models know as Long Short-Term Memory networks (LSTMs) that were first designed by Hochreiter and Schmidhuber[1] to address the issue of gradient flow behaviour in RNNs. LSTMs outperform traditional methods like ARIMA in handling complex, nonlinear relationships within data, as evidenced by the work of Kong et. al.[2] on the forecasting of electrical grid load. The ability of LSTMs to remember information over extended periods makes them heterogeneously applicable: they enhance text classification, language modeling, and environmental data forecasting[3], improve prediction of production efficiency[4] and support cybersecurity for attack detection in IoT environments[5], just to name a few. Integration of advanced architectures like attention mechanisms and sequence-to-sequence (Seq2Seq) models with LSTMs networks enhances their performance in handling complex sequential data even further. The Seq2Seq model, which includes an encoder-decoder structure, has been particularly successful in tasks like machine translation and time series forecasting, while the attention mechanisms allow the model to focus on relevant parts of the input sequence during decoding. These qualities are an opportunities for applications in accurate energy usage prediction. This is especially important considering the increasing global population and urbanization (4.5 Billion of "urban citizen" in 2023 compared to the 2.28 billion in 1990[6]) which surged the demand for reliable electricity supply has surged[7, 8], necessitating accurate load forecasting to maintain system stability amidst the integration of renewable energy sources. The selective focus of attention mechanisms in LSTM leads to more accurate predictions and allows for better resource allocation and load balancing in power grids[9]. This research aims to demonstrate the superiority of LSTM networks in managing long-range temporal dependencies and their practical application in predicting short-term imbalances in German electricity generation and consumption. By integrating an attention mechanism within a sequence-to-sequence architecture, the study seeks to advance the understanding and implementation of LSTM networks in addressing real-world

energy sector challenges.

"Artificial intelligence is the science of making machines do things that would require intelligence if done by humans."
— John McCarthy

2. Theory and literature review

The present chapter outlines the theoretical concepts of Neural Netowrks (NNs) and provides a comprehensive review of the literature to delineate the advanced architecture of long short term memory (LSTM) networks. It begins with the fundamentals of NNs, including their elements, basic structures, and training mechanisms. The chapter follows the progression from multilayer perceptrons (MLPs) networks to Recurrent Neural Networks (RNNs) through the principle of parameter sharing. An extensive discussion of RNNs architecture and functionality is also presented, detailing the mathematical challenges in learning long-term dependencies. Regularization techniques such as skip connections and teacher forcing are further examined to improve model performance and stability. The chapter continues by introducing LSTM networks, which incorporate gated mechanisms to effectively model sequences and manage long-term dependencies in temporal data. It explores various advanced LSTM architectures, including Bidirectional LSTM (BiLSTM) and Extended Long Short-Term Memory (xLSTM), as well as the integration within sequence-to-sequence (Seq-to-Seq) frameworks. In addition, the chapter discusses the mechanisms of attention and their role in making LSTM networks more capable of handling long sequences. The subsequent chapter (Chap. 3) will then integrate and implement these advanced components into Seq-to-Seq models to demonstrate the practical application of these networks.

2.1. Neural Networks fundamentals

Neural networks represent foundational models in artificial intelligence, inspired by the architecture and processes of the human brain and capable of handling complex tasks. This inspiration drives the development of artificial neurons and networks that can learn, adapt, and make decisions that resemble

human cognitive processes[10]. These networks are structured in multiple layers of interconnected nodes, or neurons, that process input data through various transformations[11]. The term neural networks originated with the work of McCulloch and Pitts[12] in 1943, who proposed a theoretical model of an artificial neuron that combined inputs to produce an output. However, their model did not include a practical implementation. The first practical learning algorithm, introduced by Rosenblatt[13] in 1958, was the perception, designed to mimic the way biological neurons operate. It consisted of an input layer, weights, a bias term, and an activation function. Mathematically, perception can be described as

$$y = f(\sum_{i=1}^{n} w_i x_i + b)\,, \tag{2.1}$$

where the outcome y, is a function of the input features x_i, which are influenced by the weights, w_i, and the bias term b. The activation function f, is an operation that transforms the input features into the output values[14]. The perception model was capable of solving simple linear classification problems by adjusting the weights to minimize the classification errors. However, its capacity was constrained to linearly separable data, as evidenced by the XOR problem, which necessitated the development of more complex models in neural networks[15].

Feedforward Neural Networks (FNNs) represent a fundamental class of neural networks that are widely utilized for supervised learning tasks, where the model is trained on input-output pairs to infer a mapping function that generalizes to unseen data. They are characterized by unidirectional information flow, from input to output, without forming cycles[16]. The historical development of FNNs can be traced back to 1960s when Minsky and Papert[17] evaluated the limitations of perceptrons in addressing non-linear problems. They proposed that the incorporation of hidden layers, along with non-linear activation functions, could significantly enhance the network's capacity to learn complex input-output relationships. This insight led to MLPs (Figure 2.1), a particular subclass of FNNs characterized by their layered architecture, which includes an input layer, one or several intermediate hidden layers, and an output layer.

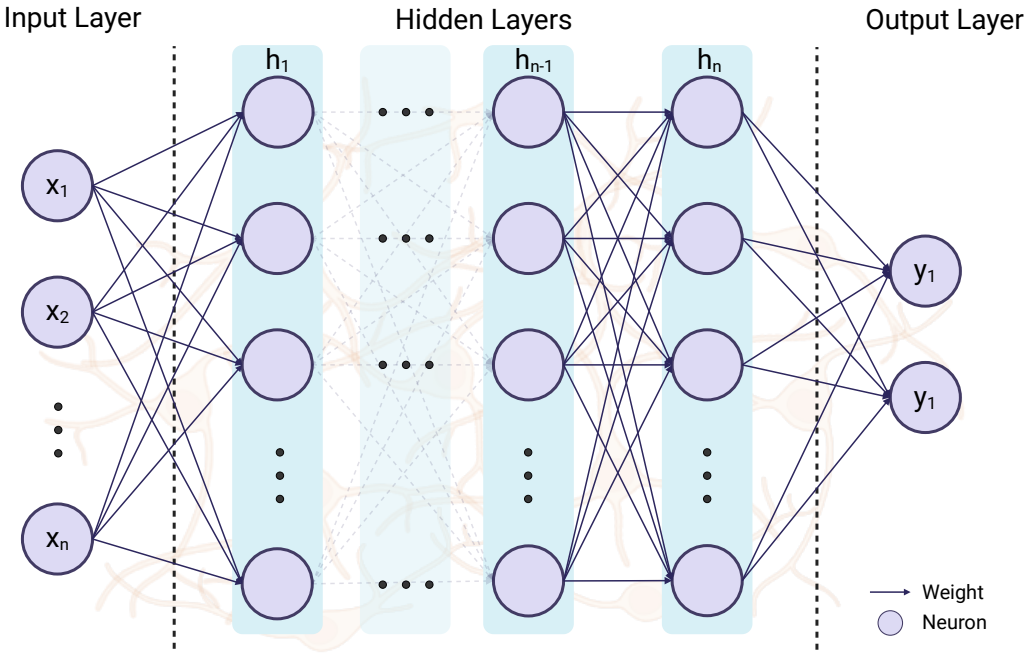

Figure 2.1 A schematic representation of MLPs architecture with a single input, multiple hidden layers, and an output layer. Each node within these layers, referred to as neurons, is connected to neurons in the subsequent layer via weighted connections, illustrated by arrows. The hidden layers apply activation functions to transform the input data, enabling the model to unravel the non-linear relationships between features. In this architecture, the number and the size of hidden layers and units are variables, while the output layer delivers the final predictions, which can be either continuous values for regression tasks or discrete classes for classification tasks.

MLPs leverage activation functions to model non-linear relationships[18, 10].

2.1.1. Activation Functions

The earliest activation function used in the perceptron model was the step function[16], which outputs either 0 or 1 based on whether the input is below or above a certain threshold (θ):

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases} \tag{2.2}$$

However, the sigmoid function gained popularity in the 1980 for tackling non-linearities[19]. This function maps any real-valued number to a value within the interval of 0 and 1, allowing MLPs to solve more complex problems[10] and is given by

$$\sigma(x) = \frac{1}{1 + e^{-x}}\,. \tag{2.3}$$

Despite their utility, the sigmoid function can impede the learning process of deep networks due to issues such as the vanishing gradient problem. During the same historical period, the function hyperbolic tangent (tanh), a scaled and shifted version of the sigmoid function, was adopted. It can be written as

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}\,. \tag{2.4}$$

This function maps the input values to a range between -1 and 1, often resulting in faster convergence and greater stability during training compared to the sigmoid function[20]. The rectified linear unit (ReLU) function, which was less prevalent in the early days of neural networks[21], was later revisited and gained attention from researchers such as Jarret et al.[22] in 2009 and Nait and Hinton[23] in 2010. This activation function outputs the input directly if it is positive; otherwise, it outputs zero and is expressed as:

$$\mathrm{ReLU}(x) = \max(0, x)\,. \tag{2.5}$$

The ReLU addresses the limitations of earlier functions and has become the default choice for many neural networks[20]. To allow gradient flow for negative values, later variants such as the leaky ReLU[24] (Eq.2.6) and Exponential Linear Unit (ELU)[25] (Eq.2.7) were introduced in 2013 and 2015, respectively. The leaky ReLU introduces a small, positive slope for negative input values, whereas ELU outputs an exponential function. The ELU tends to bring the mean activation closer to zero, thus accelerating the learning process[25]. An overview of these

activation functions can be seen in Figure 2.2.

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{2.6}$$

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \tag{2.7}$$

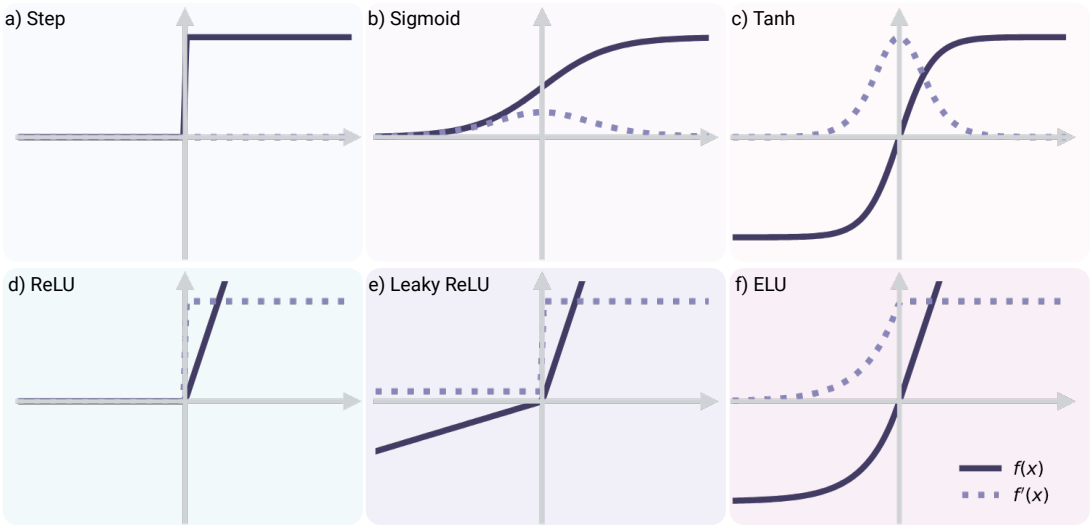

Figure 2.2 Illustration of six exemplary activation functions and their derivatives: a) Step function outputs 1 for positive inputs and 0 otherwise, with its derivative being zero except at the step change. b) Sigmoid function maps inputs to the interval of $[0, 1]$, with its derivative peaking at the inflection point. c) Tanh function produces outputs between -1 and 1, with its derivative highest at the origin. d) The ReLU function maps negative inputs to zero, with a derivative of zero for negative inputs and one for positive inputs. e) Leaky ReLU allows a small, positive derivative for negative inputs, beneficial for gradient flow during the training of neural networks, and one for positives. f) ELU outputs the input itself for positive values and scales the negative ones exponentially, with a derivative that transitions smoothly around zero, which is advantageous for reducing the bias shift during training.

2.1.2. Deep Neural Networks

Networks with a single hidden layer are typically referred to as shallow neural networks. In contrast, Deep Neural Networks (DNNs) extend the FNNs architec-

ture by integrating multiple hidden layers, the basis of the nomenclature "deep" architectures[26]. The term "deep learning" was first introduced by Dechter[27] in 1986, reflecting the depth and complexity of these models. The additional hidden layers and activation functions enable DNNs to learn hierarchical characteristics, capturing intricate patterns and abstract representations at multiple levels[28]. This depth allows DNNs to approximate any continuous function, a principle defined in the universal approximation theorem, as proposed by Cybenko[19] and expanded by Hornik et al.[29] in 1989. The proliferation of these networks has been driven mainly by significant advancements in computational power, notably the Graphics Processing Units (GPUs) of 2006 onward, which have substantially accelerated the training processes of deep networks. Additionally, the availability of large-scale datasets has enabled the training of complex models[30].

2.1.3. Training Neural Network

Training a neural network requires identifying the optimal parameters for mapping inputs to outputs in the best possible manner. This process begins with the forward pass, where input data passes through the network layer by layer, ultimately producing predictions at the output layer[10]. A loss function, also known as a cost function $\mathcal{L}(\theta)$, is then utilized to evaluate the network's performance and quantify the discrepancy between predictions ($\hat{y}_i(\theta)$) and the ground truths ($y_i(\theta)$). For regression tasks, which include the prediction of continuous numerical variables, commonly used loss functions include Mean Square Error (MSE), which estimates the average squared error and is beneficial in penalizing larger errors[31] and is expressed as

$$\mathcal{L}_{\text{MSE}}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i(\theta) - \hat{y}_i(\theta))^2\,. \tag{2.8}$$

Another widely applied loss function is Mean Absolute Error (MAE), which quantifies the average of absolute errors and offers a more balanced measure of

prediction accuracy[31, 32]:

$$\mathcal{L}_{\text{MAE}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} |y_i(\theta) - \hat{y}_i(\theta)| \,. \tag{2.9}$$

On the other hand, Mean Absolute Percentage Error (MAPE), is another variation that calculates the mean of absolute percentage errors, provides a scale-independent metric that is advantageous for comparing prediction accuracy across different scales[33], and is written as

$$\mathcal{L}_{\text{MAPE}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \left| \frac{y_i(\theta) - \hat{y}_i(\theta)}{y_i(\theta)} \right| \times 100\% \,. \tag{2.10}$$

In scenarios where the implications of predictions can vary significantly with the range of possible outputs, it is essential to understand the variability and uncertainty of predictions. In such cases, predicting the distribution of target values rather than a single-point estimate is preferable[34, 35]. Initially proposed by Koenker and Hallock[34] in 2001, the quantile loss function addresses this need by weighting deviations in one direction, which makes it especially relevant for these applications. This loss can be expressed as

$$\mathcal{L}_{\tau}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \left[\tau(y_i(\theta) - \hat{y}_i(\theta))_+ + (1-\tau)(\hat{y}_i(\theta) - y_i(\theta))_+\right] , \tag{2.11}$$

where τ is the quantile to be estimated, and $(x)_+$ refers to the positive part of the x. The recent research by Rodrigues and Pereira[36] in 2020 demonstrated that predicting multiple quantiles can explicate the overall distribution shape, indicating a more comprehensive understanding of prediction uncertainty.

To minimize the loss function, the gradients of the loss for the network parameters are calculated to fulfill the optimization objective. This process is referred to as "learning the network parameters" or "fitting the model"[31]. The gradients, denoted as $\nabla_\theta L(\theta)$, indicate the direction in which the parameters need to be adjusted to reduce the loss[16], ultimately identifying the optimal parameter set

θ^*:

$$\theta^* = \arg\min_{\theta} L(\theta)\,. \tag{2.12}$$

The iterative process of updating parameters based on these gradients follows the update rule

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)\,, \tag{2.13}$$

where α is the learning rate that controls the magnitude of the parameter update and can be either fixed or dynamic[16]; the latter leverages scheduling techniques that adjust α over time to enhance convergence[37].

A fundamental mechanism in the optimization process is backpropagation, a term coined by Rumelhart et al.[38] in 1985. Backpropagation estimates the gradient of the loss function relative to each parameter utilizing the chain rule, allowing the error to propagate backward through the network, layer by layer[39]. The chronological evolution of these algorithms can be traced back to Cauchy[40] in 1847, who first coined the term "gradient descent". Over the years, various enhancements to gradient descent have been developed to improve its convergence properties[10]. For instance, Stochastic Gradient Descent (SGD) updates the parameters using a subset of the data, also known as a mini-batch, rather than the entire dataset[41]. Momentum-based methods were further enhanced SGD by guiding the optimization process along relevant directions and reducing oscillations (Figure 2.3). The introduction of various optimizer variants, including AdaGrad by Duchi et al.[42] in 2011 and RMSProp by Hinton et al.[43] in 2012, constituted significant milestones in optimization techniques. In 2015, Kingma and Ba[44] introduced the Adaptive Moment Estimation (Adam), which benefited from both adaptive learning rates and momentum. The Adam maintains per-parameter learning rates that adjust based on the first and second moments of the gradients, ensuring more stable and rapid convergence (Figure 2.3). The

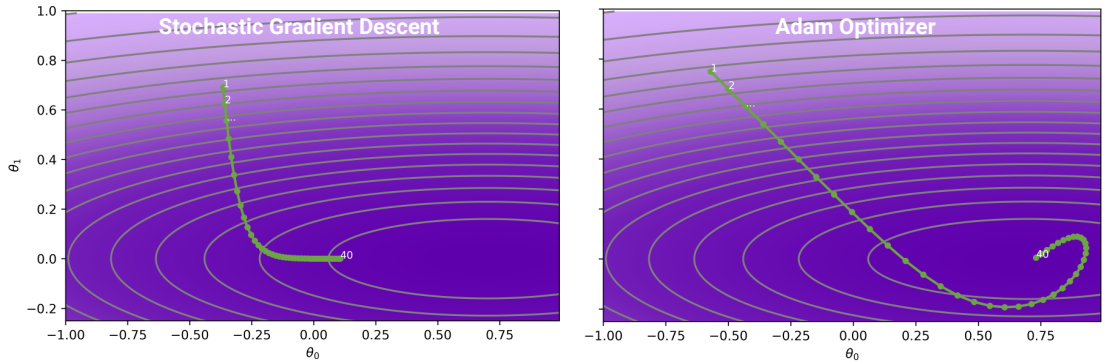

Figure 2.3 Visualization of the optimization trajectories of a linear regression model with θ_0 and θ_1 as the intercept and slope, respectively, using SGD and Adam optimizers, with 0.07 as the learning rate. The heatmap represents the loss values, with darker shades indicating lower loss. The green points illustrate the gradient values at every iteration. a) SGD optimizer shows faster change in the vertical directions while slower progress horizontally. b) Adam optimizer, on the other hand, provides a smoother path and converges more quickly to the minimum value around $(0.7, 0)$.

update rule for Adam is given by

$$\theta \leftarrow \theta - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.14}$$

where $\hat{m}_t$ and $\hat{v}_t$ are adjusted estimates of the gradients' first and second moments to correct for bias, respectively, and ϵ is a small constant to ensure numerical stability. The continuous application of these gradient-based update rules allows neural networks to iteratively optimize their parameters, minimize the loss function, and enhance the model's accuracy. This process is fundamental to the training of deep neural networks[44].

2.2. Sequence modeling

2.2.1. Recurrent Neural Networks

FNNs have demonstrated effectiveness across a variety of applications. However, they have significant shortcomings in handling sequential data due to their independent processing of input features and lack of mechanisms to account for temporal dependencies[10]. This limitation poses a challenge for tasks involving

time series data, where the context of previous inputs is essential.

RNNs were introduced by Rumelhart et al.[39] in 1986 to overcome this problem. RNNs integrate recurrent connections by incorporating a feedback loop into standard neural networks, thus creating a memory mechanism that retains information from past sequences. This feedback loop, or internal state, gives RNNs a recursive nature, allowing the network to utilize the same set of parameters, including weights and biases, across different time steps. Unlike traditional FNNs, where each layer is defined by its own unique parameters[28], RNNs can capture the dynamic nature of input sequences and learn from temporal data, recognizing patterns irrespective of their position within the sequence[39]. The recursive nature of RNNs can be mathematically expressed as

$$h_t = f(h_{t-1}; x_t; \theta)\,, \tag{2.15}$$

where h_t indicates the hidden state at time step t, representing the network's memory. The term x_t refers to the input feature at this time step, and θ corresponds to the trainable parameters. Training for these networks uses an extension of the backpropagation algorithm known as backward propagation through time (BPTT)[45]. During BPTT, the network is unrolled through time (Figure 2.4), allowing for the calculation of gradients of the loss function with respect to each parameter at each time step. This unrolling process can be visualized as generating a sequence of FNNs, where each network represents the state of the RNNs at a specific time step[46]. This can be mathematically summarized as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial \theta}\,. \tag{2.16}$$

To elucidate the functionality of the BPTT, the unfolded schematic of RNN is considered (Figure 2.4), where the forward propagation at each time step can be expressed as

$$a_t = \mathrm{W}_x \cdot x_t + \mathrm{W}_h \cdot h_{t-1} + b\,, \tag{2.17}$$

where W_x is the weight matrix for the input x_t, W_h is the recurrent weight matrix applied to the previous hidden state h_{t-1} and b is the bias term. The hidden state h_t is updated recursively through an activation function, such as tanh function[10]

$$h_t = \tanh(a_t)\,. \tag{2.18}$$

The output of time step t, calculated through the hidden-to-output weight matrix W_o, and the output bias c, is given by

$$\hat{y}_t = W_o \cdot h_t + c\,. \tag{2.19}$$

Assuming the MSE (Eq. 2.8) as the loss function, the derivative of this function with respect to the output for the time step t is

$$\delta_t = \frac{\partial \mathcal{L}}{\partial \hat{y}_t} = \hat{y}_t - y_t\,. \tag{2.20}$$

The gradient of the loss in relation to the hidden states results from the direct impact of hidden states at time t on the output. Additionally, given the recurrent nature of RNNs, the prediction error at the subsequent time step $t+1$ will also influence the gradients at the current time step. Utilizing the chain rule, these two error terms can be written as follows:

$$\frac{\partial \mathcal{L}}{\partial h_t} = \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} = \delta_t \cdot W_o\,, \tag{2.21}$$

$$\frac{\partial \mathcal{L}}{\partial h_{t_{\text{from } h_{t+1}}}} = \frac{\partial \mathcal{L}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial a_{t+1}} \cdot \frac{\partial a_{t+1}}{\partial h_t} = \nabla_{h_{t+1}} \mathcal{L} \cdot (1 - h_{t+1}^2) \cdot W_h\,. \tag{2.22}$$

By combining these two equations, the overall gradient of the loss with respect to the hidden states is expressed as

$$\nabla_{h_t} \mathcal{L} = \delta_t \cdot W_o + \nabla_{h_{t+1}} \mathcal{L} \cdot (1 - (1 - h_{t+1}^2) \cdot W_h\,. \tag{2.23}$$

The trainable parameters in the network include the weights and bias terms of the output and hidden layers. The gradient of loss relative to the output bias parameter, denoted as $\nabla_c \mathcal{L}$, is given by

$$\nabla_c \mathcal{L} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} = \sum_{t=1}^{T} \delta_t \,. \tag{2.24}$$

This gradient with respect to the output weights, influenced by the respective hidden state, is expressed as

$$\nabla_{W_o} \mathcal{L} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot h_t = \sum_{t=1}^{T} \delta_t \cdot h_t \,. \tag{2.25}$$

The gradients relative to W_h and b, which are essential for capturing the temporal dependencies, are calculated as follows

$$\nabla_b \mathcal{L} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial a_t} = \sum_{t=1}^{T} \delta_t W_o \odot (1 - h_t^2) \,, \tag{2.26}$$

$$\nabla_{W_h} \mathcal{L} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial a_t} \cdot h_{t-1} = \sum_{t=1}^{T} \delta_t \cdot W_o \odot (1 - h_t^2) \cdot h_{t-1} \,. \tag{2.27}$$

Finally, the gradient with respect to W_x, which accounts for the influence of the input parameters, is given by

$$\nabla_{W_x} \mathcal{L} = \sum_{t=1}^{T} \delta_t \cdot W_o \odot (1 - h_t^2) \cdot x_t \,. \tag{2.28}$$

One major challenge with BPTT is the difficulty in capturing long-term dependencies (Eq.2.23) due to the repeated multiplication of gradients during backpropagation. This can lead to gradients that either vanish or explode, causing instability during training[11].

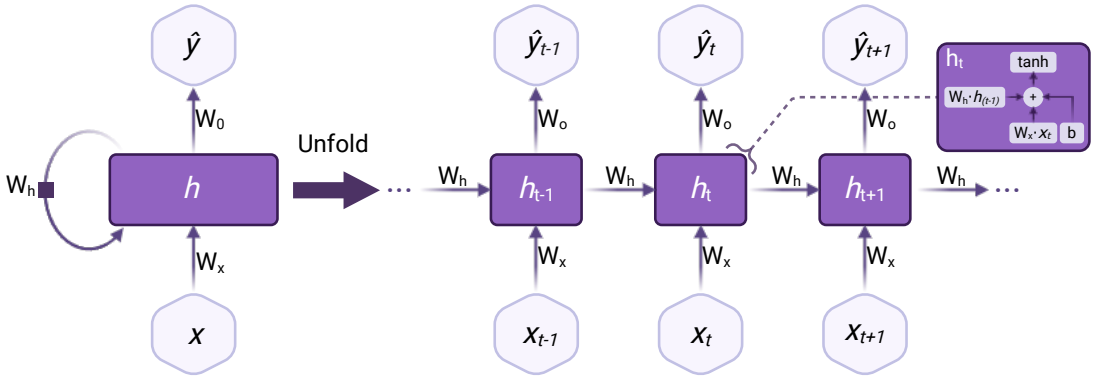

Figure 2.4 Schematic representation of a RNNs architecture.The left diagram illustrates the cyclic architecture, where at each time step t, the hidden units (h_t) are recurrently updated based on the present input x_t and the previous states. On the right-hand side, an unfolded computational graph of the network over multiple time steps is depicted. Here, the hidden state at time t is calculated by the weighted sum of the x_t and h_{t-1}, which is then passed through a tanh activation function. The final output $\hat{y}_t$ is obtained by transforming the hidden state using the weight matrix W_o.

Regularization

To reduce gradient instability during training of RNNs, several regularization techniques[47] have been explored, with skip connections representing a prominent example. These techniques trace their origins to "delay connections" in FNNs introduced by Lang and Hinton[48] in 1988, which inspired the development of residual connections for recurrent networks[49]. These designs provide more stable training by creating a direct pathway for gradients to bypass specific layers, preventing the cumulated multiplication of gradients and preserving their magnitude during backpropagation. These mechanisms incorporate identity mappings that pass the input of one layer directly to the output of a subsequent layer. As illustrated in Figure 2.5, this can be mathematically represented as

$$h_1 = x + f_1[x, \theta_1]\,, \tag{2.29}$$

$$h_2 = h_1 + f_2[h_1, \theta_2]\,, \tag{2.30}$$

$$y = h_2 + f_3[h_2, \theta_3]\,. \tag{2.31}$$

Teacher forcing can be applied as another technique to maintain the integrity of

Figure 2.5 A pictorial representation of a residual connection architecture. Here, the input x is sequentially processed through a series of functions $f_1[x, \theta_1]$, $f_2[h_1, \theta_2]$, and $f_3[h_2, \theta_3]$. At each step, the input is added to the output of each function via a skip connection, which is depicted by additive operations. The final output y is obtained after the last operation.

gradient flow in RNNs[50]. During standard training, the RNNs model generates predictions $\hat{y}_t$, which are subsequently fed as inputs for $t+1$. This recursive process can lead to error accumulation, where initial prediction errors propagate through the network, leading to instability. Teacher forcing guarantees that the model learns the correct sequence dependencies from the true data distribution rather than relying on potentially erroneous predictions from its earlier stages[10]. In practice, this involves feeding the actual target outputs from the training dataset to the network during training instead of its own forecasts. Mathematically, this process is represented as

$$\hat{y}_t = f(y_{t-1}, h_{t-1})\,, \tag{2.32}$$

where the true output y_{t-1} replaces the predicted output $\hat{y}_{t-1}$. However, during inference, the model must rely on its own previous predictions, which can result in a decline in performance due to a condition known as exposure bias[51]. This occurs because the model has not been trained to correct its own errors. Scheduled sampling overcomes this by progressively transitioning the model from using true outputs to its own predictions throughout the training process, thus helping it adapt to generating its own sequence predictions[52]. Teacher forcing is schematized in Figure 2.6.

There are a variety of other methods to optimize the training of RNNs. Recurrent dropout, for instance, involves applying a dropout mask at each step to deactivate the same subset of neurons. The addition of L2 regularization

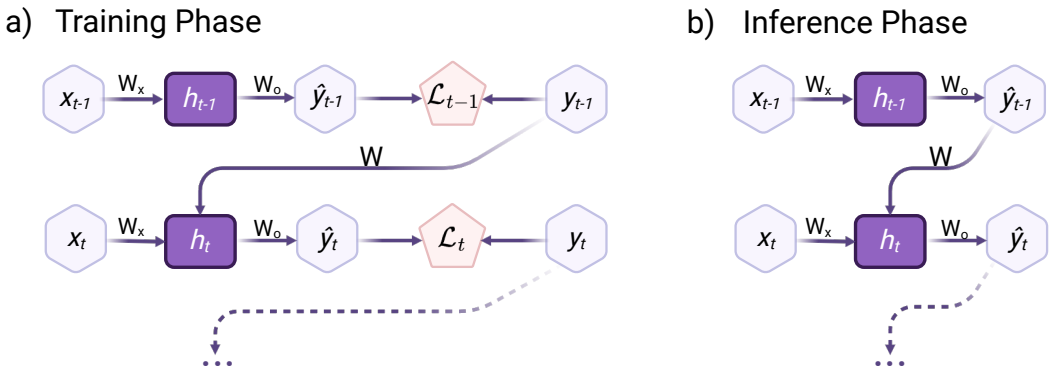

Figure 2.6 Illustration of the teacher forcing technique as a regularization technique for networks with recurrent connection. a) showcases the training phase, during which the model receives the ground truth target from the training dataset as input for the next time step. The hidden state h_{t-1} and the true output y_{t-1} are used to calculate the loss $\mathcal{L}_{t-1}$, which impact the weights W. b) During the inference phase, the model utilizes its own previous predictions $\hat{y}_{t-1}$ as input for the next time step to generate the subsequent predictions $\hat{y}_t$.

introduces a penalty term to the loss function[14], which promotes simpler models and further minimizes the occurrence of overfitting[31]. Gradient clipping constrains the gradients within a specified range, preventing the exploding gradient problem[53]. Regularization of temporal activation penalizes significant changes in activations between consecutive time steps, ensuring smoother transitions and improved model stability[54]. Each of these regularization techniques tackles shortcomings associated with training RNNs; however, difficulties persist in modeling long-term dependencies[28].

2.2.2. Long-Short Term Memory

In 1997, Hochreiter and Schmidhuber[1] introduced the LSTM networks for the enhancements of gradient flow. LSTM networks extend the capabilities of conventional RNNs by incorporating a distinctive architectural configuration that enables the maintenance of information over prolonged periods. This is achieved through a memory cell known as the cell state c_t, which is responsible for the network's long-term memory. The hidden state h_t functions as short-term memory, providing outputs and predictions at each time step. The LSTM architecture consists of two recurrent states that regulate the flow of

information, its addition and removal, through a set of gates, including the forget gate, the input gate, and the output gate.

The forget gate, f_t, is responsible for the selective determination of which portions of the previous cell state c_{t-1} should be discarded from the long-term memory. This decision is made by processing the present input features x_t and the previous hidden state h_{t-1} through a sigmoid function σ, which assesses each element in this gate. Irrelevant information is assigned a value of 0, while essential information is preserved with a value of 1, preventing the network from overloading with unnecessary data. The mathematical expression of this process is as follows:

$$f_t = \sigma(\mathrm{U}_f \cdot x_t + \mathrm{W}_f \cdot h_{t-1} + b_f)\,, \tag{2.33}$$

where W_f and U_f represent the weight matrices for the hidden state and input, respectively, whereas b_f refers to the bias.

The input gate regulates which values from the current input and the previous short-term memory are used to update the cell state. This gate has two components; the input gate layer, which calculates the values to update by using the weight matrices W_i and U_i, and bias b_i, processed through a sigmoid function

$$i_t = \sigma(\mathrm{U}_i \cdot x_t + \mathrm{W}_i \cdot h_{t-1} + b_i)\,, \tag{2.34}$$

and the candidate value layer, which generates new candidate values to be added to the cell state, represented as

$$\tilde{C}_t = \tanh(\mathrm{U}_c \cdot x_t + \mathrm{W}_c \cdot h_{t-1} + b_c)\,. \tag{2.35}$$

The current cell state is updated by combining the previous cell state, which has been scaled by the forget gate, and the new candidate cell state, scaled by

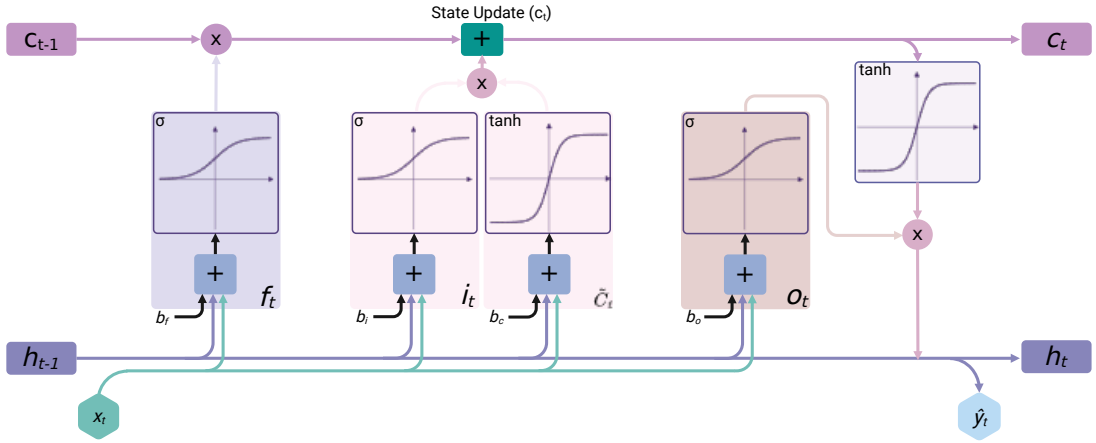

Figure 2.7 Schematic representation of LSTM cell architecture illustrating the internal operation across three gates of the forget f_t, input i_t, and output o_t. At each time step t, the cell state c_{t-1} and the hidden state h_{t-1} are updated based on the current input x_t. The f_t controls the amount of information that needs to be remembered from the c_{t-1} using a sigmoid activation function. The i_t regulates the update of the cell state with the new candidate state $\tilde{C}_t$, calculated by applying a tanh activation function. The cell state c_t is updated through the summation of the scaled previous state and the new candidate state. The o_t determines the next hidden state h_t, which is used to calculate the final output $\hat{y}_t$.

the input gate:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{C}_t. \tag{2.36}$$

Lastly, the output gate, o_t, controls the proportion of the updated cell state to be considered for the hidden output state[1]. By utilizing the previous hidden state and the current input data, the output gate calculates the short-term memory[50] to be remembered as

$$o_t = \sigma(\mathrm{U}_o \cdot x_t + \mathrm{W}_o \cdot h_{t-1} + b_o), \tag{2.37}$$

$$h_t = o_t \cdot \tanh(c_t). \tag{2.38}$$

A detailed illustration of this network, along with its gates, is depicted in Figure 2.7.

2.2.3. Advanced LSTM architectures

Over the years, researchers have introduced various advanced architectures to further enhance the ability of traditional LSTM networks to learn complex patterns and dependencies that require granular temporal distinctions[16].

Peephole LSTM

One such advancement is the Peephole LSTM, introduced by Gers et al.[55] in 2002. This network integrates peephole connections into the original LSTM architecture, which improves the network's ability to regulate information flow and learn precise timing intervals by efficiently preserving internal states. Peephole connections achieve this by providing direct connections from the cell state to the gates within the same memory block (Figure 2.8a). Such a configuration, known as the peephole mechanism, allows the gates to access the cell state even when the output gate is closed[56]. This mechanism provides finer control over the memory and gating processes by capturing the most relevant data at any given time, reducing the risk of retaining obsolete information[55].

Mathematically, the inclusion of peephole connections modifies the gate activation functions by introducing weight matrices V_f, V_i, and V_o for these connections to the forget, input, and output gates, respectively. The updated equations are expressed as

$$f_t = \sigma(U_f \cdot x_t + W_f \cdot h_{t-1} + V_f \cdot c_{t-1} + b_f), \tag{2.39}$$

$$i_t = \sigma(U_i \cdot x_t + W_i \cdot h_{t-1} + V_i \cdot c_{t-1} + b_i), \tag{2.40}$$

$$o_t = \sigma(U_o \cdot x_t + W_o \cdot h_{t-1} + V_o \cdot c_t + b_o), \tag{2.41}$$

Bidirectional LSTM

Bidirectional Recurrent Neural Networks (BiRNNs) were proposed by Schuster and Paliwal[57] in 1997, presenting an advanced neural network architecture that processes data from both past and future states. This bidirectional mechanism unravels a deeper relationship of contextual information within input sequences and improves the modeling of the underlying temporal dynamics, which is beneficial for sequential data[58]. The introduction of LSTM models further advanced this concept, leading to Graves' development[59, 60], of BiLSTM networks in 2005.

A BiLSTM network comprises two LSTM layers operating concurrently, one in the forward direction and the other in the backward direction. At each time step, the outputs of these two layers are concatenated to form the network's final output[61] (Figure 2.8b). Mathematically, the forward propagation for the hidden state at time t in the original sequence order is expressed as

$$\overrightarrow{h}_t = \text{LSTM}(x_t, \overrightarrow{h}_{t-1}) \tag{2.42}$$

while the backward propagation for the hidden state at time t in the reverse order is given by

$$\overleftarrow{h}_t = \text{LSTM}(x_t, \overleftarrow{h}_{t+1})\,. \tag{2.43}$$

The integration of information from both directions results in the concatenated hidden state[62], which can be represented as

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]\,. \tag{2.44}$$

Numerous architectures have been developed that utilize LSTM networks at their core and subsequently expand them to create more complex structures. Among these, the stacked LSTM architectures proposed by Graves et al.[61] in 2013, and Sutskever et al.[63] in 2014 led to the creation of deeper hierarchical structures. In 2015, Kalchbrenner et al.[64] from Google DeepMind proposed the Grid LSTM, which extended the capabilities of these original networks across

multiple dimensions, including spatio-temporal ones. This innovation enhanced the network's performance in sequence memorization tasks, enabling more effective handling of multidimensional data. In the same year, Shi et al.[65] introduced Convolutional LSTM (ConvLSTM), which replaced the fully connected layers in traditional LSTM with convolutional operations. This development enabled the model to capture both temporal and local spatial correlations in the data, which is advantageous for tasks where spatial patterns change over time. Further advancements include the Residual LSTM[66], inspired by the residual networks (ResNets)[49] in Convolutional Neural Network (CNN) architectures. The implementation of shortcut paths that generate direct connections between layers has been demonstrated to improve training stability in this architectural configuration, leading to a more robust learning process.

Extended Long Short-Term Memory

The most recent xLSTM architecture represents a step forward in the development of recurrent networks. This architectural innovation further enhances the scalability and performance of traditional LSTM models through the introduction of exponential gating mechanisms and matrix memory structures, termed scalar LSTM (sLSTM) and matrix LSTM (mLSTM)[67]. In the sLSTM model, the conventional sigmoid activation functions for the forget and input gates are replaced with exponential functions. The mathematical formulation for the gates is given by

$$f_t = \exp(\mathrm{U}_f \cdot x_t + \mathrm{W}_f \cdot h_{t-1} + b_f)\,, \tag{2.45}$$

$$i_t = \exp(\mathrm{U}_i \cdot x_t + \mathrm{W}_i \cdot h_{t-1} + b_i)\,, \tag{2.46}$$

To maintain numerical stability and accommodate a more extensive range of values, these gates are stabilized (f'_t and i'_t), using a stabilizer state[68], denoted as m_t:

$$m_t = \max(\log(f_t) + m_{t-1}, \log(i_t))\,, \tag{2.47}$$

$$f'_t = \exp(\log(f_t) + m_{t-1} - m_t)\,, \tag{2.48}$$

$$i'_t = \exp(\log(i_t) - m_t)\,. \tag{2.49}$$

The cell state update is performed as follows:

$$c_t = f't \cdot c_{t-1} + i'_t \cdot \tilde{C}_t\,. \tag{2.50}$$

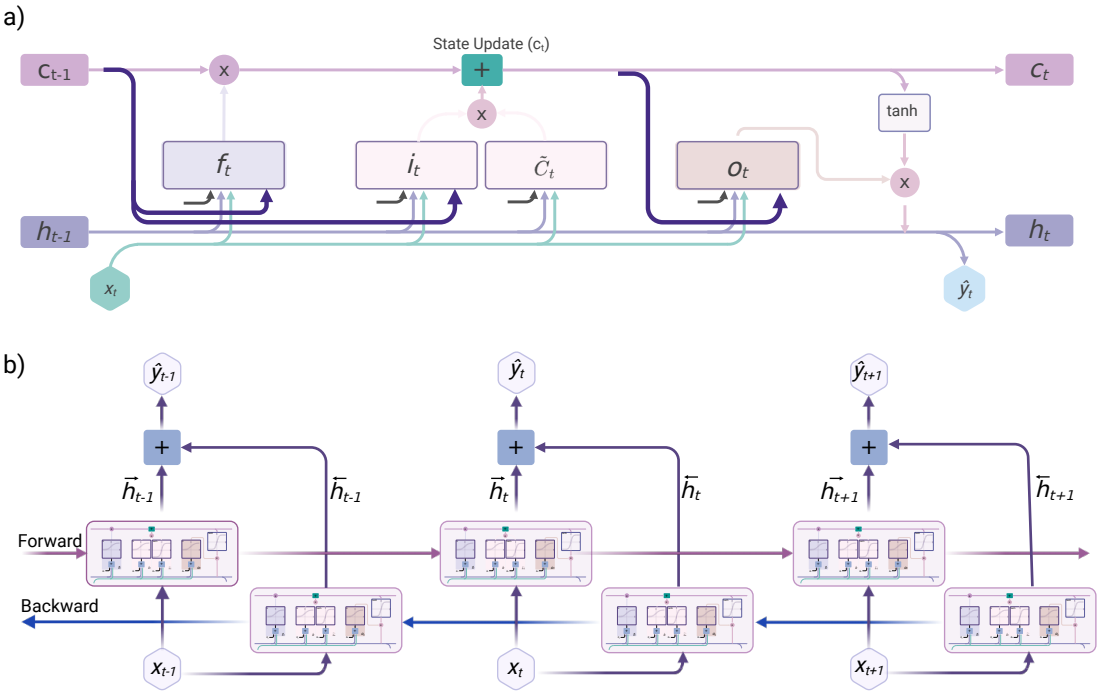

Figure 2.8 Pictorial representation of two advanced architectures of LSTM. a) Peephole LSTM extends the original LSTM architecture by incorporating peephole connections, which allow direct access of c_{t-1} to f_t and i_t and a direct connection of c_t to o_t, enabling more granular control over information flow. These connections are shown in bold purple arrows. b) Bidirectional LSTM consists of two LSTM layers that operate in parallel, one from forward (left to right) and the other one from backward (left to right) direction which reverse the sequence in reverse.

By applying an additional normalizer state (n_t), the hidden state is then calculated as

$$n_t = f'_t \cdot n_{t-1} + i'_t \,, \tag{2.51}$$

$$h_t = o_t \cdot \frac{c_t}{n_t} \,. \tag{2.52}$$

Additionally, the sLSTM incorporates memory mixing through recurrent connections, which enables interactions between multiple memory cells and promotes the generation of a more integrated memory representation[67].

The mLSTM model extends the traditional cell state to a matrix form, facilitating parallelization and efficient memory management. The hidden state updates in mLSTM are defined as

$$c_t = f_t \cdot c_{t-1} + i_t \cdot v_t k_t^T \,, \tag{2.53}$$

$$n_t = f_t \cdot n_{t-1} + i_t \cdot k_t \,, \tag{2.54}$$

$$h_t = o_t \cdot \frac{c_t \cdot q_t}{\max(|n_t^T q_t|, 1)} \,, \tag{2.55}$$

where v_t, k_t, and q_t are the value, key, and query vectors, respectively, which are of great significance in optimizing information storage and retrieval through a covariance update rule. The integration of sLSTM and mLSTM within residual block structures[49, 69] results in a unified and scalable architecture, named as xLSTM.

Sequence-to-Sequence Architectures

Another state-of-the-art LSTM-based design is the Seq-to-Seq architecture initially proposed by Sutskever et al.[63] in 2014 for machine translation. This model

maps an input sequence to an output sequence through two primary components, namely the encoder and the decoder[70]. These architectures are highly adaptable and can manage multivariate time series data[71]. Utilizing LSTM units, the encoder processes the input sequence consecutively and updates its hidden state h_t. The encoder's final hidden state condenses the content of the entire input sequence into a single condensed representation of context vector C:

$$\mathrm{C} = h_t = \mathrm{LSTM}(x, h_{t-1})\,. \tag{2.56}$$

The decoder, which can include various types of LSTM networks, takes the context vector along with the previous output, represented by y_{t-1}, as inputs to generate the decoder hidden states s_t and the output sequences for time step t which is

$$y_t = f(Ws_t + b)\,, \tag{2.57}$$

where

$$s_t = \mathrm{LSTM}(y_{t-1}, s_{t-1}, C)\,. \tag{2.58}$$

One drawback of this architectural approach is its reliance on a fixed-length context vector, which may prove to be inadequate for comprehensively capturing details from longer sequences, potentially leading to information loss. In response to this shortcoming, the attention mechanism was developed. First introduced by Bahdanau et al.[72], it enables learning the relative importance of different elements of the context vector sequence and their alignment with the output sequence, thus strengthening the precision of the model in handling long sequences[73, 74].

2.3. Attention mechanism

The attention mechanism has been designed to replicate the characteristics of human attention, originating from the psychological observation that human perception tends to focus on specific features of a scene rather than processing it in its entirety[75]. The initial implementation, known as additive attention, calculates a weighted sum of the encoder states, using an alignment score $e_{t,i}$ that incorporates the decoder state from the previous time step s_{t-1}, the encoder hidden states h_i and passes them from a feedforward neural network with one hidden layer[72]:

$$e_{t,i} = v^T \tanh(\mathrm{W}_a s_{t-1} + \mathrm{U}_a h_i)\,, \tag{2.59}$$

where W_a and U_a are the weight matrices that convert the respective states into a unified feature space, and v represents the learnable weights, projecting the activated sum into a scalar score. The $e_{t,i}$ is later normalized using a softmax function. Mathematically, these attention weights are represented as

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{k=1}^{T_x} \exp(e_{t,k})}\,, \tag{2.60}$$

and the context vector of the Seq-to-Seq model that aggregates each encoder hidden state with its correspondent attention weight is then calculated as

$$c_t = \sum_{i=1}^{T_e} \alpha_{t,i} h_i\,, \tag{2.61}$$

where T_e is the length of the encoder sequence.

In 2017, Vaswani et al.[76] introduced a more generic variant of this mechanism, known as self-attention. This groundbreaking approach — the publication "Attention is all you need"[76] has more than 125 thousand citations at the time of writing — has had a transformative impact on the advancement of contemporary generative models. The self-attention mechanism enables each element of the input sequence to attend to every other element by calculating the attention weights for each pair of temporal positions in the sequence[74]. The attention

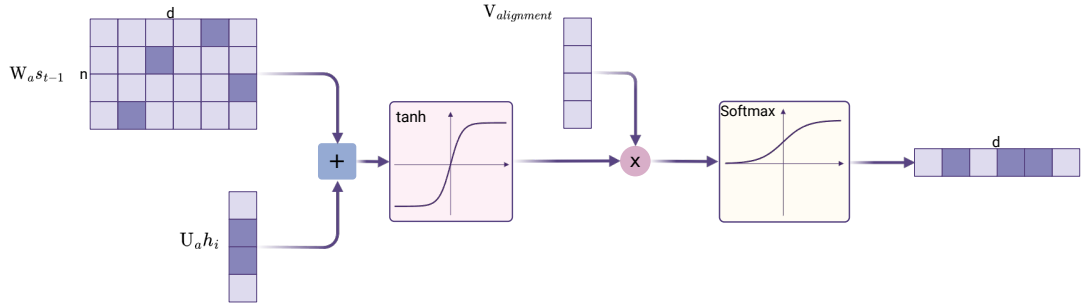

Figure 2.9 Attention mechanism, highlighting the use of tanh and softmax functions to transform the input matrix into attention weights for context vector computation. The $V_alignment$ highlights the focus on relevant input segments.

weights are derived as

$$\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{softmax}\left(\frac{\text{QK}^T}{\sqrt{d_\text{k}}}\right)\text{V}\,, \tag{2.62}$$

where Q, K, and V are query, key, and value matrices, respectively. The query represents the model's current state, and the key and value are derived from the same input sequence. This formulation uses a dot-product that maps the query to the key, is then scaled by the square root of the key vector dimensionality d_k to prevent a large magnitude of these scores, and is normalized through a softmax function to stabilize gradients during training. The attention scores are then multiplied by their corresponding values, which refer to the weighted values. With the introduction of the transformer architecture, the self-attention mechanism further extended to multi-head attention[76] (Figure 2.10), which enables the model to simultaneously consider various parts of the input sequence, described by:

$$\text{MultiHead}(\text{Q}, \text{K}, \text{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\text{W}^O\,, \tag{2.63}$$

where W^O refers to the output weight matrix, and each head is represented as

$$\text{head}_i = \text{Attention}(\text{QW}_i^Q, \text{KW}_i^K, \text{VW}_i^V)\,, \tag{2.64}$$

with W_i^Q, W_i^K, and W_i^V are the parameter matrices. This architecture enables

parallelization and allows the model to capture long-range relationships within the data through enriched representations[76].

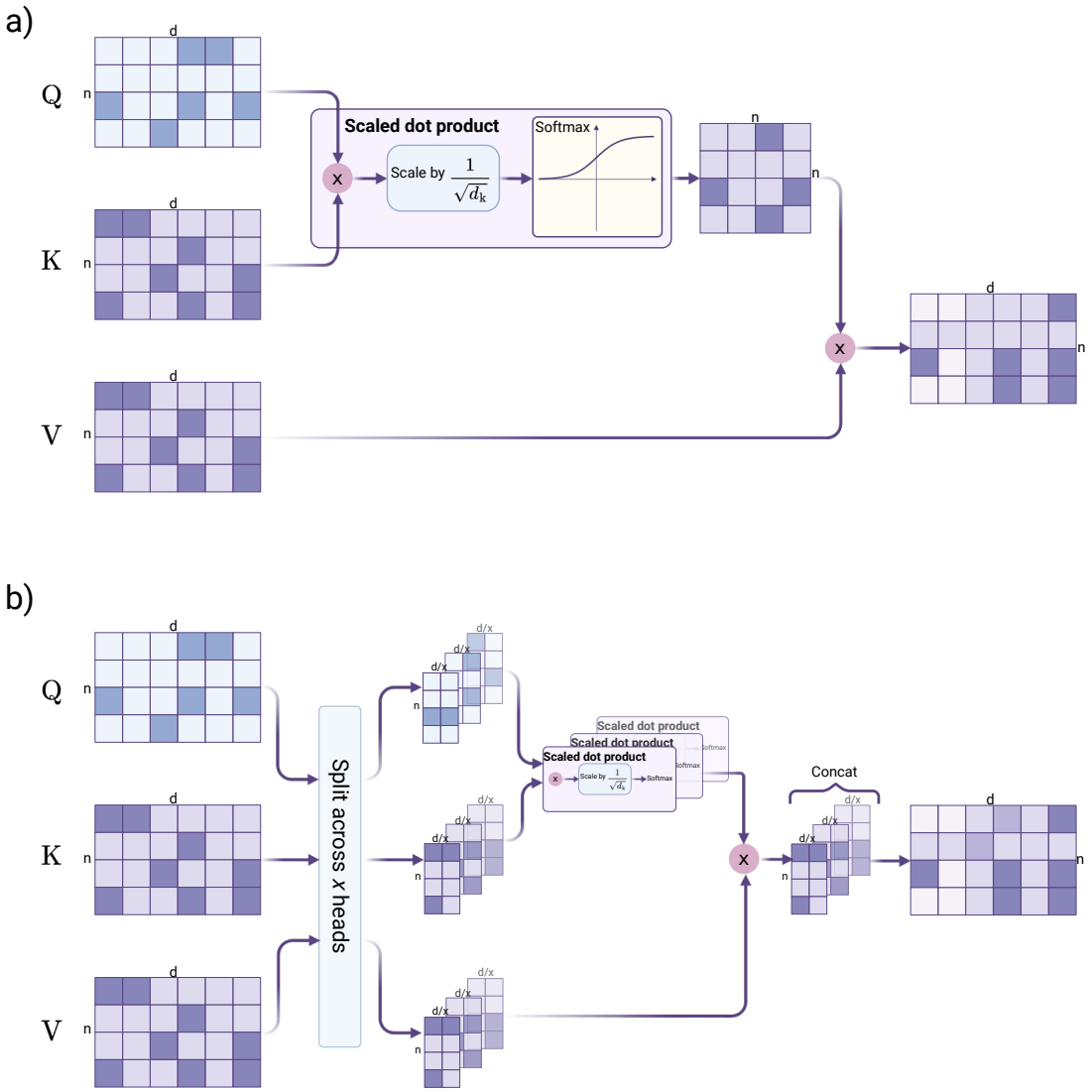

Figure 2.10 (a) Self-Attention involves a scaled dot product and softmax operations applied to the input matrix to generate attention scores, which are used to produce the context vector. (b) Multi-Headed Attention splits the input across multiple heads, each performing self-attention independently, and concatenates the results, allowing the model to capture diverse features and dependencies within the input sequence.

3. Data and Methodology

4. Results and Discussion

- the overview of the case study - data collection and processing explanation - implementation : uml, coding , structure , diagram - show the results - benchmarks and evaluation metrics

5. Conclusion and Outlook

-> from NxAI I got it: LSTM has stood the test of time and contributed to numerous Deep Learning success stories, in particular they constituted the first Large Language Models (LLMs). However, the advent of the Transformer technology with parallelizable self-attention at its core marked the dawn of a new era, outpacing LSTM at scale.

- explain what did we learn - challenge: scalability, interpretability, future trends.

A. Appendix*

* All Figures were created with BioRender.com

List of Figures

Abbreviations

Adam Adaptive Moment Estimation.

BiLSTM Bidirectional LSTM.

BiRNNs Bidirectional Recurrent Neural Networks.

BPTT backward propagation through time.

CNN Convolutional Neural Network.

ConvLSTM Convolutional LSTM.

DNNs Deep Neural Networks.

ELU Exponential Linear Unit.

FNNs Feedforward Neural Networks.

GPUs Graphics Processing Units.

LSTM long short term memory.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MLPs multilayer perceptrons.

mLSTM matrix LSTM.

MSE Mean Square Error.

NNs Neural Netowrks.

ReLU rectified linear unit.

ResNets residual networks.

RNNs Recurrent Neural Networks.

Seq-to-Seq sequence-to-sequence.

SGD Stochastic Gradient Descent.

sLSTM scalar LSTM.

tanh hyperbolic tangent.

xLSTM Extended Long Short-Term Memory.

Bibliography

[1] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 1530-888X. DOI: 10.1162/neco.1997.9.8.1735.

[2] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang. "Short-Term Residential Load Forecasting Based on LSTM Recurrent Neural Network". In: *IEEE Transactions on Smart Grid* 10.1 (2019), pp. 841–851. DOI: 10.1109/TSG.2017.2753802.

[3] K. Yokoo, K. Ishida, A. Ercan, T. Tu, T. Nagasato, M. Kiyama, and M. Amagasaki. "Capabilities of deep learning models on learning physical relationships: Case of rainfall-runoff modeling with LSTM". In: *Science of The Total Environment* 802 (Jan. 2022), p. 149876. ISSN: 0048-9697. DOI: 10.1016/j.scitotenv.2021.149876.

[4] Z. Li, H. Guo, A. V. Barenji, W. M. Wang, Y. Guan, and G. Q. Huang. "A sustainable production capability evaluation mechanism based on blockchain, LSTM, analytic hierarchy process for supply chain network". In: *International Journal of Production Research* 58.24 (2020), pp. 7399–7419. DOI: 10.1080/00207543.2020.1740342. eprint: https://doi.org/10.1080/00207543.2020.1740342. URL: https://doi.org/10.1080/00207543.2020.1740342.

[5] A. Diro and N. Chilamkurti. "Leveraging LSTM Networks for Attack Detection in Fog-to-Things Communications". In: *IEEE Communications Magazine* 56.9 (2018), pp. 124–130. DOI: 10.1109/MCOM.2018.1701270.

[6] W. Bank. "World Development Indicators: Urban population". In: (2023). URL: https://databank.worldbank.org/reports.aspx?dsid=2%5C&series=SP.URB.TOTL.

[7] T. Ahmad and D. Zhang. "A critical review of comparative global historical energy consumption and future demand: The story told so far". In: *Energy Reports* 6 (Nov. 2020), pp. 1973–1991. ISSN: 2352-4847. DOI: 10.1016/j.egyr.2020.07.020.

[8] G. Plessmann, M. Erdmann, M. Hlusiak, and C. Breyer. "Global Energy Storage Demand for a 100% Renewable Electricity Supply". In: *Energy Procedia* 46 (2014), pp. 22–31. ISSN: 1876-6102. DOI: 10.1016/j.egypro.2014.01.154.

[9] G. Zhang, X. Bai, and Y. Wang. "Short-time multi-energy load forecasting method based on CNN-Seq2Seq model with attention mechanism". In: *Machine Learning with Applications* 5 (Sept. 2021), p. 100064. ISSN: 2666-8270. DOI: 10.1016/j.mlwa.2021.100064.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. Ed. by A. Courville and Y. Bengio. Adaptive computation and machine learning. Includes bibliographical references and index. Cambridge, Massachusetts: The MIT Press, 2016. 1775 pp. ISBN: 9780262337373.

[11] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[12] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/bf02478259.

[13] F. Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961. DOI: 10.2307/1419730.

[14] J. Gareth, W. Daniela, H. Trevor, and T. Robert. *An introduction to statistical learning: with applications in R. With applications in R*. Ed. by D. Witten, T. Hastie, and R. Tibshirani. Second edition. Springer texts in statistics. Includes bibliographical references and index. New York: Springer, 2021. 1607 pp. ISBN: 1071614185.

[15] M. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, Sept. 2017. ISBN: 9780262343930. DOI: 10.7551/mitpress/11301.001.0001.

[16] S. J. D. Prince. *Understanding deep learning*. Includes bibliographical references and index. Cambridge, Massachusetts: The MIT Press, 2023. 1 p. ISBN: 9780262377102.

[17] M. Minsky and S. Papert. "An introduction to computational geometry". In: *Cambridge tiass., HIT* 479.480 (1969), p. 104. ISSN: 0 262 13043 2.

[18] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. DOI: 10.48550/arXiv.1811.03378.

[19] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/bf02551274.

[20] X. Glorot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323.

[21] K. Fukushima. "Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements". In: *IEEE Transactions on Systems Science and Cybernetics* 5.4 (1969), pp. 322–333. DOI: 10.1109/TSSC.1969.300225.

[22] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th International Conference on Computer Vision* (IEEE International Conference on Computer Vision). IEEE. IEEE, Sept. 2009, pp. 2146–2153. DOI: 10.1109/iccv.2009.5459469.

[23] V. Nair and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077. DOI: 10.5555/3104322.3104425.

[24] A. L. Maas, P. Qi, Z. Xie, A. Y. Hannun, C. T. Lengerich, D. Jurafsky, and A. Y. Ng. "Building DNN acoustic models for large vocabulary speech recognition". In: *Computer Speech & Language* 41.1 (Jan. 2017), pp. 195–213. ISSN: 0885-2308. DOI: 10.1016/j.csl.2016.06.007.

[25] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. DOI: 10.48550/arXiv.1511.07289.

[26] T. J. Sejnowski. *The deep learning revolution*. Cambridge, Massachusetts: The MIT Press, 2018. 1342 pp. ISBN: 9780262346825.

[27] R. Dechter. "Learning while searching in constraint-satisfaction-problems". In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. AAAI'86. Philadelphia, Pennsylvania: AAAI Press, 1986, pp. 178–183.

[28] I. Goodfellow. *Deep learning*. Ed. by A. Courville and Y. Bengio. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. 1775 pp. ISBN: 9780262337373.

[29] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

[30] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. ISSN: 1530-888X. DOI: 10.1162/neco.2006.18.7.1527.

[31] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York, 2009. ISBN: 9780387848587. DOI: 10.1007/978-0-387-84858-7.

[32] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee. "On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression". In: *IEEE Signal Processing Letters* 27 (2020), pp. 1485–1489. DOI: 10.1109/LSP.2020.3016837.

[33] A. de Myttenaere, B. Golden, B. Le Grand, and F. Rossi. "Mean Absolute Percentage Error for regression models". In: *Neurocomputing* 192 (2016). Advances in artificial neural networks, machine learning and computational intelligence, pp. 38–48. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.12.114. URL: https://www.sciencedirect.com/science/article/pii/S0925231216003325.

[34] R. Koenker and K. F. Hallock. "Quantile Regression". In: *Journal of Economic Perspectives* 15.4 (Nov. 2001), pp. 143–156. ISSN: 0895-3309. DOI: 10.1257/jep.15.4.143.

[35] R. Koenker. *Quantile Regression*. Vol. 38. Cambridge University Press, 2005. ISBN: 9780511754098. DOI: 10.1017/cbo9780511754098.

[36] F. Rodrigues and F. C. Pereira. "Beyond Expectation: Deep Joint Mean and Quantile Regression for Spatiotemporal Problems". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.12 (Dec. 2020), pp. 5377–5389. ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2966745.

[37] O. V. Johnson, C. XinYing, O. E. Johnson, K. W. Khaw, and M. H. Lee. "Learning Rate Schedules and Optimizers, A Game Changer for Deep Neural Networks". In: *Advances in Intelligent Computing Techniques and Applications*. Springer Nature Switzerland, 2024, pp. 327–340. ISBN: 9783031597114. DOI: 10.1007/978-3-031-59711-4_28.

[38] D. E. Rumelhart and J. L. McClelland. *Learning Internal Representations by Error Propagation*. MIT Press, 1987, pp. 318–362.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[40] A. Cauchy et al. "Méthode générale pour la résolution des systemes d'équations simultanées". In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[41] H. Robbins and S. Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (Sept. 1951), pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729586.

[42] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[43] G. Hinton, N. Srivastava, and K. Swersky. *Neural networks for machine learning*. Tech. rep. 8. Lecture 6a Overviewof mini-batchgradient descent, 2012, p. 2.

[44] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: https://api.semanticscholar.org/CorpusID:6628106.

[45] P. J. Werbos. "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural Networks* 1.4 (Jan. 1988), pp. 339–356. ISSN: 0893-6080. DOI: 10.1016/0893-6080(88)90007-X.

[46] R. J. Williams and J. Peng. "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories". In: *Neural Computation* 2.4 (Dec. 1990), pp. 490–501. ISSN: 1530-888X. DOI: 10.1162/neco.1990.2.4.490.

[47] J. Kukačka, V. Golkov, and D. Cremers. *Regularization for Deep Learning: A Taxonomy*. 2017. DOI: 10.48550/arXiv.1710.10686.

[48] K. Lang. "The development of the time-delay neural network architecture for speech recognition". In: *Technical Report CMU-CS-88-152* (1988).

[49] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016, pp. 770–778. DOI: 10.1109/cvpr.2016.90.

[50] R. DiPietro and G. D. Hager. "Deep learning: RNNs and LSTM". In: *Handbook of Medical Image Computing and Computer Assisted Intervention*. Elsevier, 2020, pp. 503–519. ISBN: 9780128161760. DOI: 10.1016/b978-0-12-816176-0.00026-0.

[51] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. *Sequence Level Training with Recurrent Neural Networks*. 2015. DOI: 10.48550/arXiv.1511.06732.

[52] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. *Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks*. 2015. DOI: 10.48550/arXiv.1506.03099.

[53] T. Mikolov. "Statistical language models based on neural networks". PhD thesis. Brno University of Technology, Faculty of Information Technology, Oct. 2012.

[54] S. Merity, B. McCann, and R. Socher. *Revisiting Activation Regularization for Language RNNs*. 2017. DOI: 10.48550/arXiv.1708.01009.

[55] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. "Learning precise timing with lstm recurrent networks". In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 115–143. ISSN: 1532-4435. DOI: 10.1162/153244303768966139. URL: https://doi.org/10.1162/153244303768966139.

[56] F. Landi, L. Baraldi, M. Cornia, and R. Cucchiara. "Working Memory Connections for LSTM". In: *Neural Networks* 144 (Dec. 2021), pp. 334–341. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2021.08.030.

[57] M. Schuster and K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. ISSN: 1053-587X. DOI: 10.1109/78.650093.

[58] S. Siami-Namini, N. Tavakoli, and A. S. Namin. "The Performance of LSTM and BiLSTM in Forecasting Time Series". In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. IEEE, Dec. 2019, pp. 3285–3292. DOI: 10.1109/bigdata47090.2019.9005997.

[59] A. Graves and J. Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 4 (2005), pp. 2047–2052. DOI: 10.1016/j.neunet.2005.06.042.

[60] A. Graves and A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg, 2012. ISBN: 9783642247972. DOI: 10.1007/978-3-642-24797-2.

[61] A. Graves, A.-r. Mohamed, and G. Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (IEEE International Conference on Acoustics, Speech, and Signal Processing). Ieee. IEEE, May 2013, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947. arXiv: 1303.5778.

[62] Y. Li, G. Du, Y. Xiang, S. Li, L. Ma, D. Shao, X. Wang, and H. Chen. "Towards Chinese clinical named entity recognition by dynamic embedding using domain-specific knowledge". In: *Journal of Biomedical Informatics* 106 (June 2020), p. 103435. ISSN: 1532-0464. DOI: 10.1016/j.jbi.2020.103435.

[63] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in neural information processing systems* 27 (2014). DOI: 10.48550/arXiv.1409.3215.

[64] N. Kalchbrenner, I. Danihelka, and A. Graves. *Grid Long Short-Term Memory*. 2015. DOI: 10.48550/arXiv.1507.01526.

[65] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. "Convolutional LSTM Network: a machine learning approach for precipitation nowcasting". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 802–810.

[66] J. Kim, M. El-Khamy, and J. Lee. *Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition*. 2017. DOI: 10.48550/arXiv.1701.03360.

[67] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. *xLSTM: Extended Long Short-Term Memory*. 2024. DOI: 10.48550/arXiv.2405.04517.

[68] M. Milakov and N. Gimelshein. *Online normalizer calculation for softmax*. 2018. DOI: 10.48550/arXiv.1805.02867.

[69] R. K. Srivastava, K. Greff, and J. Schmidhuber. *Training Very Deep Networks*. 2015. DOI: 10.48550/arXiv.1507.06228.

[70] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *arXiv preprint arXiv:1406.1078* (2014). DOI: 10.48550/arXiv.1406.1078.

[71] R. Pérez-Chacón, G. Asencio-Cortés, A. Troncoso, and F. Martínez-Álvarez. "Pattern sequence-based algorithm for multivariate big data time series forecasting: Application to electricity consumption". In: *Future Generation Computer Systems* 154 (May 2024), pp. 397–412. ISSN: 0167-739X. DOI: 10.1016/j.future.2023.12.021.

[72] D. Bahdanau, K. Cho, and Y. Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. Preprint at. 2014. DOI: 10.48550/arXiv.1409.0473. arXiv: 1409.0473.

[73] M.-T. Luong, H. Pham, and C. D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *arXiv preprint arXiv:1508.04025* (2015). DOI: 10.48550/arXiv.1508.04025.

[74] J. Cheng, L. Dong, and M. Lapata. "Long Short-Term Memory-Networks for Machine Reading". In: *arXiv preprint arXiv:1601.06733* (2016). DOI: 10.48550/arXiv.1601.06733.

[75] A. Graves, G. Wayne, and I. Danihelka. *Neural Turing Machines*. 2014. DOI: 10.48550/arXiv.1410.5401.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. 2017. DOI: 10.48550/arXiv.1706.03762.