

GBMs are a type of ensemble learning technique that combines multiple weak learners, typically decision trees, into a strong predictive model. In the context of time series forecasting, GBMs work by iteratively fitting weak learners (decision trees) to the residuals of the previous iteration, with the goal of minimizing the overall prediction error.

The GBM algorithm is initialized with a set of hyperparameters, such as the number of trees, the maximum depth of the trees, the learning rate, and the loss function (e.g., squared error for regression tasks). The algorithm then iteratively fits decision trees to the residuals of the previous iteration, with each new tree focusing on the instances that were poorly predicted by the previous ensemble.

The main advantages of using GBM for time series forecasting include:

- Ability to capture non-linear relationships and complex patterns in the data.
- Robustness to outliers and noise due to the ensemble nature of the algorithm.
- Automatic feature selection and handling of feature interactions through the decision tree structure.
- Regularization techniques, such as shrinkage and subsampling, help prevent overfitting.

However, GBMs can be prone to overfitting if not properly tuned, and they may struggle with long-range dependencies or highly non-stationary time series. Additionally, GBM models can be relatively complex and computationally expensive, especially for large datasets or deep trees.

In practice, GBMs also are often used in combination with other techniques, such as feature engineering, rolling-window validation, and ensemble methods, to improve their forecasting performance for time series data.

4.3. Deep Neural Networks (DNNs)

Recently, more powerful computing platforms and cloud services have appeared thanks to the development of hardware and software tools that enable the implementation of more complex forecasting algorithms – DNNs with dozens and hundreds of layers, the theoretical foundations of which were laid in the work [10].

DNNs have proven to be quite effective for the tasks of image recognition and segmentation, streaming video processing, machine translation and many other applications. During the last ten years, there

has been an increased attention of researchers in the application of DNNs to the problems of time series research.

The base concept behind Recurrent Neural Networks (RNNs) for time series forecasting is their ability to model sequential data and capture long-term dependencies within the data.

In traditional Feed Forward ANNs, the inputs and outputs are independent of each other, making them unsuitable for modelling sequential data like time series. RNNs, on the other hand, introduce a form of memory that allows them to process inputs sequentially, using information from previous time steps to inform the current output.

The key idea behind RNNs is the use of recurrent connections, which allow the network to maintain an internal state or memory that captures information from previous time steps (Fig. 4.4). This internal state is updated at each time step based on the current input and the previous state, creating a dynamic system that can model temporal dependencies.

At each moment of time t , the network receives not only the input values (vector $x(t)$), but also the values of the parameters characterizing the internal state of the network at the previous moment of time $h(t-1)$. This allows you to process a series of events over time.

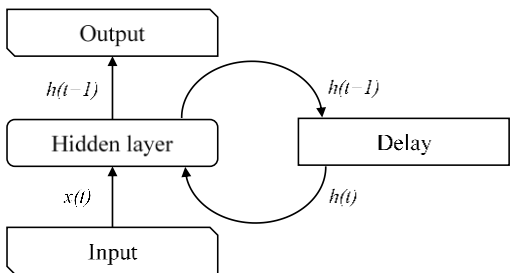


Figure 4.4. Simplified structure of a RNN

Source: designed by the authors based on [16]

So, the main advantage of RNNs over Feed Forward ANNs is their ability to use internal memory to process sequences of arbitrary length. Therefore, networks of this architecture are more suitable for processing sequences (text, time series data) than Feed Forward networks, in particular MLP.

Other common used architecture of DNNs is Convolutional Neural Networks (ConvNets, CNN), which is the main tool in processing with

data which has a spatial structure. Such networks are successfully used to extract spatial properties of graphic images, streaming video or audio.

Recurrent Neural Networks (RNNs)

Let's give a formalized description of the RNN using the apparatus of differential equations. Let $\vec{s}(t)$ be the value of the d -dimensional state signal vector. Then the general nonlinear inhomogeneous differential equation of the first order, which describes the evolution of the state signal as a function of time, t , can be written as:

$$\frac{d\vec{s}(t)}{dt} = \vec{f}(t) + \vec{\phi}, \quad (4.1)$$

where $\vec{f}(t)$ – d -dimensional function of time, $t \in R^+$, and $\vec{\phi}$ – constant d -dimensional vector [12].

The canonical form of $\vec{f}(t)$ is:

$$\vec{f}(t) = \vec{h}(\vec{s}(t), \vec{x}(t)), \quad (4.2)$$

where $\vec{x}(t)$ – is d -measurement vector of the input signal, a $\vec{h}(\vec{s}(t), \vec{x}(t))$ – vector-valued function of vector-valued arguments [17].

In certain cases, it is possible to consider s and x as «analog» quantities (that is, functions not only of time t , but also of another independent continuous variable ξ , which determines coordinates in multidimensional space). Using these notations, the input intensity displayed on a flat two-dimensional scale will be represented as $x(\vec{\xi}, t)$ $\xi \in R^2$.

Sampling $x(\vec{\xi}, t)$ on a uniform two-dimensional scale transforms this data set into the representation $x(\vec{l}, t)$, where $\vec{l}$ is a discrete two-dimensional index. Finally, adding the values $x(\vec{l}, t)$ or all permutations of the components of index $\vec{l}$ into the column vector gives $\vec{x}(t)$, as represented by Eq. (4.3)

$$\frac{d\vec{s}(t)}{dt} = \vec{h}(\vec{s}(t), \vec{x}(t)) + \vec{\phi}. \quad (4.3)$$

A special case of $\vec{f}(t)$ of Eq. (4.2) is:

$$\vec{f}(t) = \vec{a}(t) + \vec{b}(t) + \vec{c}(t), \quad (4.4)$$

The component equations (4.4) $\vec{a}(t), \vec{b}(t), \vec{c}(t)$, d -dimensional values vector of functions of time t . Equation (4.4) is called an «additive model» in the brain dynamics research literature because it adds terms,

possibly nonlinear, that determine the rate of change of neuronal activity or potentials $\vec{s}(t)$.

As a cornerstone of neural network research, the abstract form of the additive model in equation (4.4) has been elaborated in many ways, including the inclusion of delay effects, the imposition of «maneuvering» (or «saturating») system state limits, and other factors.

Biologically motivated uses of the additive model include computational vision analysis, decision-making, reinforcement learning, sensory-motor control, short- and long-term memory, and the study of temporal order in language [17]. It has also been pointed out that the additive model generalizes the Hopfield model [18], which, being rooted in biological plausibility, has had an impact on physics and engineering [17].

In fact, a simplified and discretized form of the additive model played a key role in relating the nonlinear dynamical systems governing morphogenesis, one of the fundamental aspects of developmental biology, to a generalized version of the Hopfield network and its application to an engineering problem in image processing.

Consider the saturating additive model in equation (4.4) with three components, $\vec{a}(t)$, $\vec{b}(t)$, $\vec{c}(t)$, which defined as follows:

$$\vec{a}(t) = \sum_{k=0}^{K_s-1} \vec{a}_k \left(\vec{s}(t - \tau_s(k)) \right) \quad (4.5)$$

$$\vec{b}(t) = \sum_{k=0}^{K_r-1} \vec{b}_k \left(\vec{r}(t - \tau_r(k)) \right) \quad (4.6)$$

$$\vec{c}(t) = \sum_{k=0}^{K_x-1} \vec{c}_k \left(\vec{x}(t - \tau_x(k)) \right) \quad (4.7)$$

$$\vec{r}(t - \tau_r(k)) = G \left(\vec{s}(t - \tau_s(k)) \right) \quad (4.8)$$

where $\vec{r}(t)$, the input signal vector, which is a warped version of the state signal vector, $\vec{s}(t)$. For an elementary non-linear, saturating and rotating «strain» (or «activation») function $G(z)$ is an optionally scaled and/or shifted hyperbolic tangent form.

Then the resulting system, obtained by substituting equations 4.5–4.8 into Eq. (4.4) and inserting into (4.1):

$$\begin{aligned} \frac{d\vec{s}(t)}{dt} = & \sum_{k=0}^{K_s-1} \vec{a}_k \left(\vec{s}(t - \tau_s(k)) \right) + \sum_{k=0}^{K_r-1} \vec{b}_k \left(\vec{r}(t - \tau_r(k)) \right) + \\ & + \sum_{k=0}^{K_x-1} \vec{c}_k \left(\vec{x}(t - \tau_x(k)) \right) \end{aligned} \quad (4.9)$$

Eq. 4.9 is a nonlinear ordinary delay differential equation with discrete delays. Delay is a common feature of many processes in biology, chemistry, mechanics, physics, ecology, and economics, among others, so that the nature of the processes dictates the use of delay equations as the only appropriate modeling tool.

The definition of the three weight matrices and the displacement vector transforms Eq. 4.9 and 4.8. into the canonical form of a recurrent neural network (RNN):

$$\vec{s}[n] = W_s \vec{s}[n-1] + W_r \vec{r}[n-1] + W_x \vec{x}[n] + \vec{\theta}_s \quad (4.10)$$

$$\vec{r}[n] = G(\vec{s}[n]) \quad (4.11)$$

A visualization of the principle of operation of the RNN from the Eq. 4.4 is shown in the Figure 4.5.

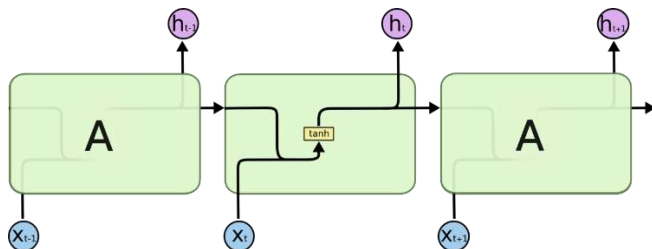


Figure 4.5. The recurrent layer in a standard RNN consists of one layer

Source: designed by the authors based on [19]

In the RNN guarding system, the cell reading signal is a twisted version of the cell signal itself. It is important that a copy of this distorted signal is transmitted back from one cycle to the next as part of the update signal to the coming stage.

This close connection between the signal read on one step and the signal on the next step immediately flows onto the gradient of the goal function of the signal. This influx will occur at the stage of training, which will end with a significant gradient.

Long-Short Term Memory (LSTM)

One of the different types of RNN architectures that are designed to accommodate hourly durations is a measure of «long short-term memory» (LSTM, from the English Long Short-Term Memory). Instead of traditional RNNs, LSTMs are more effective in forecasting

hourly series during periods of time when the important parts are separated by hourly lags from unidentified fluctuations or intervals. The LSTM measure was found to solve the problem of missing gradients.

The key insight of the LSTM model was the inclusion of nonlinear, latent data elements in the RNN package, which can be ensured in such a way that the gradient of the target function is consistent with the signal (a value that is directly proportional to the updated parameters is calculated per hour of training using the Gradient Descent help), without knowing [15].

The LSTM cell can be extracted from the canonical RNN package, as before (4.10) by introducing various additional elements. To avoid the problem of the existing gradient, a cell can be created by modifying the cell design. As the exit point, it is necessary to divide the right part of the level 4.10 (the update signal will become a mark on the index, n) into two parts:

$$\vec{s}[n] = \vec{F}_s(\vec{s}[n-1]) + \vec{F}_u(\vec{r}[n-1], \vec{x}[n]), \quad (4.12)$$

$$\vec{r}[n] = G_d(\vec{s}[n]). \quad (4.13)$$

Where $G_d(\vec{s}[n])$ is hyperbolic tangent.

$$\vec{F}_s(\vec{s}[n-1]) = W_s \vec{s}[n-1], \quad (4.14)$$

$$\vec{F}_u(\vec{r}[n-1], \vec{x}[n]) = W_r \vec{r}[n-1] + W_x \vec{x}[n] + \vec{\theta}_s. \quad (4.15)$$

The first part, $\vec{F}_s(\vec{s}[n-1])$, represents the input from the signal at the front edge. The other part, $\vec{F}_u(\vec{r}[n-1], \vec{x}[n])$, represents the update information that results from the combination of the signal read from the forward timeline and the input signal (external destructive force) on exact time (plus displacement vector $\vec{\theta}_s$).

In discrete systems, time has less historical significance. It would be more correct to use the term «smart» if we were to talk about the meaning of the signal at the stages. Here the terms «front», «stream» and «offensive» are used more for clarity.

RNN systems can be easily activated in a reversal direction, in which case all indices are reversed, and the values of «forward» and «advance» change. Improved performance was achieved through bidirectional processing in various applications.

As the input data is prepared in advance and supplied to the system as a whole, then the causality bias can be completely weakened.

This can be done in additions, where before the start of processing the entire set of initial data or a collection of independent segments of

initial data is collected. Gratuitous processing (a technique characterized by the use of historical input data) can be useful for identifying the presence of «context» among data expressions.

Corresponding to the same (4.11), the signal will be absorbed by the information source in the same proportions on the skin. These proportions can be adjusted by multiplying two values by special «shutter» signals $\vec{g}_{cs}[n]$ («control state») and $\vec{g}_{cu}[n]$ («updated control»), as follows:

$$\vec{s}[n] = \vec{g}_{cs}[n] \odot \vec{F}_s(\vec{s}[n-1]) + \vec{g}_{cu}[n] \odot \vec{F}_u(\vec{r}[n-1], \vec{x}[n]), \quad (4.16)$$

$$\vec{0} \leq \vec{g}_{cs}[n], \vec{g}_{cu}[n] \leq \vec{1}. \quad (4.17)$$

The elements of the gate signals are positive fractions. Abbreviated designation, $\vec{g}[n] \in [\vec{0}, \vec{1}]$, means that the value of all elements of the gate vector signal, $\sim \vec{g}[n]$, at the step with index n lie on a closed segment between 0 and 1.

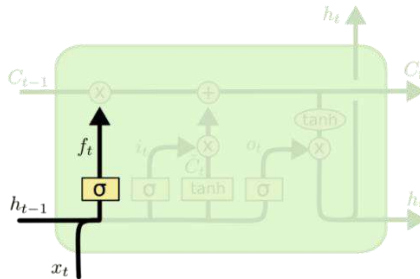


Figure 4.6. Adding a state control gate to control the magnitude of the state signal

Source: designed by the authors based on [19]

Gate signals $\vec{g}_{cs}[n]$ and $\vec{g}_{cu}[n]$ in the equations (4.16), (4.17) provide a mechanism for fine-grained control of two types of contributions to the state signal at each step. In particular $\vec{g}_{cs}[n]$ allows you to control the value of the state signal, which is retained from the previous step, and $\vec{g}_{cu}[n]$ controls the amount of update signal that should be injected into the state signal at the current step.

In Eq. (4.18) W_s is a diagonal matrix with positive shares $\frac{1}{|a_{ij}|}$ on the main diagonal. So, since the elements $\vec{g}_{cs}[n]$ are also fractions. Let's define:

$$W_s = I \quad (4.18)$$

$\vec{g}_{cs}[n] \odot W_s$ is acceptable if the gate functions can be parameterized and their parameters are learned during training. Under these conditions, equation (4.14) can be simplified to:

$$\vec{F}_s(\vec{s}[n-1]) = \vec{s}[n-1] \quad (4.19)$$

Thus, equation (4.15) can be reduced to the following form:

$$\vec{s}[n] = \vec{g}_{cs}[n] \odot \vec{s}[n-1] + \vec{g}_{cu}[n] \odot \vec{F}_u(\vec{r}[n-1], \vec{x}[n]) \quad (4.20)$$

The contribution from the state signal in the previous step remains fractional, ensuring the stability of the overall system. Schematically, the insertion of advanced control elements from equation (4.20) into the canonical RNN system of equation (4.1) is shown in Fig. 4.7.

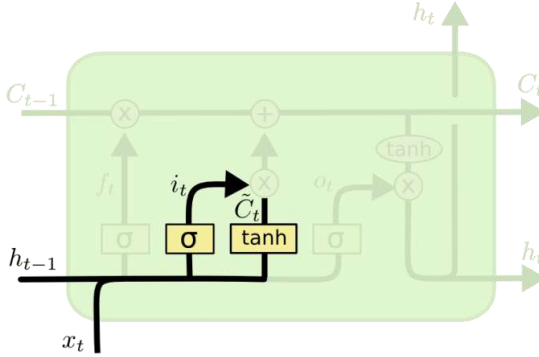


Figure 4.7. State control gate

Source: designed by the authors based on [19]

Although the update time $\vec{F}_u(\vec{r}[n-1], \vec{x}[n])$, is generally controlled by $\vec{g}_{cu}[n]$, the internal state of $\vec{F}_u(\vec{r}[n-1], \vec{x}[n])$ needs to investigate.

According to equation (4.20), the readout signal from the previous step and the input signal at the current step constitute the update candidate signal at each step with index n, and both terms contribute in equal proportion.

The problem is to always use $W_r \vec{r}[n-1]$ in its entirety, is that when $\vec{g}_{cu}[n] \rightarrow 1$, $\vec{s}[n-1]$ and $\vec{s}[n]$ are connected through W_r and the strain function. This limits the gradient of the objective function with respect to the state signal, thereby subjecting the system to the vanishing/exploding gradients problem.

To cancel this feedback path, the readout signal, $\vec{r}[n]$ distributed using another gate signal, $\vec{g}_{cr}[n]$ («input control»), as follows:

$$\vec{v}[n] = \vec{g}_{cr}[n] \odot \vec{r}[n] \quad (4.21)$$

$$\vec{0} \leq \vec{g}_{cr}[n] \leq \vec{1} \quad (4.22)$$

The control element $\vec{g}_{cr}[n]$, determines the partial magnitude of the readout signal, which becomes the signal of the observed value of the cell at the stage with the index n .

Thus, using $\vec{v}[n - 1]$ instead of $\vec{r}[n - 1]$ in Eq. (4.20) turns it into:

$$\vec{F}_u(\vec{v}[n - 1], \vec{x}[n]) = W_r \vec{v}[n - 1] + W_x \vec{x}[n] + \vec{\theta}_s \quad (4.23)$$

The schematic diagram, an extension to accommodate the control reading, is shown in equation (4.20), and the modified repetition ratio used in equation (4.21) is shown in Fig. 4.8.

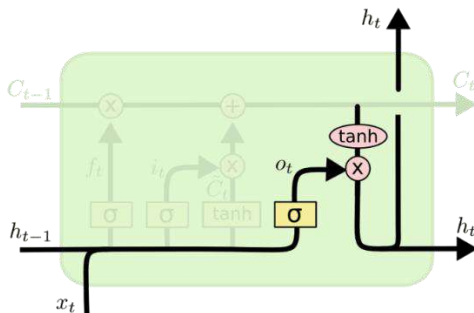


Figure 4.8. Gate «control reading»

Source: designed by the authors based on [19].

The external input does not affect the stability of the system and does not affect its tendency to vanishing / exploding gradients, the combination of the input data with its own «control input» input makes the system more flexible. By multiplying the external input signal, $\vec{x}[n]$, in equation 4.16 on the selected gate signal, $\vec{g}_{cx}[n]$, transforms equation (4.21) into:

$$\vec{F}_u(\vec{v}[n - 1], \vec{x}[n]) = W_r \vec{v}[n - 1] + \vec{g}_{cx}[n] \odot W_x \vec{x}[n] + \vec{\theta}_s \quad (4.24)$$

According to equation (4.21) and equation (4.23), using as a control read gate, $\vec{g}_{cr}[n]$, and the input control gate, $\vec{g}_{cx}[n]$, allows you to enter

the update period, $\vec{F}_u(\vec{v}[n-1], \vec{x}[n])$, to contain any combination of read signal and external input.

The dynamic range of the cell value $\vec{v}[n]$, is determined by the readout signal $\vec{r}[n]$, which is limited by the deformation nonlinearity $G_d(z)$. In order to maintain the same dynamic range while absorbing contributions from the input signal, $\vec{x}[n]$ (or $\vec{g}_{cx}[n] \odot \vec{x}[n]$, if the control input is part of the system architecture), the aggregate signal $\vec{F}_u(\vec{v}[n-1], \vec{x}[n])$, is hardened by a saturating nonlinearity deformation $G_d(z)$, in order to obtain a candidate signal for updating $\vec{u}[n]$:

$$\vec{u}[n] = G_d \left(\vec{F}_u(\vec{v}[n-1], \vec{x}[n]) \right). \quad (4.25)$$

Thus, replacing the update period in equation (4.19) with $\vec{u}[n]$, given by equation (4.25):

$$\vec{s}[n] = \vec{g}_{cs}[n] \odot \vec{s}[n-1] + \vec{g}_{cu}[n] \odot \vec{u}[n]. \quad (4.26)$$

Equation (4.26) is the main component of the set of formulas defining the cell of the LSTM network. According to Equation (4.26), the cell state signal at the current stage is a weighted combination of the cell state signal at the previous stage and the aggregation of historical and new update information available at this stage.

The complete data path of an LSTM cell, completed by strengthening the canonical RNN system with controls and signal suppression, is illustrated in Fig. 4.9.

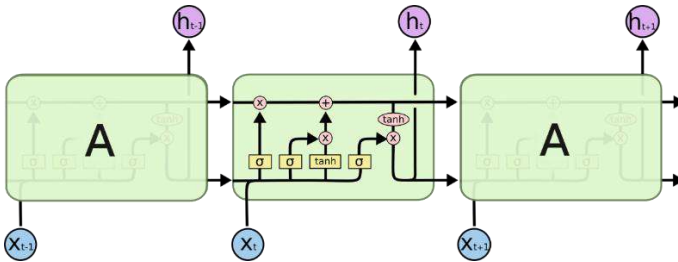


Figure 4.9. The recurrent LSTM network model consists of four interacting layers

Source: designed by the authors based on [19]

Similar to a standard RNN, an LSTM network is trained using Gradient Descent. LSTM can also learn short-range dependencies among subsequence samples comprising the training data.

Gated Recurrent Unit (GRU)

A more modern version of the DNN architecture is the Gated Recurrent Unit Neural Networks (GRU) model, which is a modification of the LSTM model (Fig. 4.10).

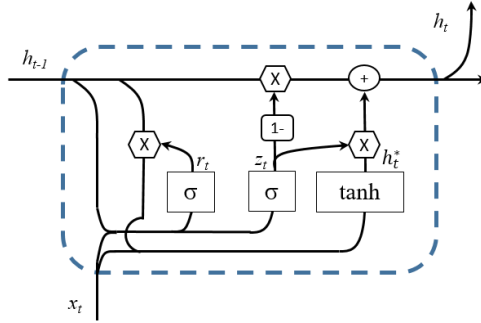


Figure 4.10. Architecture of a neural network cell with a Gate Recurrent Unit (GRU)

Source: designed by the authors based on [21]

Here through h_{t-1} is marked the initial information from the cell at the previous moment in time. This information, together with the input of the current moment of time x_t , is fed to the input gate (filter), which determines kind of information to save and pass on to the next step, and which should be «forget». In parallel with this, the update vector z_t is calculated.

Mathematically, this is written as:

$$r_t = \sigma(W_r \cdot (x_t, h_{t-1})), r_t \in [0,1], \quad (4.27)$$

$$z_t = \sigma(W_z \cdot (x_t, h_{t-1})), z \in [0,1], \quad (4.28)$$

where $\sigma(\cdot)$ – Sigmoid activation function, a W_r, W_z – weight matrices of forgetting and updating gates.

Then a new initial state vector h_t is calculated, which is a weighted linear combination of the previous output state h_{t-1} and «candidate» for saving in a new state h_t^* :

$$h_t^* = \tanh(W_h \cdot (x_t, r_t \cdot h_{t-1})), \quad (4.29)$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h_t^*, \quad (4.30)$$

where $\tanh(\cdot)$ – is a Hyperbolic Tangent activation function, and W_h is weight matrices of the output gate.

The performance of GRU-based models for many problems is comparable to that of LSTMs, but they have fewer parameters and therefore train faster, while requiring less data to train and tune.

RNNs and their modifications (LSTM, GRU) are a type of ANNs particularly well-suited for processing sequential data, such as text, speech, or time series data. Unlike traditional FNNs, RNNs have an internal state or memory that allows them to capture and incorporate information from previous inputs when processing the current input. This capability makes RNNs valuable for modelling and analysing sequences, enabling them to uncover patterns and dependencies within the data over time.

In the context of information control systems, RNNs can play a crucial role in various applications, such as Natural Language Processing (NLP), sentiment analysis, and online content moderation. For example, RNNs can be employed to analyze text data from social media platforms, news articles, or online forums to identify potentially harmful or misleading content, hate speech, or misinformation. By processing the text sequentially, RNNs can capture contextual information and understand the underlying meaning and intent, making them more effective than traditional bag-of-words models.

Moreover, RNNs can be integrated into larger ICS that monitor and analyse data streams in real-time. These systems can leverage the sequential processing capabilities of RNNs to detect patterns, trends, and potential threats or risks as they emerge. For instance, RNNs could be used to monitor online discussions and identify coordinated disinformation campaigns or the spread of harmful narratives, allowing for timely intervention or counter-messaging efforts.

In addition to content analysis, RNNs can also be employed in information control systems for user profiling and personalization. By analysing a user's historical interactions, browsing behaviour, and content consumption patterns, RNNs can learn to model individual preferences and interests over time. This information can then be used to tailor content recommendations, targeted advertising, or personalized information feeds, enhancing user engagement and serving relevant information to the right audiences.

Convolutional Neural Networks (CNNs)

CNNs are considered the best tool for identifying graphic images, finding patterns for recognizing and segmenting objects, persons, and scenes in deep machine learning [21]. They learn directly from images,

using patterns for classification, eliminating the need for manual «extraction» or feature construction.

CNNs are effectively used for image and streaming video classification and segmentation tasks. They are capable of finding hidden regularities and patterns in data, extracting features from local input patterns, and can also be used for time series analysis and forecasting tasks. The use of CNNs for deep learning has become more popular due to three important factors:

- CNNs eliminate the need for manual feature extraction, the network itself extracts or constructs features;
- CNNs have the best recognition results among neural networks of various architectures;
- they can be reconfigured to perform new recognition tasks, allowing the use and retraining of existing networks.

CNNs provide an optimal architecture for pattern detection. Combined with advances in GPUs and parallel computing, such networks are an effective technology that can be used to detect patterns in time series dynamics. A convolutional neural network can have dozens or hundreds of layers, each of which learns to detect different features of an image.

Filters are applied to each training image with different resolution, and the output of each collapsed image is used as input to the next layer. Filters start processing from the simplest «surface» characteristics of the image (at the initial convolutional layers of the network) to the characteristics that uniquely define the object (depth characteristics). Like other neural networks, CNN consists of an input layer, an output layer, and a set of hidden layers between them (Fig. 4.11).

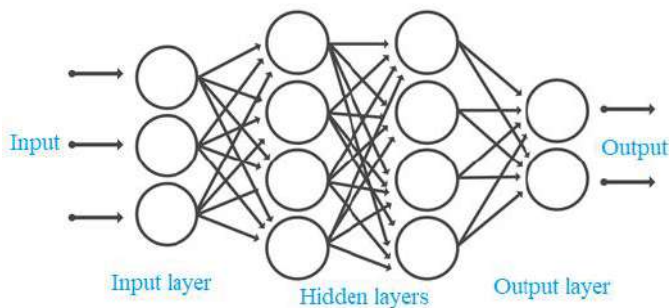


Figure 4.11. Structure of CNN

Source: designed by the authors based on [16]

These layers perform operations that modify data to learn characteristics specific to that data. The most common types of layers include:

- *Convolution* passes input images presented as numerical matrices through a set of filters, each of which activates certain features;
- *Rectified Linear Unit (ReLU)* provides faster and more efficient training, displaying negative values to zero and saving positive values. This is sometimes called activation because only activated characteristics carry over to the next level;
- *Pooling* simplifies inference by performing non-linear degradation of image quality, reducing the number of parameters the network needs to learn.

These operations are repeated on tens or hundreds of layers, while each layer learns to define different characteristics (Fig. 4.12).

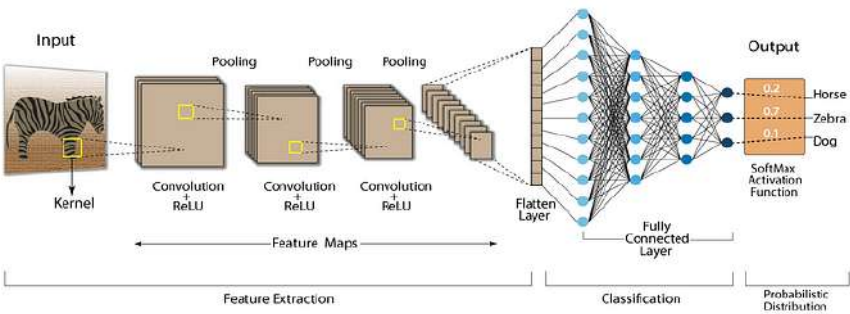


Figure 4.12. Scheme of learning process of CNN

Source: designed by the authors based on [16]

Filters are applied to each training example with different resolution, and the output of each collapsed image is used as input to the next layer.

The penultimate layer is a fully connected layer that outputs a vector of size K , where K is the number of classes the network will be able to predict. This vector contains the probabilities for each class of any image under study.

After learning the features at many levels, the CNN architecture proceeds to classification: the last layer of the CNN architecture directly performs the classification using the Softmax activation function.

The mathematical formalization of the convolution operation is given in equation (4.31), where t is time, s denotes a map of objects, w is a kernel, x is feature map of previous layer.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \quad (4.31)$$

Equation (4.32) shows the convolution operation for a two-dimensional image, where l denotes the input image, K is a kernel, (m, n) – image size, and i, j are the indexes of variables.

$$S(i, j) = (l * K)(i, j) = \sum_m \sum_n l(m, n)K(i - m, j - n) \quad (4.32)$$

Thus, a sequence of convolution layers, unification and fully connected layers form a deep convolutional network [16].

Equation (4.33) describes the operation of the fully connected layer of the convolutional neural network, where W denotes the weights, x denotes the input, b – biases, and z – output of neurons.

$$z_i = \sum_j W_{i,j}x_j + b_i. \quad (4.33)$$

At the last layer of the network, the Softmax function is used to obtain the classification result (4.34):

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \quad (4.34)$$

Applying a CNN to time series can be thought of as applying and shifting a time series filter. Unlike images, scalar time series filters have only one dimension (time) instead of two dimensions (width and height).

A filter can also be considered as a general nonlinear transformation of a time series. So, if we apply a filter of length 3 of the form $\begin{bmatrix} 1 & 1 & 1 \\ 3 & 3 & 3 \end{bmatrix}$ to a one-dimensional (1D) time series, the convolution will result in a moving average with a sliding window of length 3.

The general form for applying convolution for a centrifugal time stamp t can be defined as follows:

$$C_t = f\left(\omega * X_{\frac{t-l/2}{t+l/2}} + b\right) \forall t \in [1, T], \quad (4.35)$$

where C – is the result of a convolution (element-wise product and addition, «*») applied to a 1D time series X of length T with a filter ω of length l , a bias parameter b , and a nonlinear activation function f (for example, ReLU).

The result of the convolution (of one filter) on the input time series X can be viewed as another one-dimensional time series C that has undergone the filtering process.

Thus, the application of several filters to a time series will lead to a multidimensional time series, the dimension of which is equal to the number of used filters [21]. The same convolution (the same filter values ω and b) will be used to find the result for all timings $t \in [1, T]$.

This property of CNNs, called weight distribution, allows them to self-tune (train) filters that are time-invariant. Considering the multidimensional time series as input to the convolutional layer, the filter also has a dimension consistent with the dimension of the input multidimensional time series.

In addition to pooling layers, some deep learning architectures include normalization layers to reduce sensitivity to the initial initialization of the weights. For time series data, the batch normalization operation is performed on each channel, thus preventing the covariance shift of the internal weights during one mini-packet of data.

Another type of normalization is to normalize each instance, rather than the entire mini-batch, by determining the mean and standard deviation of each training instance for each layer using gradient descent. The last approach is called instance normalization and simulates the determination of z-normalization parameters for training time series data [16].

The last fully connected layer takes a representation of the input time series (the results of convolutions and joins) and gives a probability distribution of belonging to a certain class. Typically, this layer uses a Softmax activation function, similar to a multilayer perceptron. In general, the learning process of a deep CNN network is identical to the learning of a multilayer perceptron: as a rule, the method of error backpropagation is used.

Thus, CNNs have emerged as a powerful tool in the field of artificial intelligence, particularly for tasks involving image and video processing. These neural networks are designed to effectively capture spatial and temporal dependencies in data by applying learnable filters or kernels across the input data. The ability of CNNs to automatically learn and extract relevant features from raw data has led to their widespread adoption in various applications, including computer vision, natural language processing, and information control systems.

In the realm of ICS, CNNs can play a crucial role in various stages, such as data preprocessing, feature extraction, and decision-making. For instance, in content moderation systems, CNNs can be employed to analyse images and videos, identify potentially harmful or inappropriate content, and assist in content filtering or removal processes. By learning patterns and features from vast datasets of labelled images and videos, CNNs can achieve high accuracy in

detecting and classifying different types of content, such as explicit or violent material, hate speech, or misinformation.

Moreover, CNNs can be integrated into larger ICS that monitor and analyse data streams from various sources, including social media platforms, news outlets, and online forums. These systems can leverage the feature extraction capabilities of CNNs to identify patterns, trends, and potential threats or risks within the data. This information can then be used to inform decision-making processes, such as content moderation, risk assessment, or targeted interventions.

In addition to content analysis, CNNs can also contribute to information control systems by improving user experience and personalization. For example, recommender systems that suggest relevant content to users can employ CNNs to analyse user behaviour, preferences, and interactions with the system. By learning patterns from historical data, CNNs can help these systems provide more accurate and tailored recommendations, ultimately enhancing user engagement and satisfaction.

While CNNs have demonstrated remarkable performance in various applications, it is essential to consider ethical implications and potential biases that may arise from their use in information control systems. These systems can have significant impacts on freedom of expression, privacy, and the spread of information. Therefore, it is crucial to ensure transparency, accountability, and the incorporation of human oversight mechanisms to mitigate potential risks and biases associated with the use of CNNs and other AI models in these sensitive domains.

Similar to RNNs, the use of CNNs in ICS raises important ethical considerations. These systems can have significant impacts on freedom of expression, privacy, and the spread of information. It is crucial to ensure transparency, accountability, and the incorporation of human oversight mechanisms to mitigate potential biases and unintended consequences associated with the use of RNNs and other AI models in these sensitive domains.

Transformers

Transformers are a type of neural network architecture that has gained significant popularity in recent years, particularly in the field of natural language processing (NLP). Unlike traditional recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformers rely solely on attention mechanisms to capture the relationships between different parts of the input sequence, without relying on recurrence or convolution operations [22–24].

At the core of the transformer architecture is the attention mechanism, which allows the model to weigh the importance of different parts of the input sequence when processing a specific part of the sequence (Fig. 4.13). This mechanism enables the model to selectively focus on relevant information and capture long-range dependencies within the data, a task that was particularly challenging for traditional RNNs.

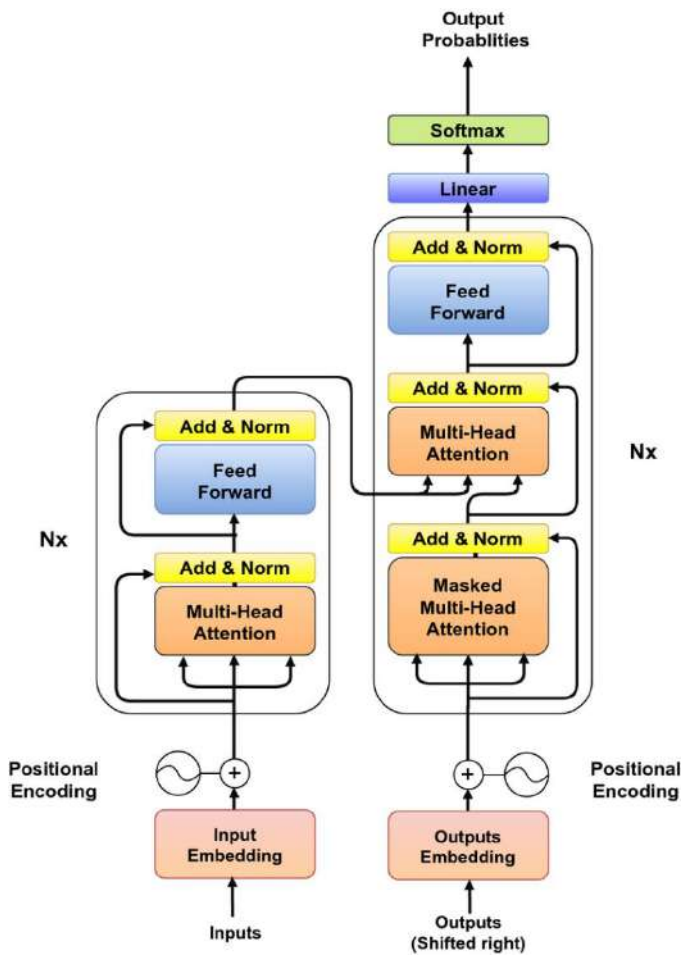


Figure 4.13. Structure of Transformer

Source: designed by the authors based on [22–24]

The transformer architecture is composed of two main components: the encoder and the decoder. Both components are based on the multi-head attention mechanism, which is the core building block of the transformer model.

Encoder: The encoder is responsible for processing the input sequence and generating a set of vector representations that capture the context and relationships within the input data. It consists of several identical layers, each containing two sub-layers: a multi-head attention layer and a position-wise fully connected feed-forward network.

- *Multi-Head Attention Layer:* This layer applies the multi-head attention mechanism to the input sequence, allowing the model to attend to different positions of the input simultaneously. The attention weights are computed based on the similarity between the query, key, and value vectors derived from the input. The outputs of the attention heads are then concatenated and linearly transformed to produce the final attention output.

- *Position-wise Feed-Forward Network:* This sub-layer consists of two fully connected layers with a ReLU activation function in between. It applies a position-wise transformation to the output of the multi-head attention layer, allowing the model to capture non-linear relationships and further refine the representations.

The encoder layers are stacked on top of each other, and the output of each layer is fed as input to the next layer, enabling the model to capture increasingly complex representations of the input sequence.

Decoder: The decoder is responsible for generating the output sequence, one token at a time. Similar to the encoder, it consists of several identical layers, each containing three sub-layers: a multi-head attention layer over the output sequence, a multi-head attention layer over the encoder output, and a position-wise fully connected feed-forward network.

- *Masked Multi-Head Attention Layer* (over output sequence): This layer applies the multi-head attention mechanism to the previously generated output tokens, allowing the model to attend to the past context when generating the next token. However, to prevent the model from attending to future tokens, a masking technique is applied to the attention weights, ensuring that the model only attends to the tokens it has already generated.

- *Multi-Head Attention Layer* (over encoder output): This layer allows the decoder to attend to the output of the encoder, enabling it to incorporate relevant information from the input sequence when generating the output.

– *Position-wise Feed-Forward Network*: Similar to the encoder, this sub-layer applies a non-linear transformation to the output of the multi-head attention layers, refining the representations further.

The decoder layers are also stacked on top of each other, with the output of each layer serving as input to the next layer. Additionally, residual connections and layer normalization are employed throughout the encoder and decoder to improve the training stability and convergence of the model.

The transformer architecture's innovative use of self-attention mechanisms and parallelization capabilities have made it a powerful and versatile model for various sequence-to-sequence tasks, such as machine translation, text summarization, and language generation.

One of the key advantages of transformers is their ability to parallelize computations, making them more efficient and scalable compared to RNNs, which process sequences sequentially. This parallelization is achieved by the attention mechanism, which operates on the entire input sequence simultaneously, rather than processing it one element at a time.

Transformers have revolutionized various NLP tasks, such as machine translation, text summarization, question answering, and language modelling. They have also found applications in other domains, such as computer vision and speech processing, where they can effectively capture long-range dependencies and leverage self-attention mechanisms.

Despite their success, transformers are computationally expensive and require large amounts of training data to achieve optimal performance. Additionally, the attention mechanism can be susceptible to certain biases and limitations, such as the inability to capture hierarchical or nested structures effectively. On-going research efforts aim to address these challenges and further improve the efficiency, interpretability, and robustness of transformer models.

Application of transformers:

- machine translation: Transformers significantly improved the quality of machine translation;
- text summarization: they can generate short descriptions of texts, preserving their main content;
- answer to questions: they can give clear and informative answers to questions posed in natural language;
- detection and correction of errors: they can detect grammatical and spelling errors in texts;
- writing texts: they can generate texts of various styles and formats, for example, articles, poems, code, etc.

Disadvantages of transformers:

- complexity: transformers can be difficult to understand and debug;
- amount of data: transformers need large amounts of data for training;
- computing resources: learning and using transformers can require significant computing resources.

Examples of successful use of transformers:

- GPT-4 model from OpenAI or Gemini from Google, which process and generate texts based on prompts;
- the Google Translate machine translation system, which uses transformers to translate texts between 135 languages.

Transformers have already significantly improved the quality of machine translation, text summarization, question answering and other NLP tasks. In the future, transformers will probably play an even bigger role in our lives, because they give us the opportunity to better understand complex processes and use natural language for this.

Transformers have emerged as a powerful architecture for artificial intelligence models, particularly in the field of NLP. Their ability to effectively capture long-range dependencies and model complex relationships within sequential data has made them invaluable in various applications, including information control systems.

In the context of ICS, transformers can play a crucial role in content moderation, misinformation detection, and sentiment analysis. By processing textual data, such as social media posts, news articles, or online forums, transformers can identify potentially harmful or misleading content, hate speech, or coordinated disinformation campaigns. Their self-attention mechanism allows them to understand the context and nuances within the text, making them more effective at identifying subtle forms of misinformation or malicious intent.

In addition to content analysis, transformers can also be employed in ICS for user profiling and personalization. By analysing a user's historical interactions, browsing behaviour, and content consumption patterns, transformers can learn to model individual preferences and interests over time. This information can then be used to tailor content recommendations, targeted advertising, or personalized information feeds, enhancing user engagement and serving relevant information to the right audiences.

Furthermore, transformers can be applied to tasks such as text summarization and question-answering, which can be valuable in ICS. For example, transformers could be used to generate concise summaries of lengthy documents or online discussions, allowing human

moderators or analysts to quickly identify key points and potential issues. Similarly, they can be employed in question-answering systems, enabling users to quickly access relevant information from large datasets or knowledge bases.

While transformers have demonstrated remarkable performance in various applications, their use in ICS raises important ethical considerations. These systems can have significant impacts on freedom of expression, privacy, and the spread of information. It is crucial to ensure transparency, accountability, and the incorporation of human oversight mechanisms to mitigate potential biases and unintended consequences associated with the use of transformers and other AI models in these sensitive domains.

4.4. Application of DNNs to multimodal data processing tasks (images, video, audio, text)

DNNs are becoming an increasingly powerful tool for solving multimodal data processing problems. Their ability to handle complex nonlinear dependencies makes them ideal for tasks where data is presented in multiple formats, such as text, images, audio, and video.

DNNs can handle complex nonlinear dependencies and can be used for training on large volumes of data. This is important for multimodal data processing problems, where the data are often very complex and multidimensional, and solving these problems is difficult or impossible for traditional methods.

We will give several examples of the application of deep neural networks to the problems of multimodal data processing.

Intelligent agents in the context of NLP are advanced software or algorithmic entities that possess the remarkable ability to interact with users through natural language. These agents are designed to comprehend and process human speech, enabling them to provide insightful answers, solve complex problems, or perform a wide range of intricate tasks tailored to the user's needs.

One prominent application of intelligent agents in NLP is the development of conversational agents, commonly known as chatbots. These agents are specifically designed to engage in natural language conversations with humans, leveraging cutting-edge language understanding and generation techniques. Chatbots can be employed in various domains, including customer service, marketing, and education, offering users a seamless and intuitive interface for accessing information, resolving inquiries, or seeking guidance.