HIS PROJECT - TSA

Task 05

Author
Alberto Mejia

Contents

1 Chapter 12: Building Blocks of Deep Learning for Time Series 3
1.1 Understanding the encoder-decoder paradigm 3
1.2 Feed-forward networks 3
1.3 Recurrent neural networks 4
1.4 Long short-term memory (LSTM) networks 5
1.5 Gated recurrent unit (GRU) 5
1.6 Convolution networks 6

1 Chapter 12: Building Blocks of Deep Learning for Time Series

1.1 Understanding the encoder-decoder paradigm

The encoder-decoder algorithm is a fundamental architecture in deep learning, often used for sequence-to-sequence tasks. Its main purpose is to take a variable-length input sequence, encode it into a fixed-length representation (the encoder part), and then decode this representation into a variable-length output sequence (the decoder part). In other words, it's used for tasks where you have an input sequence and want to generate a corresponding output sequence. The encoder-decoder architecture allows deep learning models to handle input and output sequences of different lengths, making it powerful for tasks involving sequential data. It's particularly useful for tasks where the relationship between input and output is complex and not one-to-one.

- **Input/Feature Space:** The input space, also known as the feature space, is where your original data resides. In the context of the encoder-decoder algorithm, the input space typically represents the data you're trying to process or transform.
- **Encoder:** The encoder takes the data from the input space and processes it layer by layer through neural networks.Each layer extracts increasingly abstract and meaningful features from the input data.
- **Latent Space:** The output of the encoder is the latent representation in the latent space. This latent representation is a compressed, abstract, and semantically meaningful representation of the input data. It captures the essential information required for generating an appropriate output.
- **Decoder:** Takes this latent representation from the encoder and transforms it back into a form that corresponds to the output space. The decoder leverages the information encoded in the latent space to generate an appropriate output.
- **Loss:** Is a measure of how far the prediction is from the desired output. The goal in training the model is to minimize this loss. This training process ensures that the decoder learns to extract relevant information from the latent space and use it to generate accurate outputs.

1.2 Feed-forward networks

A Feed-forward network (FFN) or fully connected network is one of the simplest neural network architectures. It takes a fixed-size input vector and processes it through a series of layers to produce the desired output. The term "feed-forward" signifies that data flows in one direction through the network, and "fully connected" means that each unit in a layer is connected to every unit in the previous and subsequent layers.

- **Input and Output Layers:** The first layer is called the input layer, and its size matches the dimension of the input data. The final layer is the output layer, which is determined by the desired output format. For example, if you need one output, you have one unit in the output layer; if you need ten outputs, you have ten units.
- **Hidden Layers:** All the layers between the input and output layers are known as hidden layers. These layers contain units that perform computations and contribute to the network's ability to learn complex patterns.
- **Network Structure:** The structure of the FFN is defined by two key hyperparameters: the number of hidden layers and the number of units in each layer. These hyperparameters determine the network's capacity to represent and learn from the data.

In the context of time series forecasting, an FFN can serve as both an encoder and a decoder. As an encoder, it can transform time series data, making it suitable for regression tasks. As a decoder, it operates on the latent vector, generated by the encoder, to produce the desired output. This decoder role is the most common use of FFNs in time series forecasting.

1.3 Recurrent neural networks

RNNs are a class of neural networks specifically designed for processing sequential data. They are particularly well-suited for tasks where the order and context of the data matter. Unlike feed-forward networks (FFNs), RNNs have connections that loop back on themselves, allowing them to maintain a hidden state, or memory, that captures information from previous timesteps.

Key Components of an RNN:

- **Hidden State (Memory):** The core component of an RNN is the hidden state, often referred to as the "memory." It's a vector that stores information about the data seen in previous timesteps. This hidden state is updated at each timestep based on the current input and the previous hidden state.
- **Input:** At each timestep, the RNN receives an input, which could be a single element of a sequence or a vector representing the entire input at that timestep. The input is used to update the hidden state.

In essence, Recurrent Neural Networks (RNNs) can be conceptualized as comprising two concurrent feed-forward networks (FFNs). The first FFN operates on the input data, mapping it to a specific output. Simultaneously, the second FFN is applied to the initial hidden state, aiming to transform it into a specific hidden state output. As a result, RNN blocks can be organized to accommodate a diverse range of input and output scenarios, including many-to-one setups, as well as many-to-many configurations. That being said, the main drawback of

RNNs in sequence modeling is the potential for vanishing or exploding gradients during backpropagation, which occurs as the network struggles with lengthy sequences, leading to either halted learning or unstable training. We can analogize this phenomenon to the process of repeatedly multiplying a scalar number by itself: If the number is less than one, it gradually diminishes with each iteration, approaching zero; conversely, if the number exceeds one, it rapidly grows exponentially.

1.4 Long short-term memory (LSTM) networks

LSTM was designed to address the challenges of vanishing and exploding gradients encountered in vanilla RNNs. The core concept behind LSTM is inspired by computer logic gates, and it introduces a crucial element known as a "memory cell" to facilitate long-term memory retention alongside the traditional hidden state memory of RNNs.

- **Input Gate:** This gate determines how much information to read from the current input and the previous hidden state.
- **Forget Gate:** The forget gate's role is to decide how much information should be discarded from the long-term memory.
- **Output Gate:** The output gate determines the extent to which the current cell state contributes to creating the current hidden state, which serves as the output of the cell.
- **Memory Cell:** It can be thought of as a container that holds information over multiple timesteps. It has the capability to store and update information for long-term dependencies, and its content is modified by the input and output gates.

So, while both RNNs and LSTMs have elements that can be seen as forms of memory, the memory cell in LSTMs is a more specialized and flexible mechanism explicitly designed to address the limitations of traditional RNNs in capturing and retaining long-term information in sequential data.

1.5 Gated recurrent unit (GRU)

The idea behind GRU is akin to using gates to control the flow of information within the network. However, GRU distinguishes itself by eliminating the long-term memory cell component found in LSTMs. Instead, it relies solely on the hidden state to convey and manage information. In essence, the hidden state itself acts as the "gradient highway," enabling information to pass through the network efficiently. Unlike LSTMs, GRUs use two key gates: the reset gate, which determines how much of the previous hidden state is considered when computing the candidate's hidden state for the current timestep, and the update gate, which governs how much of the previous hidden state is carried forward

and how much of the current candidate's hidden state contributes to the new hidden state.

In relation to the previously mentioned concepts, GRUs are another variation of recurrent neural networks designed to address the challenges faced by traditional RNNs. Similar to LSTMs, GRUs introduce gating mechanisms to control information flow, but they simplify the architecture by eliminating the dedicated memory cell. Instead, GRUs rely on the hidden state as the primary carrier of information, making them computationally more efficient and suitable for capturing dependencies in sequential data.

1.6 Convolution networks

At the core of CNNs lies the mathematical operation of convolution, which plays a central role in data processing. In the context of image data, CNNs employ kernels, which are 2D matrices with numerical values. These kernels are smaller than the images they operate on, and they are "slid" across the image, performing element-wise multiplications and summations at each position. This process allows CNNs to extract features from the data effectively.

For time series data, CNNs transition to 1D convolutions, where a 1D kernel is similarly moved along the sequence to produce an output. Unlike manually engineered features, CNNs learn the kernel weights from the data, making them adaptable and capable of generating various features. By adjusting kernel weights, CNNs can capture different characteristics in the data, enhancing their capacity as feature generators.

1D Convolution for Time Series Data:

- **Kernel Movement:** Similar to image data, 1D kernels are moved along the sequence of data points. At each position, the kernel interacts with a segment of the time series data, performing element-wise operations and producing an output value.
- **Learnable Weights:** Just like with image data, the weights within the 1D kernels are learned from the time series data during training. This adaptability enables CNNs to discover meaningful patterns or features in the sequence.
- **Feature Generation:** By adjusting the weights within the kernels, CNNs can capture different characteristics within the time series. For example, specific weights may emphasize short-term trends, while others may focus on long-term patterns. This flexibility makes CNNs powerful feature generators for time series analysis.
- **Adaptive Learning:** It's crucial to note that the values within these kernels are not fixed but are learned by the CNN during training. This adaptability allows CNNs to adjust their feature extraction behavior based on the specific dataset and task.

- **Stride:** refers to the step size at which a convolutional kernel moves across the input data during the convolution operation. It determines how much the kernel "jumps" or shifts between each operation.
- **Dilation:** Adjusts the gaps or "holes" between the kernel values during convolution, expanding the receptive field without changing the kernel size, enabling the layer to capture more input information.
- **Receptive Field:** The receptive field of a convolutional layer is the area in the input data that influences the feature generated by that layer. It's essentially the spatial extent or size of the input window over which the convolution operation is performed.

It it important to note that in multi-layered Convolutional Neural Networks (CNNs), calculating the receptive field becomes more complex due to the hierarchical structure of the network. In multi-layered CNNs, each layer processes features extracted by previous layers. As a result, the receptive field of deeper layers encompasses larger portions of the input data, allowing them to capture more context and complex patterns.