CNNS AND LSTM MODELS EXPLAINED

Abstract

The concepts of CNNs and LSTM models are explained in this documentation.

Jinyoon Kim

jinyoonok@gmail.com

Table of Contents

Neural Networks (NNs)

I) What is NNs?

Deep learning, which has significantly advanced AI in recent years, is a modern iteration of neural networks, an approach with a cyclical history of uprising and decline since its inception in 1944 by Warren McCullough and Walter Pitts. These networks were initially conceptualized as a series of interconnected processing nodes, not organized into layers and without a defined training mechanism, yet they demonstrated the potential for a neural net to perform any computation a digital computer could.

Despite their early promise, neural networks faced skepticism, particularly after Marvin Minsky and Seymour Paper published "Perceptrons" in 1969, highlighting limitations in neural nets, specifically the simple single layer Perceptrons, which were inadequate for certain computations. This critique substantially discouraged interest in neural networks, aligning with the era's shift towards digital programming.

Neural networks operate by learning from examples, typically hand-labeled data like images of various objects, which they process to find patterns corresponding to the labels. Nodes within these networks assign "weights" to their inputs and pass on data if the resulting weighted sum surpasses a certain threshold. During training, these weights and thresholds are adjusted to produce consistent outputs for given inputs. The architecture of modern neural networks involves multiple layers of nodes that data passes through unidirectionally. This structure is crucial for complex pattern recognition and has been inspired by the understanding of the human brain's processing abilities.

As we approach the discussion on Convolutional Neural Networks (CNNs), it's important to note how foundational neural networks paved the way. The concept of layers in neural networks was a foundation for the development of CNNs, which are a specialized kind of neural network with a deep architecture specifically designed to process structured arrays of data like images.

Convolutional Neural Networks (CNNs)

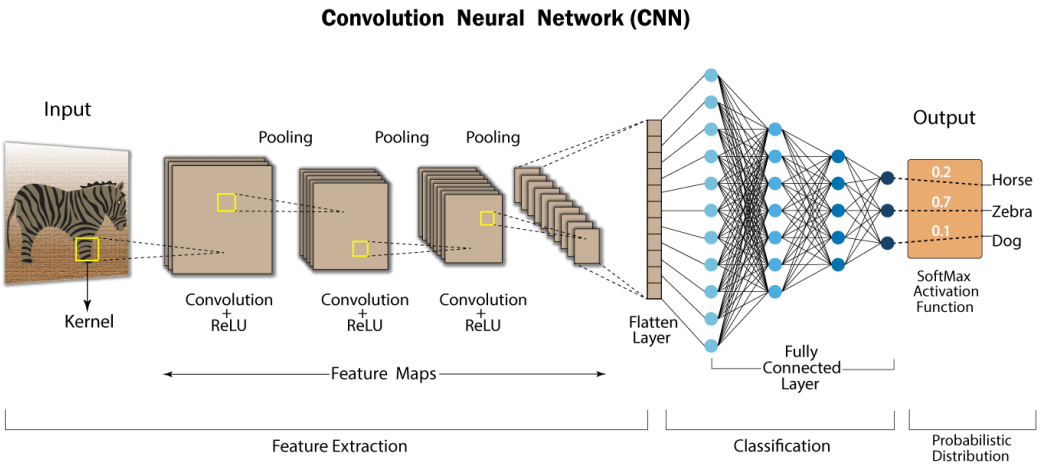
I) What is CNNs?

CNNs (Convolutional Neural Networks), which originated its concept from the study of the brain's visual cortex, has emerged as a pivotal point in the realm of artificial intelligence, particularly when dealing with visual data. This network is the primarily used technique for image and video recognition, holding a prominent position in the field of computer vision within machine learning. The digital age has driven us to the era where vast amounts of visual data are generated daily, from daily life images to medical scans. CNNs has been proven to possess the ability to autonomously navigate through this sea of data, extracting and learning vital features for a long period through myriad of applications. Whether it's distinguishing between different objects in an image through Image Classification or pinpointing the exact location of these objects via Object Detection, CNNs has proven its mettle time and again.

Delving into its structure, CNNs is a specialized type of artificial neural network. Unlike traditional neural networks that process data linearly, CNNs is designed to process data in the form of multiple arrays. This architecture mirrors the way our human visual cortex works, making CNNs exceptionally adept at analyzing visual imagery. The essence of its ability is to extract multiple levels of features from images. Starting from basic edges and textures, it delves deeper into intricate patterns and structures, constructing a rich tapestry of information. This multi-layered, hierarchical approach enables CNNs to make sense of visual data in a comprehensive manner, bridging the gap between raw pixels and meaningful content.

II) How does CNNs work?

We will analyze the process of CNNs layers as we walk through the below example.



[Credit: https://developersbreach.com/convolution-neural-network-deep-learning/]

1. Input:

The input is an image of a zebra. In the context of a CNN, this image would be represented as a matrix of pixel values. If it's a colored image, then it would typically have three channels: Red, Green, and Blue (RGB). Each channel is represented as a 2D matrix where each entry corresponds to the pixel intensity in the range [0, 255].

2. Feature Extraction:

A. Convolutional Layer (Convolution + ReLU):

The process begins by sliding the "Kernel" (a smaller-sized matrix) over the input image. This operation is known as “Convolution”. For each position of the kernel, a dot product is computed between the kernel and the portion of the input image it covers. This generates a new matrix called the 'feature map' or 'activation map'. After convolution, the CNN applies a Rectified Linear Unit (ReLU) operation, which introduces non-linearity to the system. In the diagram, this is represented by the transition from the input image to the series of feature maps. Each feature map highlights specific features of the image, like edges, textures, etc.

B. Rectified Linear Unit (ReLU):

ReLU is the most popular type of activation which is widely used in convolutional and deep neural networks. Mathematically, it's defined as:

$$f(x) = max(0,x)$$

In simpler terms, the function returns x if x is greater than or equal to zero or returns zero otherwise. Here's why it's used:

***Avoiding the Vanishing Gradient Problem:** other activation functions such like the Sigmoid or Tanh squash their output into a small range between 0 and 1 or -1 and 1 respectively. During backpropagation, this can lead to small gradients and thus the network might update the weights very slowly, making the training process much longer. On the other hand, ReLU allows larger positive values to pass through unchanged, mitigating this issue to an extent.

Introducing Non-linearity: Even though the operations like convolution are linear, having an activation function introduces non-linearity to the model. This allows the network to learn from the error and adjust, which is essential for learning complex patterns.

Computational Efficiency: ReLU is computationally efficient because it only requires a simple thresholding at zero. This allows models to train faster and requires less computational resources compared to other activation functions like the sigmoid or tanh.

C. Pooling Layer:

Following the convolution operation, the feature map undergoes pooling to reduce its dimensions. In the illustration, this is depicted by the "Pooling" blocks. The highlighted yellow boxes show the area being pooled, often taking the maximum value (Max Pooling) or average value (Average Pooling) from that area. Pooling helps in reducing computational costs and provides a form of translation invariance to the network.

3. Classification:

A. Flatten Layer:

After several convolutional and pooling layers, the feature maps are flattened into a single long vector. This is to prepare the data for the next step, which involves traditional neural network layers.

B. Fully Connected Layer:

This is where the high-level reasoning happens. Every neuron in a fully connected layer is connected to every neuron in the previous layer, as indicated by the densely interwoven lines in the illustration. The primary purpose of the fully connected layers is to use these features for classifying the input image into various categories.

C. SoftMax Activation Function:

The final layer in the CNN typically has as many neurons as there are classes for the classification task (In our example: Horse, Zebra, Dog). The SoftMax function is applied to the outputs to convert them into probability scores for each class. As we can see in the diagram, the model has assigned the highest probability to "Zebra", which indicates that the model has recognized the image as a zebra.

4. Summary:

In essence, the CNNs starts by extracting low-level features (like edges and textures) using convolution and pooling. As it progresses deeper into the network, it starts recognizing more complex features. By the end of the network, these features are utilized to make a classification decision. The entire process, from feature extraction to classification, is learned automatically from the data during training.

III) CNNs on numerical data

CNNs shine when dealing with numerical data due to their structure, which is inherently designed to recognize and interpret complex patterns. This capability is not limited to visual data; it extends to any numerical dataset where patterns and structures can be discerned. The foundation of CNNs lies in their convolutional layers, which perform a form of mathematical operation on numerical data to extract features. This operation is comparable to a filter that highlights certain aspects of the data while downplaying others, making it particularly effective at uncovering hierarchical patterns.

Images are typical examples of numerical data sets where each pixel value contributes to a larger pattern or feature within the image. CNNs excel at image processing because they transform the raw numerical pixel data through multiple layers of convolutions, pooling, and fully connected layers to understand and classify the content of the image. By learning from the intricate structure of the numerical data that makes up an image, CNNs are capable of tasks ranging from recognizing faces to interpreting medical scans. This same principle can be applied to any form of numerical data with spatial or structured patterns, enabling CNNs to tackle a wide array of tasks with high efficiency.

IV) CNNs on regression

Convolutional Neural Networks (CNNs), predominantly known for their success in classification tasks, can also be adeptly used for regression problems. In this context, CNNs learn to predict continuous values rather than discrete classes. They do this by adjusting their final layer to output a single value or a vector of continuous values, and by using a loss function that measures the difference between the predicted and actual values, such as mean squared error, to guide the training process.

In the example of the paper "A novel deep Convolutional Neural Network-based regression approach has been proposed for estimating the Remaining Useful Life of components or systems (Sateesh Babu, Zhao, & Li, 2016)," CNNs are employed for a regression task that involves estimating the Remaining Useful Life (RUL) of machinery. This is a significant departure from their more common application in image processing domains. CNNs in this case learns from time-

series data obtained from sensors to predict how long a component will last before it needs maintenance or replacement. Since such an application is not very common, we will outline the features employed for regression within the example provided by the paper.

Key elements of how CNNs are used for regression problem according to the paper include:

Temporal Feature Learning: Instead of recognizing spatial patterns as in image recognition, the CNNs here identifies patterns over time, learning from the sequence of sensor readings that correspond to the health and operational status of the machinery.

Deep Learning Architecture: CNNs uses multiple layers to learn a hierarchy of features from raw sensor data. Lower layers might detect simple patterns such as increases in temperature or vibration, while deeper layers interpret these patterns in terms of the machinery's degradation.

Supervised Learning: The CNNs is trained on historical sensor data where the actual RUL is known, allowing the network to learn the correlation between sensor readings and RUL.

Regression Output: The CNNs architecture is modified for regression by including a final layer that predicts a continuous value (the RUL) instead of a class label. The loss function is also chosen to reflect the regression nature of the problem, such as the mean squared error between the predicted RUL and the actual RUL.

By adapting CNNs to this predictive task, the authors of the paper can leverage the network's capability to learn complex, non-linear representations from large amounts of high-dimensional data, resulting in a model that outperforms traditional linear models and can more accurately predict the RUL. This demonstrates the flexibility of CNNs and their potential beyond typical classification tasks.

V) Why are they important?

Autonomous Feature Learning: In the early days of computer vision, the onus was on scientists and engineers to manually design and extract features that would allow algorithms to make sense of images. This was both time-consuming and often lacked precision. However, CNNs possesses the innate ability to automatically learn and sharpen the most pertinent features directly from data. This has led to models that are not just more robust and accurate, but also models that can be trained with less human bias and effort, paving the innovative way for more substantial understanding of visual data.

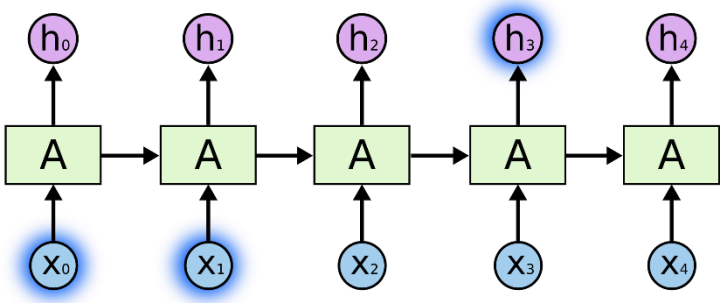
High Accuracy: It's not an overstatement to say that CNNs have revolutionized computer vision. They consistently set benchmark records, often leaving traditional algorithms in the dust. This unmatched accuracy has propelled advancements in various fields, from self-driving cars that can interpret their surroundings in real-time, to medical imaging where early disease detection can save lives.

Versatility: While CNNs have made their name in the realm of visual data, their application is not confined to it. Researchers are finding innovative uses for CNNs in areas like natural language processing, audio recognition, and even in complex systems like weather forecasting. Their adaptability speaks to the foundational strength of their design.

Real-world Applications: The ripple effects of CNNs in the real world are vast. In healthcare, they aid in diagnosing diseases with unprecedented accuracy. In the entertainment industry, they're behind the realistic graphics of blockbuster movies and video games. Financial institutions use them for fraud detection, ensuring the security of our transactions. Even in areas like security and surveillance, CNNs play a pivotal role in threat detection and crowd management.

LSTM (Long Short-Term Memory)

I) What is LSTM?

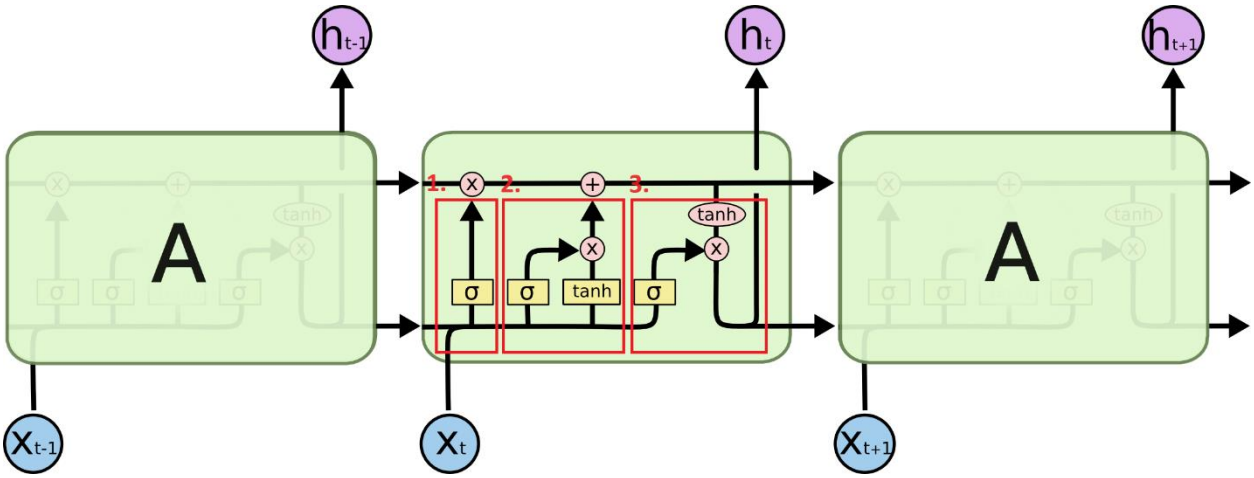


[Credit: https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

Our brains understand words based on the context and sequence in which they appear. Similarly, Recurrent Neural Networks (RNNs) were designed to remember past information, enabling them to understand sequences better. The above image illustrates the fundamental mechanism of an RNN. Each green box, "A", can be thought of as the RNN's memory cell, which computes and forwards information. At every timestep, the RNN receives an input denoted as ***xi***. This input can be considered a word in a sentence. Crucial to its design, the RNN passes information from one step to the next. This 'memory' is depicted by the horizontal arrows linking each "A" cell. This continuous chain of passing information allows the RNN to remember previous inputs in its sequence. After incorporating both the current input and the passed-on memory, the RNN produces an output for that timestep, shown as ***hi***.

However, while RNNs are great in theory, they have practical difficulties. When dealing with long sequences, they tend to forget the earlier parts of the sequence. This is primarily due to the vanishing gradient problem, which makes it hard for RNNs to learn and retain long-term dependencies. **LSTM (Long Short-Term Memory)** units were introduced to overcome these shortcomings. They possess a more intricate internal structure compared to the standard RNN, specifically designed to remember long-term dependencies. LSTM achieves this by utilizing more complex gates that regulate the flow of information, ensuring that the network can learn from early parts of a sequence and use this information to inform later parts, thus resolving the downsides of the basic RNNs.

II) How does LSTM work?



[Credit: https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

The LSTM is a specialized type of RNN that is designed to handle the challenges associated with learning long-term dependencies in sequences. It is achieved through its inner complicated architecture, which you can see in the above diagram, which differs from the straightforward loop mechanism of traditional RNNs.

1. Cell State:

The Cell State(Horizontal Line across the cell) is the backbone of the LSTM, serving as its memory track. It stretches the length of the entire LSTM cell and undergoes only minor linear modifications. The key feature here is that the LSTM can add or remove information to the cell state, regulated by structures called gates.

2. Gates:

Gates are essential components in the LSTM cell and are absent in the original RNNs. These gates decide how information flows from entering to leaving, and even being deleted from the cell state.

A. Forget Gate:

The forget gate's primary function is to determine which portions of the cell state should be discarded or kept. It takes the previous hidden state h_{t-1} and the current input x_t as its inputs and runs them through a sigmoid function. The sigmoid function outputs values between 0 and 1,

indicating whether a particular component of the cell state should be forgotten (values close to 0) or retained (values close to 1).

B-1. Input Gate:

A tanh layer processes the input x_t and the previous hidden state h_{t-1}. This layer generates a vector of potential new values that might be added to the state. These values represent the new information the LSTM might consider storing.

B-2. Input Gate:

The input gate manages the actual updates to the cell state based on the candidate values. It consists of a sigmoid function layer, often referred to as the **"input gate layer"**. This function determines which parts of the candidate values will be permitted to update the cell state.

The candidate values are element-wise multiplied by the output of the sigmoid function. This multiplication filters which parts of the candidate information get incorporated into the cell state.

C. Output Gate:

Deciding what the LSTM should output at each step is the output gate's job. It uses the current input x_t and the previous hidden state h_{t-1} and subjects them to a sigmoid function to decide which parts of the cell state will be shown in the output. The cell state is also processed by a tanh function to normalize its values between -1 and 1, and this result is multiplied by the output of the sigmoid gate, producing the final output h_t.

3. Summary:

Through this interplay of gates, cell state, and other components, the LSTM ensures the network selectively focuses on relevant inputs over varying time steps. This capability to recognize and preserve long-term patterns and relationships distinguishes LSTMs from standard RNNs and equips them to excel in tasks that require understanding over extended sequences.

III) LSTM on numerical data

LSTM is particularly adept at handling numerical data that is sequential or time dependent. Unlike CNNs, which excel at identifying spatial hierarchies in data such as images, LSTM is designed to recognize patterns in sequences of data over time. This makes them especially suitable for tasks like time-series forecasting, where each data point is related to the previous ones in a meaningful way.

For numerical data that does not have a temporal sequence, LSTM may not be the best choice. They are specialized for scenarios where the context provided by the order of data points is crucial. For instance, LSTM can predict the next number in a series by learning from the history of previous numbers and their progression. This is something CNNs is not inherently designed to do, as they lack the internal state mechanisms that LSTM uses to remember and utilize past information.

IV) LSTM on Regression

LSTM is particularly effective for regression problems where the data is sequential, and the task is to predict future numerical values. These networks excel at capturing temporal dependencies, making them ideal for time-series forecasting, such as predicting stock prices, energy consumption, or sales trends. Let's delve into how these capabilities are applied through each step of the regression process with LSTM.

Problem Understanding: LSTM is uniquely qualified for sequential data due to their architecture, which allows them to retain information over long periods. This is essential for understanding and forecasting based on historical data, where the order of data points is crucial.

Feature Engineering: LSTM benefits significantly from feature engineering, which can highlight temporal dynamics that may not be immediately obvious. For example, incorporating features like time of day, day of the week, and seasonal indicators can enhance the LSTM's predictive power by providing additional context.

Data Scaling: Scaling the data is a vital preprocessing step for LSTM. It ensures that the input features have a similar scale, preventing any one feature from disproportionately influencing the model's weights, which helps the LSTM learns and converges faster.

Sequence Creation: By structuring the input data into sequences, LSTM can use their recurrent connections to learn from the past information. This sequence creation step is key to leveraging the LSTM's ability to remember long-term dependencies.

Model Architecture: Building an LSTM for regression involves designing a network that can process sequences of inputs and output a continuous value. The model architecture typically includes one or more LSTM layers, which are specifically suited to handle the temporal sequence data.

Model Configuration: Selecting the right number of neurons and layers is critical in an LSTM. Too few might not capture the complexity of the data, while too many could lead to overfitting. The model configuration step ensures the LSTM has the appropriate capacity to model the regression problem effectively.

Training Process: During training, the LSTM updates its internal state to minimize a loss function. This process allows the LSTM to learn the intricate patterns in the sequential data, improving its ability to make accurate predictions for the regression task.

Prediction: After training, the LSTM uses its learned weights to make predictions. Its ability to remember past information allows it to forecast future numerical values with a context that is informed by the sequential nature of the data.

Evaluation: Finally, evaluating the LSTM's predictions against actual outcomes is essential to measure its performance. Metrics like RMSE provide a quantitative assessment of the LSTM's regression capabilities, ensuring that the model's predictions are accurate and reliable.

Through these steps, LSTMs demonstrate their prowess in handling regression tasks with sequential data, offering powerful and insightful forecasts that are invaluable across numerous applications.

V) Why are they important?

LSTMs, with their capability to capture long-term dependencies in sequences, hold a significant place in the realm of deep learning and artificial intelligence. Their importance is underscored by their broad applications and the pivotal role they play in various tasks.

Long-Term Dependencies: Traditional RNNs face difficulty when trying to learn and remember information from long sequences, often referred to as the vanishing gradient problem. LSTMs, with their unique structure, can effectively remember information over long periods, making them suitable for tasks that involve long sequences.

Robustness to Sequence Length Variability: Unlike some other algorithms that might require fixed-size input, LSTMs can handle sequences of varying lengths, a property essential for many real-world applications.

Real-world applications: LSTMs are instrumental in diverse domains due to their capacity to recognize and remember temporal patterns. They shine in tasks like machine translation within Natural Language Processing, speech-to-text conversion for speech recognition, forecasting in time-series analysis, generating music based on note sequences, and analyzing video sequences to detect activities. Their versatility in handling sequence data makes them a cornerstone in many advanced solutions across fields.

Reference:

Hardesty, L. (2017, April 14). Explained: Neural networks. MIT News Office. Retrieved November 7, 2023, from https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

Sateesh Babu, G., Zhao, P., Li, XL. (2016). Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life. In: Navathe, S., Wu, W., Shekhar, S., Du, X., Wang, X., Xiong, H. (eds) Database Systems for Advanced Applications. DASFAA 2016. Lecture Notes in Computer Science(), vol 9642. Springer, Cham. https://doi.org/10.1007/978-3-319-32025-0_14

Swapna K E. (n.d.). Convolution Neural Network – Deep Learning. Developers Breach. Retrieved from https://developersbreach.com/convolution-neural-network-deep-learning/

CNN Explainer. (n.d.). Retrieved October 30, 2023, from https://poloclub.github.io/cnn-explainer/

Mishra, M. (2020, August 26). Convolutional Neural Networks, Explained. Medium. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

Olah, C. (2015, August 27). Understanding LSTM Networks. [Blog post]. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Saxena, S. (2023, October 25). What is LSTM? Introduction to Long Short-Term Memory. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

Dolphin, R. (2020, October 21). LSTM Networks | A Detailed Explanation to LSTMs. Medium. https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

Alhamid, M. (2021, June 26). LSTM and Bidirectional LSTM for Regression. Towards Data Science. https://towardsdatascience.com/lstm-and-bidirectional-lstm-for-regression-4fddf910c655

So, G. (2019, March 29). Should we abandon LSTM for CNN? Medium. https://medium.com/ai-ml-at-symantec/should-we-abandon-lstm-for-cnn-83accaeb93d6

Roy, R. (2021, June 25). LSTMs for regression. Medium. https://bobrupakroy.medium.com/lstms-for-regression-cc9b6677697f