20 - Prática - Visão Computacional (III)

Lucas Samuel Zanchet

Section 1:

Working with tensors is more efficient because they are specifically designed to handle multidimensional arrays of data, which are a specific need for machine learning and deep learning tasks. Tensors leverage optimized mathematical operations, such as matrix multiplications and element-wise computations, that are crucial for training and inference in machine learning models. These operations are highly parallelizable, making them ideal for execution on GPUs, which excel at handling large-scale parallel computations. By offloading tensor operations to GPUs, processing times are significantly reduced compared to performing these computations on CPUs. Additionally, tensors are a core component of popular machine learning frameworks like TensorFlow and PyTorch, which provide further optimizations, such as automatic differentiation and memory management, to enhance performance and efficiency in developing and deploying machine learning models.

Section 2:

Workflow:

The workflow for a neural network project typically begins with clearly defining the goal of the project. This involves identifying the specific problem to solve, the expected outputs, and the metrics that will be used to evaluate success. Once the goal is set, the next step is to gather as much relevant data as possible from reliable and viable sources. This data serves as the foundation for training the neural network, so its quality and relevance are critical.

After data collection, the dataset goes through a preprocessing stage to prepare it for use. Preprocessing typically includes cleaning the data by removing unusable entries, such as outliers, and addressing issues like missing values by imputing them or discarding incomplete records. During this stage, the data may also be normalized or scaled, and categorical data can

be encoded for compatibility with machine learning algorithms. Even after preprocessing and modifying the dataset, it is essential to ensure there is still enough data to comprehensively capture the patterns and variability needed for the model to generalize effectively.

Once the data is ready, attention shifts to the model's architecture. This involves deciding how the layers of the neural network will be organized, selecting the number of neurons in each layer, and choosing activation functions that dictate how the network learns and transforms data. The choice of architecture is critical, as it directly impacts the model's ability to learn and generalize.

With the architecture finalized, the next step is training the model. Training involves feeding the prepared data into the model, adjusting its weights and biases through backpropagation and optimization algorithms (e.g., gradient descent), and iteratively refining the model to minimize the error.

After training, the model's performance must be rigorously evaluated using a test dataset. This dataset is separate from the training data and represents real-world scenarios the model is likely to encounter. Evaluation helps determine how well the model generalizes to unseen data, ensuring it performs effectively and reliably in real-world applications. If the performance is not satisfactory, the workflow may loop back to earlier stages, such as improving the data or tweaking the architecture, to enhance the model's results.

Architecture:

The architecture of a NN is the disposition of nodes and connections inside the model. This structure defines the ability of the model to process data. Different architectures excel better in some applications than others, and heavy research is put upon the best architecture for a given task.

At a low level, the model is a bunch of numbers that represent the weights and biases that process the input in a specific sequence. These numbers are updated through.

For some applications we utilize CNNs (Convolutional Neural Networks), which consist of a kernel, that is, a small matrix that is learned by the model to

recognize certain patterns. It's mostly used in sound, image and video processing, or in any application where positional or temporal patterns can be useful.

Another type of NN are RNNs (Recurrent Neural Network), where the output of the data is fed back into its inputs. This method is useful in text recognition or generation.

Section 3:

Activation Layers:

Activation layers are a critical component of neural networks, as they apply activation functions to the outputs of neurons. These functions introduce non-linearity into the model, allowing it to learn complex patterns and relationships in the data that linear transformations alone cannot capture. Popular activation functions include ReLU (Rectified Linear Unit), which is computationally efficient and helps mitigate the vanishing gradient problem by setting negative values to zero, and sigmoid or tanh functions, which are often used in tasks like binary classification or when smooth gradients are required.

Loss Functions:

Loss functions measure the difference between the model's predictions and the actual target values, providing a quantitative metric to optimize during training. The loss function acts as a guide for the optimization algorithm, such as gradient descent, to adjust the model's weights and biases in the direction that minimizes error. Different tasks require different loss functions—for instance, Mean Squared Error (MSE) is commonly used for regression problems, while Cross-Entropy Loss is ideal for classification tasks.

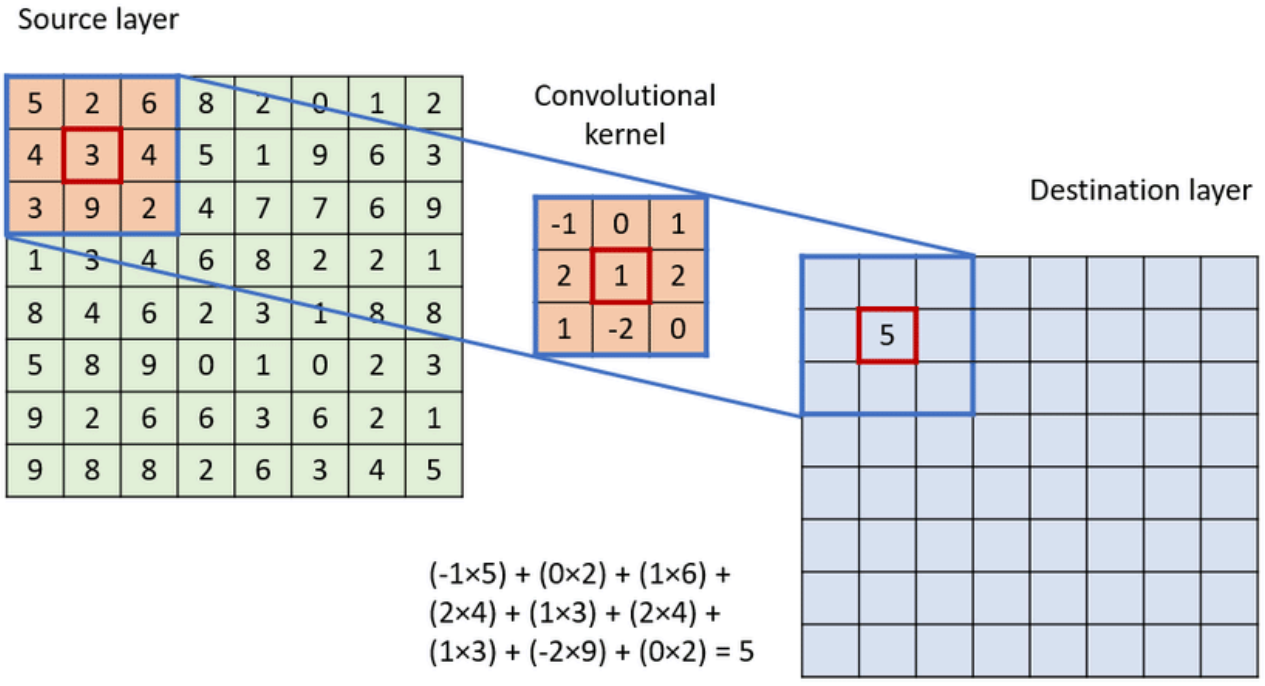
Optimizers:

Optimizers are algorithms used in training neural networks to adjust the model's weights and biases based on the loss function's feedback. Their goal is to minimize the loss function by iteratively updating the parameters, helping the model learn patterns in the data effectively. Optimizers use the gradients of the loss function, calculated via backpropagation, to determine the direction and magnitude of updates.

There are various optimizers, each suited to different scenarios. **Gradient Descent** is the simplest, where weights are updated uniformly across all training examples. Variants like **Stochastic Gradient Descent (SGD)** introduce randomness by updating weights based on a subset of the data (a mini-batch), making training faster and more scalable. More advanced optimizers, such as **Adam (Adaptive Moment Estimation)**, combine the advantages of SGD and adaptive learning rates by maintaining momentum and scaling updates for each parameter individually.

Section 4:

Convolutional layers are the building blocks of Convolutional Neural Networks (CNNs) and are designed to process grid-like data, such as images. They use filters (or kernels) that slide over the input data, performing element-wise multiplications to extract spatial features like edges, textures, or patterns. These layers are highly efficient, as they reduce the number of parameters compared to fully connected layers, while preserving the spatial structure of the data. Convolutional layers are critical for tasks like image classification, object detection, and segmentation, as they effectively capture hierarchical features from simple to complex.



Hyperparameter Tuning:

Hyperparameter tuning is the process of optimizing the settings of a machine learning model that are not learned during training, such as learning rate, batch size, number of layers, or number of neurons per layer. These hyperparameters greatly influence model performance, training speed, and generalization. Tuning can be done manually, through grid search, or more efficiently with techniques like random search or Bayesian optimization. Proper hyperparameter tuning helps achieve the best possible results for a given task without overfitting or underfitting.

Data Normalization:

Neural Networks are more efficient when data is normalized due to the removal of biases and inconsistencies. Normalization is usually done to values between 0 and 1 or -1 and 1.

Section 5:

Why CNNs?

Using a Convolutional Neural Network (CNN) to analyze images is more effective because CNNs are specifically designed to process grid-like data, such as pixel arrays in images. They are better at capturing spatial hierarchies and patterns, such as edges, textures, and complex features.

Unlike traditional machine learning models, CNNs leverage shared weights and local connections, significantly reducing the number of parameters and computational complexity. This makes them both efficient and scalable for high-dimensional data like images. Furthermore, CNNs automatically learn feature representations, removing the need for manual feature extraction. Their ability to recognize patterns regardless of position or orientation in an image makes them ideal for tasks such as object detection, image classification, facial recognition, etc..

Section 6:

Auto Encoders:

Auto Encoders are a type of neural network that is able to encode and decode some information. It's being used lately for data compression like NVIDIA's

new image compression algorithm for online transferring. The reason is the fact that the encoder converts the input data into a lower dimensional latent space, which is just an abstract representation of data that captures the most meaningful features of the input.

Variational Auto Encoders (VAE):

VAEs are a type of autoencoder that uses Bayesian inference in the learning process. The encoder instead of generating only one latent space vector, it generates a mean vector and a variance vector which are then used to sample the latent space.

The difference between traditional autoencoders and VAEs is that VAEs can output smooth interpolations between training data, which makes it capable of generating new data.

The principle is ensuring that the latent space follows a defined probability distribution. This is achieved through the following loss function:

$$\mathcal{L}_{VAE} = \mathcal{L}_{reconstruction} + \beta \cdot D_{LK}(q(z|x)||p(z) \tag{1}$$

Where D_{LK} is the Kullback-Leibler divergence function, which:

> is a type of **statistical distance**: a measure of how one reference probability distribution P is different from a second probability distribution Q Mathematically

In (1), $q(z \mid x)$ represents the probability distribution of the latent variable z given the input data x. This ensures the latent space is smooth, meaning small changes in the latent variables result in small changes in the output. Also, the latent variables are structured and follow the standard normal distribution, making it easy to sample new data from the space.

CONCLUSION

The course covers key concepts and methodologies in neural network development. Frameworks like TensorFlow and PyTorch make it easy to utilize the modern hardware for accelerating learning and predicting with NNs. Research in architectures such as CNNs, RNNs, Encoders, and many others has opened many new applications for Neural Networks.

REFERENCES

1. Pinheiro Cinelli, Lucas; et al. (2021). "Variational Autoencoder". *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer. pp. 111–149. doi:10.1007/978-3-030-70679-1_5. ISBN 978-3-030-70681-4. S2CID 240802776.