Application of Machine Learning Methods on Astronomical Databases

Apostolos Kiraleos

Abstract

Galaxy redshift is a crucial parameter in astronomy that provides information on the distance, age, and evolution of galaxies. This dissertation investigates the application of machine learning for predicting galaxy redshifts. It involves the development and training of a neural network to analyze galaxy spectra sourced from the European Space Agency's Gaia mission and showcases the practical implementation of machine learning in astronomy.

Table of Contents

List of Figures

1 Introduction

1.1 Object

This dissertation primarily addresses the critical issue of accurate galaxy redshift estimation. Galaxy redshifts are pivotal in modern astronomy, providing crucial insights into cosmic distances, universe evolution, and more. The topic is of significant interest and relevance within the field.

1.2 Purpose and Objectives

The main goal of this dissertation is to advance galaxy redshift estimation by applying advanced machine learning techniques, specifically Convolutional Neural Networks (CNNs). The objective is evaluating the performance and accuracy of a trained CNN model in predicting galaxy redshifts with the aim to understand the practical applications of this model in astronomical research.

1.3 Methodology

The methodology of this dissertation involves designing, training, and evaluating the performance of a trained Convolutional Neural Network (CNN). The CNN is trained on a dataset of galaxy spectra sourced from the European Space Agency's Gaia mission which came already preprocessed and cleaned, ready for model training. The CNN is then trained on the dataset and evaluated on a test set of galaxy spectra.

For the actual implementation of the CNN, the Python programming language was used, along with the Keras deep learning library.

1.4 Innovation

The contribution of this work lies in advancing knowledge within the field of galaxy redshift estimation. By applying state-of-the-art machine learning techniques, we aim to provide an innovative approach to predicting galaxy redshifts, potentially improving accuracy and efficiency in astronomy.

1.5 Structure

The subsequent chapters of this dissertation are organized as follows:

1. **Introduction** (the current chapter): Provides an overview of the dissertation's focus, objectives, methodology, and innovation.
2. **Background**: Introduce foundational concepts related to the Gaia mission, galaxy redshift and machine learning methods.
3. **Data Preparation**: Details the process of selecting and cleaning galaxy spectra data for model training.
4. **Methodology**: Elaborates on the methodologies used for CNN model training and evaluation.
5. **Results and Discussion**: Presents the findings of our experiments, assesses model performance, and discusses implications.
6. **Conclusion**: Summarizes key takeaways and outlines potential areas for future research.

2 Background

2.1 Gaia space obvservatory

The Gaia mission is a European Space Agency (ESA) space observatory that has been in operation since 2013. The primary goal of the mission is to create a three-dimensional map of our galaxy by measuring the positions, distances, and motions of over two billion stars.

At its core, Gaia is equipped with two optical telescopes accompanied by three scientific instruments, which collaborate to accurately ascertain the positions and velocities of stars. Additionally, these instruments disperse the starlight into spectra, facilitating detailed analysis.

Throughout its mission, the spacecraft executes a deliberate rotation, systematically scanning the entire celestial sphere with its two telescopes. As the detectors continuously record the positions of celestial objects, they also capture the objects' movements within the galaxy, along with any alterations therein.

During its mission, Gaia conducts approximately 14 observations each year for its designated stars. Its primary goals include accurately mapping the positions, distances, motions, and brightness variations of stars. Gaia's mission is expected to uncover various celestial objects, including exoplanets and brown dwarfs, and thoroughly study hundreds of thousands of asteroids within our Solar System. Additionally, the mission involves studying over 1 million distant quasars and conducting new assessments of Albert Einstein's General Theory of Relativity.[1]

2.2 Galaxy redshift

Galaxy redshift is a fundamental astronomical property that describes the relative motion of a galaxy with respect to Earth. The redshift of a galaxy is measured by analyzing the spectrum of light emitted by the galaxy, which appears to be shifted towards longer wavelengths due to the Doppler effect. Redshift is a crucial parameter in astronomy, as it provides information about the distance, velocity, and evolution of galaxies.

The redshift of a galaxy is measured in units of "z", which is defined as the fractional shift in the wavelength of light emitted by the galaxy. Specifically, the redshift "z" is defined as:

$$z = \frac{\lambda_{obsv} - \lambda_{emit}}{\lambda_{emit}}$$

where λ_{obsv} is the observed wavelength of light from the galaxy, and λ_{emit} is the wavelength of that same light as emitted by the galaxy. A redshift of $z = 0$ corresponds to no shift in the wavelength (i.e., the observed and emitted wavelengths are the same), while a redshift of $z = 1$ corresponds to a shift of 100% in the wavelength (i.e., the observed wavelength is twice as long as the emitted wavelength).

Accurate and efficient estimation of galaxy redshift is essential for a wide range of astronomical studies, including galaxy formation and evolution, large-scale structure of the universe, and dark matter distribution. However, measuring galaxy redshifts can be a challenging task due to various factors such as observational noise, instrumental effects, and variations in galaxy spectra.[2]

[1] "Gaia Overview". ESA. September 26 2023. https://www.esa.int/Science_Exploration/Space_Science/Gaia/Gaia_overview

[2] "Redshift". Wikipedia. September 26 2023. https://en.wikipedia.org/wiki/Redshift

2.3 Convolutional Neural Networks

2.3.1 Overview

Convolutional neural networks (CNNs) are a type of artificial neural network (ANN) that has proven to be highly effective for tasks involving image and video analysis, such as object detection, segmentation, and classification. CNNs are inspired by the structure and function of the visual cortex in animals, which contains specialized cells called neurons that are tuned to detect specific visual features. In a similar way, CNNs are designed to learn and extract meaningful visual features from raw image data.

At the core of a CNN are individual processing units called neurons, which are organized into layers. Each neuron receives input from other neurons in the previous layer, applies a mathematical operation to that input, and passes the output to the next layer. The output of the final layer of neurons is the predicted output of the network for a given input.

The key innovation of CNNs is their use of convolutional layers, which enable the network to automatically learn and extract local spatial features from raw input data. In a convolutional layer, each neuron is connected only to a small, localized region of the input data, known as the receptive field. By sharing weights across all neurons within a receptive field, the network can efficiently learn to detect local patterns and features, regardless of their location within the input image.

CNNs typically also include pooling layers, which downsample the output of the previous layer by taking the maximum or average value within small local regions. This helps to reduce the dimensionality of the input and extract higher-level features from the local features learned in the previous convolutional layer.

The final layers of a CNN are fully connected layers, which take the outputs of the previous convolutional and pooling layers and use them to make a prediction. In the case of image classification, for example, the output of the final fully connected layer might be a vector of probabilities indicating the likelihood of each possible class. In our case, instead of class probabilities, the output of the final fully connected layer will yield a single numeric value representing the predicted redshift of the observed galaxy.[3]

2.3.2 Mechanism of operation

Neural networks operate through a process known as backpropagation, a fundamental training algorithm that enables them to progressively refine their predictive capabilities. This iterative process involves adjusting the network's internal parameters, namely its weights and biases, to minimize a specified loss function. The loss function serves as a critical measure of the disparity between the network's predictions and the actual target values. By diminishing this loss, the neural network enhances its capacity to make increasingly accurate predictions.[4]

The core of the backpropagation process hinges on an optimization technique known as gradient descent. Gradient descent, in essence, determines how the network's weights should be updated to minimize the loss function. To achieve this, it computes the gradient of the loss function with respect to each weight in

[3] "Convolutional Neural Networks". Wikipedia. September 26 2023. https://en.wikipedia.org/wiki/Convolutional_neural_network

[4] "Backpropagation". Wikipedia. September 27 2023. https://www.wikiwand.com/en/Backpropagation

the network. In other words, it calculates the rate of change of the loss concerning individual weights. This gradient offers crucial guidance, pointing the way toward the most rapid reduction in the loss function.

As the gradient highlights the direction of steepest descent, it becomes the compass for the adjustment of weights. Weight updates are made proportionally to the gradient, ensuring that changes are made more significantly in areas where the loss function is decreasing most rapidly. This iterative process of calculating gradients and updating weights continues until the network converges to a state where the loss is minimized to the greatest extent possible.[5]

Incorporating the principles of backpropagation and gradient descent, neural networks steadily evolve their internal representations, allowing them to capture complex patterns and relationships within the data. This mechanism of operation serves as the foundation for their impressive ability to learn and generalize from data, making them valuable tools in a wide array of applications, from image recognition to galaxy redshift estimation.

2.3.3 Loss functions

Loss functions are mathematical functions that are used to measure the difference between the predicted output of a neural network and the actual output. They are used to guide the training process by indicating how well the network is performing. The most common loss functions used in neural networks are the mean squared error (MSE) and mean absolute error (MAE) functions.

The MSE function is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The MAE function is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the actual output and $\hat{y}_i$ is the predicted output for the ith sample, and n is the total number of samples.

Another loss function that is commonly used in neural networks (and the one we end up using) is the Huber loss function. It is less sensitive to outliers in data than the squared error loss. It's defined as:

$$L_\delta = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{otherwise}\end{cases}$$

where δ is a small constant.

2.3.4 Activation functions

Activation functions are mathematical functions that are applied to the output of each neuron in a neural network. They are used to introduce non-linearity into the network, which is essential for learning complex patterns and relationships in the data. The most common activation functions used in neural networks are the

[5] "What is Gradient Descent?". IBM. September 27 2023. https://www.ibm.com/topics/gradient-descent

sigmoid, tanh, and ReLU functions. For *convolutional* neural networks, the ReLU function is typically used for all layers except the output layer, which uses a linear activation function.[6]

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

where x is the input to the function. The sigmoid function is a smooth, S-shaped function that returns a value between 0 and 1. It is commonly used in binary classification problems, where it is used to convert the output of the final layer to a probability between 0 and 1.[7]

The tanh function is defined as:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

where x is the input to the function. The tanh function is a smooth, S-shaped function that returns a value between -1 and 1. It is similar to the sigmoid function, but it is zero-centered.[8]

Finally, the ReLU function is defined as:

$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

where x is the input to the function. The ReLU function is a simple function that returns 0 if the input is negative, and the input itself if the input is positive.

2.3.5 Optimizers

Optimizers serve as pivotal components in the training of neural networks, enabling the iterative adjustment of network weights to minimize the loss function and enhance overall performance. They play a crucial role in guiding the network's convergence toward optimal solutions. In the realm of CNNs, several optimizers are frequently employed to fine-tune model parameters. Notable among them are the stochastic gradient descent (SGD), Adam, and Adamax optimizers.

SGD is a foundational optimizer that forms the basis for many modern variants. It operates by updating weights in the direction that reduces the loss function. Although simple, SGD can be effective in optimizing neural networks, especially when coupled with proper learning rate schedules.[9]

The Adam optimizer, which stands for Adaptive Moment Estimation, is a powerful and widely adopted optimization algorithm. It combines the benefits of both momentum-based updates and adaptive learning rates. Adam maintains two moving averages for each weight, resulting in efficient and adaptive weight updates. This

[6] "Rectified Linear Units (ReLU) in Deep Learning". Kaggle. September 27 2023. https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning

[7] "Activation Functions in Neural Networks". Towards Data Science. September 27 2023. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[8] "Activation Functions in Neural Networks". Towards Data Science. September 27 2023. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[9] "Stochastic Gradient Descent (SGD)". GeeksForGeeks. September 27 2023. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

optimizer excels in handling non-stationary or noisy objective functions, making it a popular choice for training CNNs.[10]

Adamax is an extension of the Adam optimizer that offers certain advantages in terms of computational efficiency. In our experiments, we found that Adamax yielded better results than Adam, which is why we ended up using it.

3 Data Preparation

3.1 Source

The data used for this dissertation is a combination of the Sloan Digital Sky Survey Data Release 16 (SDSS DR16) and the Gaia Data Release 3 (DR3) datasets. Combined, they form a dataset of galaxies with known redshifts and spectra.

3.2 Composition

The dataset consists of 520.000 galaxy spectra, each with 186 data points. Each data point is the flux at a specific wavelength, ranging from 366 to 996 nanometers. The dataset also contains the redshift of each galaxy, ranging from 0 to 0.6 z which is the target variable that we aim to predict.

Here is a sample of the data (the first number is the redshift and the rest are the flux values at different wavelengths):

$$0.1735581, 0.539, 0.514, 0.439, 0.34, 0.236, 0.134, 0.034...$$

3.3 Preprocessing

The only preprocessing step that was performed was to to apply a min-max normalization to the flux values, which rescales them to the range $[0, 1]$. Min-max normalization is a common practice in machine learning, particularly for neural networks. It standardizes the data, ensuring that all features are on the same scale. This not only aids in model convergence but also enhances training stability and performance.[11] It is mathematically defined as:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x is the original data and x_{min} and x_{max} are the minimum and maximum values of x, respectively.

We first loaded the data into a Pandas dataframe, then converted it to a numpy array, and finally used the `MinMaxScaler` class from the `sklearn.preprocessing` module to perform the normalization:

```
data = pd.read_csv('./training_data.csv')
spectra = np.array(data.iloc[:, 1:])
redshifts = np.array(data.iloc[:, 0])
```

[10] "Adam: A Method for Stochastic Optimization". arXiv. September 27 2023. https://arxiv.org/abs/1412.6980

[11] "Why Data should be Normalized before Training a Neural Network". Towards Data Science. September 26 2023. https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d

```
min_max_scaler = MinMaxScaler()
spectra = min_max_scaler.fit_transform(spectra)
```

3.4 Splitting

The dataset was split into a training set, a validation set, and a test set. The training set contained 90% of the data, while the test set contained the remaining 10%. The training set was used to train the neural network, while the test set was used to evaluate the performance of the trained model.

Of the training set, 30% was used as a validation set, which was used to evaluate the model during training and tune the model's hyperparameters. Hyperparameters are parameters that are set before training and affect the training process, such as the learning rate and batch size.

We use the `train_test_split` function from the `sklearn.model_selection` module to split the data:

```
splitted_data = train_test_split(spectra, redshifts, test_size=0.1)
```

4 Methodology

4.1 Network architecture

The Convolutional Neural Network (CNN) architecture chosen for this task is specifically tailored to handle the spectral data obtained from the Gaia mission. Unlike traditional image-based CNNs, which analyze two-dimensional images, this architecture is designed to work with one-dimensional spectra.

Input Layer: The input layer accepts the normalized galaxy spectra. Each spectrum, represented as a 1D array of flux values at different wavelengths, is fed into the network. The input layer's size is determined by the number of data points in each spectrum.

Convolutional Layers: These layers are responsible for feature extraction. Convolutional filters slide over the input spectra, capturing local patterns and features at different scales. Given the varying nature of spectral data, multiple convolutional layers with different filter sizes are used to capture both fine-grained and broader features. This adaptability is essential since galaxy spectra can exhibit a wide range of shapes and characteristics.

Pooling Layers: After each convolutional layer, a pooling layer follows. Max-pooling is applied to reduce the spatial dimensionality of the feature maps, retaining the most important features. Pooling helps in managing the computational complexity of the network while preserving crucial information.

Fully Connected (Dense) Layers: Following the convolutional and pooling layers, fully connected layers are used for the final prediction. These layers gradually reduce the dimensionality of the features learned by previous layers, ultimately yielding a single numeric value: the predicted galaxy redshift.

Output Layer: The output layer consists of a single neuron with a linear activation function. This neuron's output represents the predicted redshift of the observed galaxy. During training, this prediction is compared to the actual redshift to compute the loss, which guides the network's weight adjustments.

Below is the Keras code for the architecture. We used 4 convolutional layers each followed by a max-pooling layer, followed by 4 fully connected layers, and 1 output layer. The number of filters in each convolutional layer was 256, 256, 128, and 64, respectively. The kernel size of each convolutional layer was 5, 5, 3, and 2, respectively. The number of neurons in each fully connected layer was 256, 256, 128, and 64, respectively. We used the ReLU activation function for all layers except the output layer, which used a linear activation function. The decision to use these specific hyperparameters was based on known common practices and extensive experimentation.

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(186, 1)),
    tf.keras.layers.Conv1D(filters=256, kernel_size=5, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(filters=256, kernel_size=5, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(filters=128, kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(filters=64, kernel_size=2, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='linear')
])
```

4.2 Training

4.2.1 Batch size

The batch size constitutes a pivotal hyperparameter in the training process, influencing how the network learns from the data. It signifies the number of data samples that are processed in a single forward and backward pass during each training iteration. The choice of batch size carries significant implications for the network's training dynamics.

Utilizing a larger batch size can expedite the training process, as more samples are processed in parallel. This can lead to faster convergence, especially on hardware optimized for parallel computations. However, there is a trade-off, as larger batch sizes can increase the risk of overfitting. The network might memorize the training data rather than learning to generalize from it.

Conversely, a smaller batch size entails processing fewer samples at once. This can result in slower training progress, particularly on hardware with limited parallelism. However, smaller batch sizes often lead to better generalization, as the network receives a more diverse set of samples during training. It is less likely to memorize the training data and is more likely to extract meaningful patterns.[12]

[12] "How to Control the Stability of Training Neural Networks With the Batch Size". Machine Learning Mastery. September 27 2023. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

In our experiments, we fine-tuned the batch size and discovered that a setting of 16 yielded the optimal balance between training efficiency and generalization performance.

4.2.2 Epochs

Epochs refer to the number of times the entire training dataset is processed by the neural network. Each epoch represents a complete cycle through the dataset, during which the network updates its weights based on the observed errors. The choice of the number of epochs is another vital training hyperparameter.

Training for too few epochs may result in an underfit model. In this scenario, the network has not had sufficient exposure to the data to learn complex patterns, and it may not perform well on unseen data.

Conversely, training for an excessive number of epochs can lead to overfitting. The network may become too specialized in capturing the idiosyncrasies of the training data, diminishing its ability to generalize.

Determining the ideal number of epochs involves a balance between achieving convergence and avoiding overfitting. Typically, researchers employ techniques like early stopping, which monitors validation performance and halts training when it starts to degrade, to guide epoch selection.

In our experiments, we found through early stopping that training for 20 epochs yielded satisfactory results.
