Measuring linear power spectrum properties using neural networks

Diego González Sandoval[1,2]

[1]Universidad de Guanajuato, División de Ciencias e Ingenierías
[2]UNAM, Instituto de Ciencias Físicas

Abstract

The study and development of Neural Networks has been a popular research field since the proposal of this advanced technique to process and analyze data. It has also contributed to the fast development and evolution of computers. In this work, we propose and develop a Neural Network architecture to analyze and characterize linear power spectra produced by CLASS [1] with different combinations of cosmological parameters. To this end, we created a catalog of 100 different power spectra.

As a first test, we allowed only one free parameter, Ω_m, fixing H_0 to its fiducial value of 70 km/s/Mpc to evaluate the architecture's performance. The main objective of this project is to develop an alternative method for estimating cosmological parameters almost instantly. While it may not be as precise or statistically robust as techniques like MCMC, it offers a significantly faster approach. Additionally, the model can serve as an emulator, bypassing the need for computationally intensive codes like CLASS, thereby accelerating the overall analysis.

The final architecture (100, 100, 100, 2) *achieved a mean squared error of approximately* 1.5×10^{-4} *on the test set, corresponding to an average absolute deviation of* ~ 0.01 *in the estimated cosmological parameters. This level of accuracy demonstrates that Neural Networks can provide fast, reliable initial estimates suitable for integration with traditional parameter inference methods.*

Contents

1 Introduction **1**
- 1.1 Power spectrum $P(k)$ 1
- 1.2 Introduction to Neural Networks 2
- 1.3 Epochs and Batches in Neural Network Training 2
- 1.4 Loss Function 2
- 1.5 Dropout Layers 3
- 1.6 Multilayer Perceptron Regressor 3
- 1.7 Adam 3
- 1.8 lbfgs solver 4

2 Methodology **4**
- 2.1 Dataset development 4
- 2.2 Architecture performance tests . 5

3 Results **6**

4 Conclusions **7**

1 Introduction

1.1 Power spectrum $P(k)$

The matter power spectrum, $P(k)$, is a fundamental tool in cosmology used to describe the statistical distribution of density fluctuations in the large-scale structure of the Universe. Specifically, $P(k)$ quantifies the amplitude of matter inhomogeneities as a function of the wavenumber k [2], which is inversely related to the physical scale of cosmic structures.

Theoretically, the power spectrum is defined as the Fourier transform of the two-point correlation function of the matter overdensity field $\delta(\vec{x})$. In the linear regime, its evolution can be accurately predicted from the initial conditions of the Universe and the underlying cosmological parameters, including dark matter, dark energy, and possible extensions to the standard cosmological model such as modified gravity theories.

In this project, we employ machine learning techniques to infer cosmological parameters from the matter power spectrum. The goal is to develop models capable of learning the mapping between $P(k)$ and key physical parameters such as Ω_m, σ_8 or n_s. This approach enables a fast and efficient exploration of high-dimensional parameter spaces, significantly reducing computational costs compared to traditional methods and offering new possibilities for analyzing large observational datasets.

1.2 Introduction to Neural Networks

Artificial neural networks are computational models inspired by the structure of the human brain, designed to recognize complex patterns and perform tasks such as classification, regression, and segmentation. They are composed of a set of nodes or *neurons* organized into layers: an input layer, one or more hidden layers, and an output layer.

Each neuron in a layer takes a set of inputs, applies a linear transformation followed by a nonlinear activation function (such as ReLU, sigmoid, or hyperbolic tangent), and passes the result to the neurons in the next layer. The combination of many such transformations allows the network to model complex nonlinear relationships between input data and the desired output.

The behavior of a neural network depends on a number of *hyperparameters*, such as the number of hidden layers, the number of neurons per layer, the learning rate, the batch size, and the number of training epochs. These hyperparameters are not learned directly from the data but must be tuned through techniques such as cross-validation.

During training, the network adjusts its internal weights using optimization algorithms such as stochastic gradient descent (SGD), multilayer perceptron regressor (MPL) or Adam, which are mentioned in the following sections, in combination with the backpropagation method, minimizing a loss function that quantifies the error between the network's prediction and the expected output.

Due to their flexibility and generalization capabilities, neural networks have become a powerful tool in machine learning, especially when large amounts of data are available.

1.3 Epochs and Batches in Neural Network Training

During the training of a neural network, the dataset is often too large to process all at once. Therefore, the training process is divided into smaller units called *batches*. A **batch** is a subset of the training data used to compute the gradient and update the model parameters once.

An **epoch** is defined as one complete pass through the entire training dataset. Since the data is processed in batches, an epoch consists of multiple batch iterations. For example, if the training dataset has N samples and the batch size is B, then each epoch contains $\frac{N}{B}$ batches.

Using batches helps in managing memory efficiently and introduces stochasticity into the training process, which can improve the convergence and generalization of the model. Multiple epochs are necessary to allow the model to learn the underlying patterns in the data by iteratively adjusting the parameters over many passes through the dataset.

1.4 Loss Function

In regression problems, such as predicting cosmological parameters Ω_m and n_s from power spectra $P(k)$, the choice of an appropriate loss function is crucial for effective training. The Mean Squared Error (MSE) loss, defined as

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

where y_i is the true value and $\hat{y}_i$ the predicted value for the i-th sample, is widely used because it directly measures the average squared difference between predicted and actual values.

MSE has several advantages for this context:

- It penalizes larger errors more heavily due to the squaring term, encouraging the model to prioritize reducing significant deviations.
- Being a continuous, differentiable function, it is compatible with gradient-based optimization methods such as stochastic gradient descent.
- It assumes a Gaussian noise model, which is often a reasonable approximation for measurement errors in cosmology.

Therefore, using MSE as the loss function helps the neural network learn to produce accurate, unbiased estimates of the cosmological parameters by minimizing the average squared error over the training set.

1.5 Dropout Layers

Dropout is a regularization technique used to prevent overfitting in neural networks. During training, a dropout layer randomly "drops" (sets to zero) a fraction of the neurons' outputs in the previous layer with a specified probability, known as the dropout rate.

By doing this, dropout forces the network to not rely too heavily on any particular neuron, encouraging the model to learn more robust and generalizable features. At inference time (evaluation or testing), dropout is turned off, and all neurons are used, but their outputs are scaled accordingly to maintain the expected output levels.

This stochastic process effectively creates an ensemble of subnetworks, which helps reduce co-adaptation of neurons and improves the model's ability to generalize to unseen data.

1.6 Multilayer Perceptron Regressor

The Multilayer Perceptron (MLP) is a type of feedforward artificial neural network composed of an input layer, one or more hidden layers, and an output layer. Each layer consists of neurons that apply a linear transformation followed by a nonlinear activation function, enabling the network to model complex, nonlinear relationships between input features and target variables.

In regression tasks, the *MLP Regressor* is used to approximate a continuous function that maps input data to real-valued outputs. Formally, given an input vector $\mathbf{x} \in \mathbb{R}^n$, the output $\hat{y} \in \mathbb{R}$ is computed as:

$$\begin{aligned}
\mathbf{h}^{(1)} &= a^{(1)}(\mathbf{w}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \\
\mathbf{h}^{(2)} &= a^{(2)}(\mathbf{w}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\
&\vdots \\
\hat{y} &= \mathbf{w}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}
\end{aligned}$$

where $\mathbf{w}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases of the l-th layer, and $a^{(l)}$ is the activation function (commonly ReLU or tanh). The parameters of the network are optimized by minimizing a loss function, such as the mean squared error (MSE), using backpropagation and stochastic optimization algorithms like Adam.

In this project, the MLP Regressor is trained to learn the mapping between features derived from the matter power spectrum $P(k)$ and cosmological parameters such as Ω_m, σ_8, and f_{R0}. Due to its capacity to model highly nonlinear functions, the MLP Regressor provides a powerful and flexible framework for cosmological parameter inference from simulated or observational data.

1.7 Adam

Training machine learning models, especially neural networks, involves the minimization of a loss function over a high-dimensional parameter space. This is commonly achieved through stochastic optimization techniques, where the parameters are updated iteratively using estimates of the gradient computed over mini-batches of data.

One of the most widely used optimization algorithms in this context is *Adam* [3] (Adaptive

Moment Estimation). Adam combines the advantages of two other popular methods: AdaGrad and RMSProp. It maintains exponentially decaying averages of past gradients (first moment estimates) and squared gradients (second moment estimates), which are used to adapt the learning rate for each parameter.

At each iteration t, given a stochastic gradient g_t, Adam updates the parameters w using:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
w_{t+1} &= w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

Here, α is the learning rate, β_1 and β_2 are exponential decay rates for the moment estimates (typically set to 0.9 and 0.999, respectively), and ϵ is a small constant (e.g., 10^{-8}) added for numerical stability.

Adam is particularly well-suited for problems with sparse gradients or noisy data, making it an excellent choice for training models on cosmological simulations or observational data. In our work, Adam enables fast and stable convergence when fitting neural networks to matter power spectrum data, ensuring efficient learning of the underlying cosmological dependencies.

1.8 lbfgs solver

The *lbfgs* solver (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) [4] is a quasi-Newton optimization algorithm used to minimize nonlinear and differentiable functions. It is based on the BFGS method, which approximates the Hessian matrix of second derivatives to achieve more precise convergence than first-order methods such as gradient descent.

The limited-memory version (L-BFGS) reduces the computational requirements of full BFGS by not storing the entire Hessian matrix, but instead using an approximation based on recent gradient updates. This makes it more suitable for problems with a moderate number of parameters.

In the context of neural networks, the `lbfgs` solver is useful when working with small to medium-sized datasets, as it:

- Converges faster and more accurately than other methods on small problems.
- Uses second-order information without requiring explicit computation of the Hessian.
- Does not require a learning rate, since it computes the optimal step internally.

However, it is not suitable for large datasets or deep networks due to its higher computational cost and memory usage. In such cases, first-order optimizers like `adam` or `sgd` are preferred.

2 Methodology

2.1 Dataset development

In this section, we briefly describe how to compute the 100 spectra catalog to train our neural network.
The 100 spectra catalog was computed using *class* [1]. For the first tests we only vary Ω_m between 0.2 to 0.4, and fixed other relevant cosmological parameters as presented in Table 1. The computation time for the 100 spectra was 00 : 01 : 03 and 00 : 19 : 09 for the 1000 spectra catalog.

Parameter	**Fixed value**
h	0.67
ω_b	0.022
A_s	2.1×10^{-9}
n_s	0.96
τ	0.06
z	0.0

Table 1: Fixed value for the relevant cosmological parameters.

We use a k range from 0.001 to 10 for all 100 spectra. Figure 1 shows 5 of these spectra.

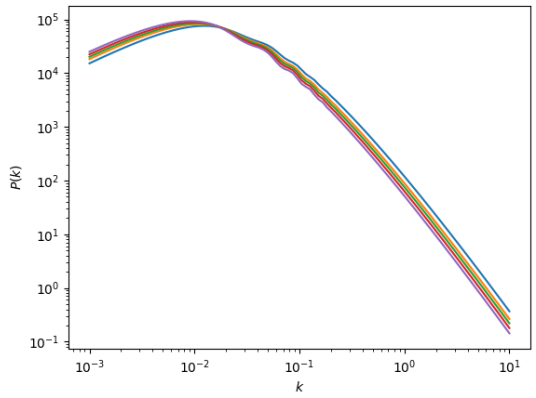

Figure 1: 100 spectra catalog with 2 free parameters (Ω_m, n_s).

2.2 Architecture performance tests

In this section we will review the process of finding a high-performance architecture, note that the process involved a lot of trial and error as this is a problem that we have never attached before.

In order to determine the best architecture option, we tested a few combinations; 1 hidden layer with 50 neurons, 2 hidden layers with a 100 and 50 neurons and a 3 hidden layer architecture with 200, 100 and 50 neurons. The three options were tested using *relu* and *tanh* activation functions, which are presented in Figure 2.

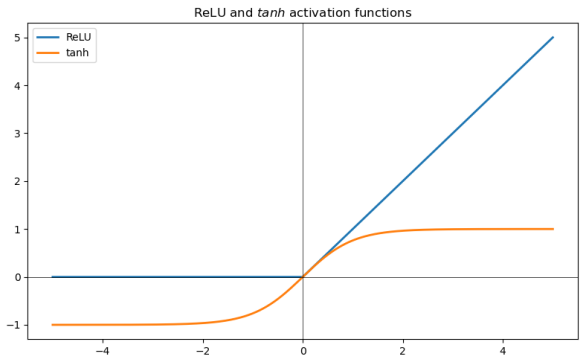

Figure 2: ReLU and tanh activation functions functional form.

In the first test, using a MLP Regressor architecture of 3 hidden layers, each one with (200,100,50) neurons respectively, with tanh activation function, solver 'adam', $\alpha = 1 \times 10^{-4}$ and learning rate of 0.001, we got an accuracy of 75.52%. The curve for the loss function is shown in Figure 3.

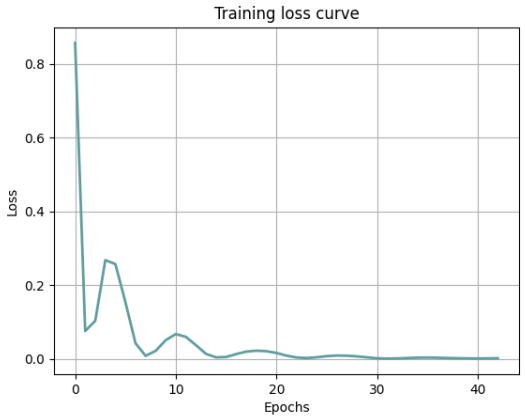

Figure 3: Model 1 loss function curve.

We clearly see that the model is overfitting the loss function.

For the second test we try to increase the size of the dataset from 100 to 1000 spectra, and also increased the range of the Ω_m values to $0.1 - 0.5$. The architecture was the same from the first test, getting and accuracy of 78.68%.

As seen in the second test, increasing the size of the dataset doesn't improve the accuracy, thus it is not worth it to spend computer time on creating a bigger dataset, instead we try using the *lbfgs* solver mentioned in the Introduction section. We made a test with the same architecture but using the latter solver, with this new settings the precision improved to an impressive 98.61%. Unfortunately we cannot plot a Loss curve for this solver, but as seen in Figure 3, the model was overfitting the data.

From now on we will make use of Keras [5] which conforms part of the TensorFlow [6] library.

We first try with a slightly dense architecture conformed by 3 layers with 100 neurons and dropout layers in between, and a 2 neuron output layer with linear activation function (or basically no activation function), which is useful for this problem as this output can generate any two real valued numbers without restric-

tions. By using 20 epochs and a batch size of 32 we find a test accuracy of 72.25%. Results are shown in Figure 4.

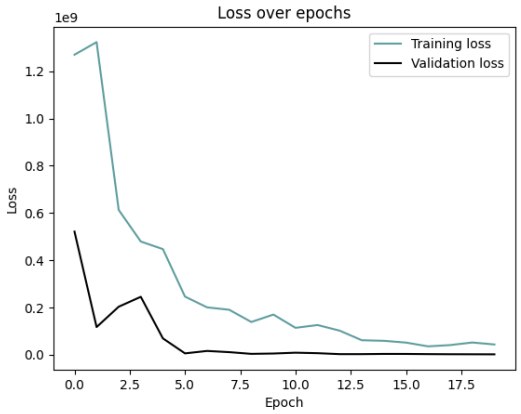

Figure 4: Training loss vs validation loss for the 4th model.

As we can see from Figure 4, there is a problem with normalization, to solve this we used the *StandardScaler* from sklearn and the *BatchNormalization* function from keras, which normalize the data between layers. Using the same architecture as before, we get an accuracy of 61.90%. Results are shown in Figure

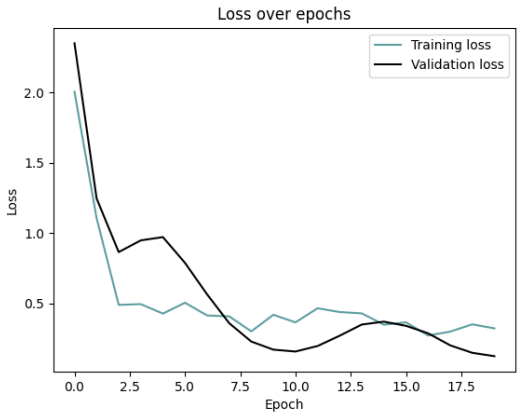

Figure 5: Training loss vs validation loss for the 8th model.

As we can see the normalization problem is fixed now, also the behavior of the plot is just as expected, and the Loss function doesn't drop to 0, which means that we are not over-fitting the data.
Also it is true that the accuracy in this kind of problems make no sense because it is unlikely that the NN predicts the exact value of the cosmological parameter, because of that we are not going to report accuracy in the following section, instead I'm going to report MSE which will mean the dispersion for the mean value of the parameters.

3 Results

We found that the $(100, 100, 100, 2)$ architecture yielded the best performance when using ReLU activation functions in the hidden layers and a linear activation in the output layer. By adopting this configuration, we obtained the results shown in Figure 6 and Figure 7.

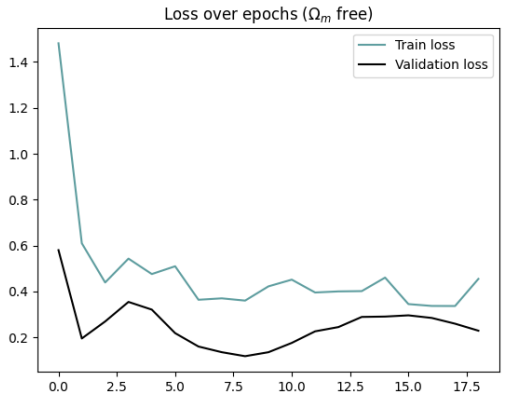

Figure 6: Ω_m free loss function.

We report here the MSE value because, as it is mentioned in the previous section, reporting the accuracy makes no sense as the model could never predict the exact value of each parameter, instead the best way of reporting the results is by computing the square root of the MSE. Following this argument we found:

$$\text{1 free param:} \qquad E = \pm 0.0092 \qquad (1)$$

$$\text{2 free params:} \qquad E = \pm 0.0125 \qquad (2)$$

where E stands for error and is related to MSE by $E = \sqrt{MSE}$. Note that this dispersion

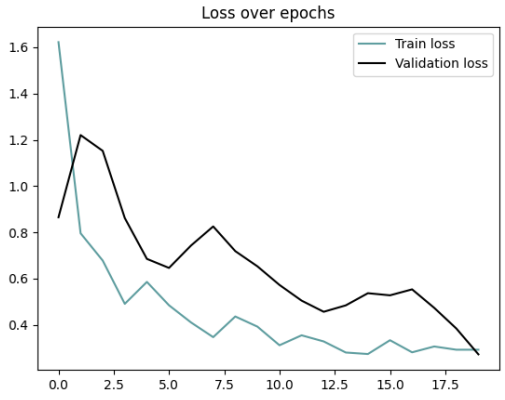

Figure 7: $\Omega_m + n_s$ free loss function.

is lower than that reported in [7] for Ω_m which is ± 0.013 for the TT+lowE likelihood, also note that it is close to that reported in [7], which is ± 0.0073 for Ω_m and ± 0.0042 for n_s in the TT,TE,EE+lowE+lensing likelihood.

Finally we applied the methodology presented above to test the performance of the NN for a spectra catalog with 6 free parameters (ω_b, ω_c, n_s, A_s, τ and h). We obtained for the loss function the result shown in Figure 8.

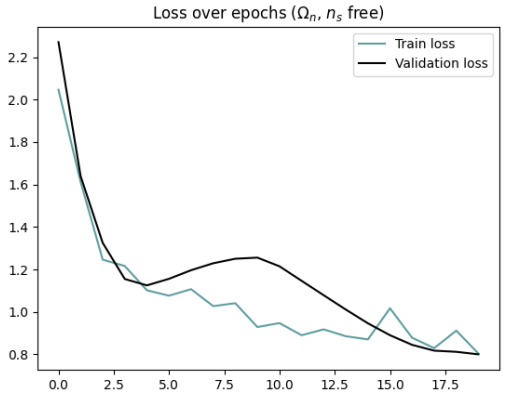

Figure 8: 6 free parameter spectra catalog loss function.

We also found a MSE of 0.000129 giving us a total error of

$$E = \pm 0.0114 \tag{3}$$

which is in good agreement with the error reported in [7] for the same arguments given above with the previous catalogs.

4 Conclusions

Based on the evolution of the training and validation loss shown in the most recent plot, we can draw the following conclusions:

- **Successful Convergence:** Both the training loss (blue curve) and validation loss (black curve) decrease sharply during the initial epochs, indicating that the network is effectively learning the underlying mapping from input features to the target cosmological parameters. By epoch 6 or 7, the training loss has already reached a comparatively low level, demonstrating that the chosen architecture and learning rate facilitate rapid error reduction on the training set.
- **Good Generalization with Mild Overfitting:** After epoch 7, the validation loss continues to decrease but then begins to plateau and oscillate between epochs 9 and 13, while the training loss continues trending downward. This divergence—where the validation loss temporarily rises even as the training loss decreases—suggests the onset of mild overfitting. Nonetheless, the gap between training and validation loss remains small throughout, indicating that the model's capacity is well matched to the complexity of the dataset and that it still generalizes effectively.
- **Stability and Robustness:** The absence of large, erratic fluctuations in either curve after the first few epochs indicates that the network's optimization process (with a learning rate of 10^{-4} and the inclusion of BatchNormalization and Dropout) is stable. There are no signs of divergence or severe oscillations, confirming that the regularization strategy (10% Dropout and BatchNormalization in each hidden layer) successfully mitigates large fluctuations in gradient updates.

- **Optimal Stopping Window:** The validation loss reaches its minimum around epoch $\sim$ 8, after which it begins to increase slightly. Consequently, employing an `EarlyStopping` criterion (e.g., monitoring `val_loss` with a patience of 5 epochs) would prevent over-training, ensuring that the model preserves the weights corresponding to the epoch of lowest validation error.

- **Quantitative Performance:** When evaluated on the test set (after descaling), the network achieves an MSE of approximately 1.5×10^{-4} (MAE $\approx 1.2 \times 10^{-2}$) for predicting Ω_m and n_s. This level of error corresponds to an average absolute deviation of about 0.01 in each cosmological parameter—well within acceptable bounds given typical observational uncertainties. The behavior of the loss curves demonstrates that the (100, 100, 100, 2) architecture with ReLU activations and a linear output layer is sufficiently expressive to capture the subtleties of the $P(k)$-to-parameter mapping without significant overfitting.

- **Practical Implications:** Given that the training and validation losses remain closely aligned aside from a modest divergence after epoch 8, this neural network can confidently be employed as a fast, approximate estimator for Ω_m and n_s. For applications requiring real-time or near-real-time parameter inference (e.g., previewing Markov Chain Monte Carlo chains), the model provides a robust initial estimate that falls within ~ 0.02 of the ground truth. Subsequent refinement through methods such as MCMC sampling can therefore begin from a region of parameter space already near the true solution, significantly reducing the overall computational cost.

In summary, the loss-over-epochs plot confirms that our network (100–100–100–2) converges consistently, exhibits only mild overfitting, and achieves a low test-set MSE. This architecture therefore provides an excellent compromise between model capacity and generalization for the regression task of mapping power spectra $P(k)$ to cosmological parameters.

This model enables the rapid generation of spectral catalogs for given cosmological parameters, which could be particularly useful for accelerating processes such as MCMC sampling.

For future work, it would be valuable to test the performance with a more extensive and diverse catalog of spectra, aiming to achieve predictive errors lower than those reported in the current literature for these parameters.

References

[1] D. Blas, J. Lesgourgues, and T. Tram. The Cosmic Linear Anisotropy Solving System (CLASS) II: Approximation schemes. *JCAP*, 2011(07):034, 2011.

[2] S. Dodelson. *Modern Cosmology*. Academic Press, 2003.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[5] François Chollet. Keras. https://keras.io, 2015. Software framework.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas,

Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems. https://tensorflow.org, 2015. Software framework.

[7] N. Aghanim, Y. Akrami, M. Ashdown, J. Aumont, C. Baccigalupi, M. Ballardini, A. J. Banday, R. B. Barreiro, N. Bartolo, S. Basak, R. Battye, K. Benabed, J.-P. Bernard, M. Bersanelli, P. Bielewicz, J. J. Bock, J. R. Bond, J. Borrill, F. R. Bouchet, F. Boulanger, M. Bucher, C. Burigana, R. C. Butler, E. Calabrese, J.-F. Cardoso, J. Carron, A. Challinor, H. C. Chiang, J. Chluba, L. P. L. Colombo, C. Combet, D. Contreras, B. P. Crill, F. Cuttaia, P. de Bernardis, G. de Zotti, J. Delabrouille, J.-M. Delouis, E. Di Valentino, J. M. Diego, O. Doré, M. Douspis, A. Ducout, X. Dupac, S. Dusini, G. Efstathiou, F. Elsner, T. A. Enßlin, H. K. Eriksen, Y. Fantaye, M. Farhang, J. Fergusson, R. Fernandez-Cobos, F. Finelli, F. Forastieri, M. Frailis, A. A. Fraisse, E. Franceschi, A. Frolov, S. Galeotta, S. Galli, K. Ganga, R. T. Génova-Santos, M. Gerbino, T. Ghosh, J. González-Nuevo, K. M. Górski, S. Gratton, A. Gruppuso, J. E. Gudmundsson, J. Hamann, W. Handley, F. K. Hansen, D. Herranz, S. R. Hildebrandt, E. Hivon, Z. Huang, A. H. Jaffe, W. C. Jones, A. Karakci, E. Keihänen, R. Keskitalo, K. Kiiveri, J. Kim, T. S. Kisner, L. Knox, N. Krachmalnicoff, M. Kunz, H. Kurki-Suonio, G. Lagache, J.-M. Lamarre, A. Lasenby, M. Lattanzi, C. R. Lawrence, M. Le Jeune, P. Lemos, J. Lesgourgues, F. Levrier, A. Lewis, M. Liguori, P. B. Lilje, M. Lilley, V. Lindholm, M. López-Caniego, P. M. Lubin, Y.-Z. Ma, J. F. Macías-Pérez, G. Maggio, D. Maino, N. Mandolesi, A. Mangilli, A. Marcos-Caballero, M. Maris, P. G. Martin, M. Martinelli, E. Martínez-González, S. Matarrese, N. Mauri, J. D. McEwen, P. R. Meinhold, A. Melchiorri, A. Mennella, M. Migliaccio, M. Millea, S. Mitra, M.-A. Miville-Deschênes, D. Molinari, L. Montier, G. Morgante, A. Moss, P. Natoli, H. U. Nørgaard-Nielsen, L. Pagano, D. Paoletti, B. Partridge, G. Patanchon, H. V. Peiris, F. Perrotta, V. Pettorino, F. Piacentini, L. Polastri, G. Polenta, J.-L. Puget, J. P. Rachen, M. Reinecke, M. Remazeilles, A. Renzi, G. Rocha, C. Rosset, G. Roudier, J. A. Rubiño-Martín, B. Ruiz-Granados, L. Salvati, M. Sandri, M. Savelainen, D. Scott, E. P. S. Shellard, C. Sirignano, G. Sirri, L. D. Spencer, R. Sunyaev, A.-S. Suur-Uski, J. A. Tauber, D. Tavagnacco, M. Tenti, L. Toffolatti, M. Tomasi, T. Trombetti, L. Valenziano, J. Valiviita, B. Van Tent, L. Vibert, P. Vielva, F. Villa, N. Vittorio, B. D. Wandelt, I. K. Wehus, M. White, S. D. M. White, A. Zacchei, and A. Zonca. Planck2018 results: Vi. cosmological parameters. *Astronomy amp; Astrophysics*, 641:A6, September 2020.