MNRAS **000**, 1–17 (2018) Preprint 7 November 2018 Compiled using MNRAS LATEX style file v3.0

Evaluating machine learning techniques for predicting power spectra from reionization simulations

W. D. Jennings [1], C. A. Watkinson [2], F. B. Abdalla [1], J. D. McEwen [3]
[1] *Department of Physics & Astronomy, University College London, Gower Street, London WC1E 6BT, UK*
[2] *Blackett Laboratory, Imperial College, London, SW7 2AZ, UK*
[3] *Mullard Space Science Laboratory, University College London, Surrey RH5 6NT, UK*

Last updated 2018 September 7; in original form 2018 September 7

ABSTRACT

Upcoming experiments such as the SKA will provide huge quantities of data. Fast modelling of the high-redshift 21cm signal will be crucial for efficiently comparing these data sets with theory. The most detailed theoretical predictions currently come from numerical simulations and from faster but less accurate semi-numerical simulations. Recently, machine learning techniques have been proposed to emulate the behaviour of these semi-numerical simulations with drastically reduced time and computing cost. We compare the viability of five such machine learning techniques for emulating the 21cm power spectrum of the publicly-available code *SimFast21*. Our best emulator is a multilayer perceptron with three hidden layers, reproducing *SimFast21* power spectra 10^8 times faster than the simulation with 4% mean squared error averaged across all redshifts and input parameters. The other techniques (interpolation, Gaussian processes regression, and support vector machine) have slower prediction times and worse prediction accuracy than the multilayer perceptron. All our emulators can make predictions at any redshift and scale, which gives more flexible predictions but results in significantly worse prediction accuracy at lower redshifts. We then present a proof-of-concept technique for mapping between two different simulations, exploiting our best emulator's fast prediction speed. We demonstrate this technique to find a mapping between *SimFast21* and another publicly-available code *21cmFAST*. We observe a noticeable offset between the simulations for some regions of the input space. Such techniques could potentially be used as a bridge between fast semi-numerical simulations and accurate numerical radiative transfer simulations.

Key words: cosmology - dark ages, reionization - statistical methods

1 INTRODUCTION

The Dark Ages of the Universe ended when the first stars and galaxies begin to form. Radiation from these sources ionized the surrounding matter, eventually giving rise to bubbles of ionized hydrogen. The size, shape and clustering properties of these bubbles contain valuable information about how our Universe evolved during these otherwise obscure times. Direct observation of these bubbles requires us to distinguish between ionized regions and neutral regions. The most promising probe for this is the 21cm hyperfine transition of hydrogen, which is emitted exclusively by neutral hydrogen during the proton-electron interaction. Measurements of the 21cm signal on the sky give us an image of the neutral hydrogen in the Universe and, by tracing this signal signal back through time, we can extend these images into three-dimensional maps.

Observational difficulties have so far prevented us from creating such three-dimensional maps. The signal is much weaker than other foreground sources at similar frequencies and it is difficult to extract the actual 21cm signal from these foregrounds. Past and ongoing experiments such as Murchison Widefield Array[1] (MWA, Tingay et al. 2013), the Low Frequency Array[2] (LOFAR, Patil et al. 2017), and the Precision Array for Probing the Epoch of Reionization[3] (PAPER, Ali et al. 2015) have begun to place limits on the overall intensity of the signal. Upcoming experiments such as the Hydrogen Epoch of Reionization Array[4] (HERA, DeBoer et al. 2017) and the Square Kilometre Array[5] (SKA, Mellema et al. 2013) will be able to provide more detailed measurements and should allow us to make first parameter constraints for our models.

Theoretical modelling of the 21cm signal involves answering questions about the reionization processes: what were the main sources of ionizing photons; when did reionization start; how long did it last? The most detailed theoretical predictions are

[1] http://www.mwatelescope.org/telescope
[2] http://www.lofar.org/
[3] http://eor.berkeley.edu/
[4] http://reionization.org/
[5] https://www.skatelescope.org/

currently from numerical and semi-numerical simulations. The most likely reionization scenarios can be extracted by comparing such simulations to data, most efficiently by combining fast approximate semi-numerical simulations with sampling methods such as MCMC to perform parameter estimation (Greig & Mesinger 2015, Hassan et al. 2017, Liu et al. 2016, Pober et al. 2016, Greig et al. 2016, Greig & Mesinger 2018). Two semi-numerical simulations are *SimFast21* (Santos et al. 2010) and *21cmFAST* (Mesinger et al. 2011), which generate three-dimensional realisations of the 21cm signal. Much work has gone into finding efficient summary statistics for the simulation outputs. Common summary statistics are the power spectrum and its higher-order equivalent the bispectrum (Shimabukuro et al. 2016, Watkinson et al. 2017, Majumdar et al. 2018 and Watkinson et al. 2018). Both statistics contain information about the clustering properties of ionized hydrogen bubbles.

Semi-numerical simulations take minutes to hours to run. Recently machine learning techniques have been suggested for a number of uses: to emulate power spectrum outputs quickly from *21cmFAST* (Schmit & Pritchard 2018, Kern et al. 2017), to derive reionization parameters directly from the 21cm power spectrum (Shimabukuro & Semelin 2017), and to derive reionization parameters from 21cm images (Gillet et al. 2018). In the first application, models are trained to mimic the outputs that would have resulted from an actual simulation. Training involves running a representative sample of actual simulations and learning to mimic their behaviour. After training, the models can make fast power spectrum predictions at any new input points. The ultimate aim of such an approach would be to train models to mimic the more accurate numerical simulations, allowing for more accurate parameter estimation.

In this paper, we evaluate the viability of five machine learning techniques for emulating the 21cm power spectrum from *SimFast21*. We analyse the prediction speeds of the resulting emulators and their accuracy across the standard reionization input parameter space. The emulators in Schmit & Pritchard 2018 and Kern et al. 2017 were trained at fixed scales and fixed redshifts. Such emulators make predictions only at these fixed scales and redshifts, so that if other scales or redshifts are desired one must interpolate further. We use the scales and redshifts directly as extra inputs to the trained models, so that they learn to make predictions for any requested scale and redshift. This method is theoretically more flexible but gives rise to poorer prediction accuracy at lower redshifts.

We then use our best emulator candidate to present a proof-of-concept technique for determining a relationship between two different simulations. We demonstrate the technique by finding a mapping between the inputs of *SimFast21* and those of *21cmFAST* by measuring which inputs result in the most similar output power spectra. This method could potentially be used to bridge between fast semi-numerical simulations and more accurate three-dimensional radiative transfer codes, see for example C^2-RAY (Mellema et al. 2006) and LICORICE (Semelin et al. 2017; Kulkarni et al. 2016). In our conclusions we comment on the feasibility of using our method for this purpose, both in light of our results and in the context of the known discrepancy between numerical and semi-numerical codes (see for example Majumdar et al. 2014).

The rest of the paper is split in to the following sections. In Section 2 we describe the reionization models used in the simulations. Section 3 contains descriptions of the machine learning techniques we used. In Section 4 we briefly describe the specifics of how our emulators were trained. We present the results of training our emulators in Section 5. Section 6 is a discussion of the accuracy and speed performance of the different machine learning techniques, and how their performance depends on the input parameters. Section 7 contains the proof-of-concept method for mapping between *SimFast21* and modified *21cmFAST*, using our best emulator. We end the paper in Section 8 with our conclusions. For Cosmological parameters, we use $\Omega_{\rm M} = 0.270$, $\Omega_{\rm b} = 0.046$, $\Omega_\Lambda = 0.730$, $H_0 = 71.0 {\rm km\ s^{-1} Mpc^{-1}}$, $n_{\rm s} = 0.960$, $\sigma_8 = 0.810$, the default parameters in the *SimFast21* package[6].

2 MODELS OF REIONIZATION

The 21cm differential brightness temperature $\delta T_{\rm b}$ is defined as the difference between the measured 21cm brightness temperature and the uniform background CMB brightness temperature. By removing the background CMB temperature, the value of $\delta T_{\rm b}(\vec{r})$ then specifies the extent of 21cm emission ($\delta T_{\rm b} > 0$) or absorption ($\delta T_{\rm b} < 0$). The actual observable for radio interferometers is $\delta T_{\rm b} - \langle \delta T_{\rm b} \rangle$, where $\langle \delta T_{\rm b} \rangle$ is the global reionization signal averaged across the whole sky. Furlanetto et al. (2006) gives an approximate relationship for the 21cm brightness temperature $\delta T_{\rm b}(\vec{r})$ as

$$\delta T_{\rm b}(\vec{r}) = 27 x_{\rm HI}(\vec{r}) \left[1 + \delta(\vec{r})\right] \left(\frac{\Omega_{\rm b} h^2}{0.023}\right) \left(\frac{0.15}{\Omega_{\rm M} h^2}\right)^{1/2} \left(1 - \frac{T_\Gamma}{T_{\rm S}}\right) \left(\frac{1+z}{10}\right)^{1/2} \left(\frac{H(z)}{H(z) + \delta_r v_r(\vec{r})}\right) {\rm mK}\,. \tag{1}$$

This approximation includes the effects of neutral hydrogen fraction $x_{\rm HI}(\vec{r})$; total matter density contrast $\delta(\vec{r})$; cosmological parameters for the densities of baryonic matter $\Omega_{\rm b}$ and total matter $\Omega_{\rm M}$; the CMB temperature T_Γ; the spin temperature $T_{\rm S}$ which quantifies the relative populations of neutral hydrogen atoms in the higher and lower energy states; the Hubble parameter $H(z)$; and $\delta_r v_r(\vec{r})$, the radial velocity gradient.

2.1 Power spectrum for $\delta T_{\rm b}$

We train our emulators to reproduce correlations in fluctuations of the differential brightness temperature. Fluctuations in $\delta T_{\rm b}(\vec{r})$ are given by

$$\Delta T_{\rm b}(\vec{r}) = \frac{\delta T_{\rm b}(\vec{r}) - \left\langle \delta T_{\rm b}(\vec{r}) \right\rangle}{\left\langle \delta T_{\rm b}(\vec{r}) \right\rangle}\,, \tag{2}$$

where $\langle \delta T_{\rm b}(\vec{r}) \rangle$ is again the global reionization signal measured across the whole sky. The correlation in these fluctuations is the power spectrum

$$P_{\Delta T_{\rm b}}(\vec{k})\, \delta_D^3(\vec{k} - \vec{k}') = \frac{1}{(2\pi)^3} \left\langle \widetilde{\Delta} T_{\rm b}(\vec{k})\, \widetilde{\Delta} T_{\rm b}^*(\vec{k}') \right\rangle\,. \tag{3}$$

Here, $\widetilde{\Delta} T_{\rm b}(\vec{k})$ is the Fourier transform of $\Delta T_{\rm b}(\vec{r})$, and the angular brackets denote an ensemble average.

[6] https://github.com/mariogrs/Simfast21

2.2 SimFast21

To generate our three-dimensional 21cm maps, we use the publicly available semi-numerical code *SimFast21*[7] (version 1.0). We briefly describe the algorithm here. The simulation begins by seeding an initial linear density field onto a three-dimensional grid at very high redshift. This linear density field is evolved using first-order perturbation theory (see Zel'dovich 1970) giving a non-linear density field $\delta(\vec{r})$.

The simulation then finds the highest density regions where the matter will collapse to form luminous structures and thus contribute ionizing photons towards the reionization process. The extent of collapse is calculated from the non-linear density field in two different ways. For the collapse of the largest and most massive regions, *SimFast21* explicitly resolves individual dark matter halos using an excursion-set formalism (Furlanetto et al. 2004). This method is only used for the collapse of regions larger than a single pixel which means that halos can be resolved down to $5 \times 10^9 M_\odot$ in our simulations. For smaller unresolved regions, *SimFast21* uses the approximate ellipsoidal collapse method from Sheth et al. (2001): if the mean enclosed density in a region exceeds a theoretical critical value then the region is assumed to collapse. The collapse fraction $f_{\rm coll}(\vec{r}, R)$ on decreasing scales R is then found from the contributions of both resolved and unresolved halos. A fixed simulation parameter $M_{\rm min}$ controls the minimum considered mass of collapsing region, since small dark matter halos are generally considered to have very low star formation rates (see Barkana & Loeb 2001 for a review) and can be ignored as not contributing a significant number of ionizing photons.

The ionization fraction field $x_{\rm HII}(\vec{r})$ is found by determining whether the collapsed matter in a region generates enough ionizing photons to ionize the enclosed hydrogen atoms. An ionizing efficiency parameter $\zeta_{\rm ion}$ specifies how many ionizing photons are sourced per unit of collapsed matter. Pixels are painted as fully ionized if $f_{\rm coll}(\vec{r}, R) \geqslant \zeta_{\rm ion}^{-1}$, otherwise they are set as partially ionized according to the collapsed fraction in the cell $\zeta_{\rm ion} f_{\rm coll}(\vec{r}, R)$. Finally, Equation (1) is used to find the 21cm brightness temperature field $\delta T_{\rm b}(\vec{r})$ from the non-linear density field $\delta(\vec{r})$ and the neutral fraction field $x_{\rm HI}(\vec{r}) = 1 - x_{\rm HII}(\vec{r})$.

Three simulation parameters stand out as the most powerful ways to constrain reionization scenarios from data:

(i) The ionization efficiency $\zeta_{\rm ion}$, specifying how many ionising photons are sourced per unit of collapsed matter;
(ii) The maximum bubble size $R_{\rm max}$, specifying the maximum travel distance for ionizing photons from their sources;
(iii) The lower mass limit $M_{\rm min}$, specifying the minimum mass of collapsed matter which produces ionizing photons.

SimFast21 also has the option to account for local fluctuations in the spin temperature, at the expense of considerably more computation time. We turn off this functionality to give a usable training dataset size in a reasonable time frame.

3 MACHINE LEARNING TECHNIQUES

The machine learning techniques in this paper are methods of multi-dimensional regression: learning the behaviour of some function $f(\vec{x})$ from noisy example training data $y_n = f(\vec{x_n}) + \text{Noise}$. The noise in all our data is sample variance from randomly seeding different density fields at the start of each simulation. We do not include instrumental noise because our emulators are intended as efficient replacements for the expensive simulations themselves. For comparison with observed telescope data, instrumental noise can be added in the comparison stage after running the clean emulated simulations. After fitting, the models can make predicted evaluations $f(\vec{x^*})$ at new input values $\vec{x^*}$. This section describes the different machine learning techniques we used along with theoretical descriptions of their specific training methodologies. Each method learns the behaviour of the *SimFast21* power spectrum for any reionization scenario specified by a continuous range of *SimFast21* input parameters. The trained models can then make fast power spectrum predictions for new scenarios, provided the new scenario parameters do not lie far outside the range of our representative training data.

3.1 Interpolation

The simplest method for prediction is to interpolate the power spectrum outputs within the training data. We use two interpolation methods, linear interpolation and nearest-neighbour interpolation, implemented using the classes *LinearNDInterpolator* and *NearestNDInterpolator* from the *scipy* module (Jones et al. 2001). These methods involve no hyperparameter searching and ignore the effect of sample variance noise in the training data. We include them as a naive benchmark to compare the accuracy and speed performance with the other models. The scipy *LinearNDInterpolator* class uses *qhull* from Barber et al. (1996) to triangulate the input data, computing five-dimensional surfaces in the input space and then performing linear interpolation on these triangles. This process takes a long time, both for training and prediction. The scipy *NearestNDInterpolator* class makes predictions by returning the output value from the nearest training data point. This process is very fast but generally results in poorer predictions.

3.2 Multilayer perceptron

An artificial neural network (ANN) represents the function $f(\vec{x_i})$ by manipulating its input values $\vec{x_i}$ through a series of weighted summations and simple function evaluations. This series of repeated operations can be thought of as occurring in a series of layers. The values in the first layer $\vec{h^{(0)}}$ are simply the input values $\vec{x_i}$. The network manipulates the values from one layer $h_j^{(l-1)}$ to the values in the next layer $h_j^{(l)}$ using

$$\vec{h^{(l)}} = h_j^{(l)} = \phi_\theta \left(\sum_{i=1}^{N_i} W_{ij}^{(l)} h_j^{(l-1)} \right) . \quad (4)$$

The values in the l-th layer are a weighted sum over the values in the previous layer, using trainable weight values $W_{ij}^{(l)}$, and are then passed through an activation function $\phi_\theta(x)$. The final layer contains the network's fitted evaluations of the function, $f(\vec{x_i})$. Training the network requires finding the weight values $W_{ij}^{(l)}$ which most closely mimic the function's behaviour.

Multilayer perceptrons (MLPs) are ANNs which contain at least one hidden layer and have a non-linear activation function. Figure 1 shows a schematic of a typical MLP's layer structure. Lines represent the weighted connections between values. Circles represent the neurons which schematically hold the values $h_j^{(l)}$ and

[7] https://github.com/mariogrs/Simfast21

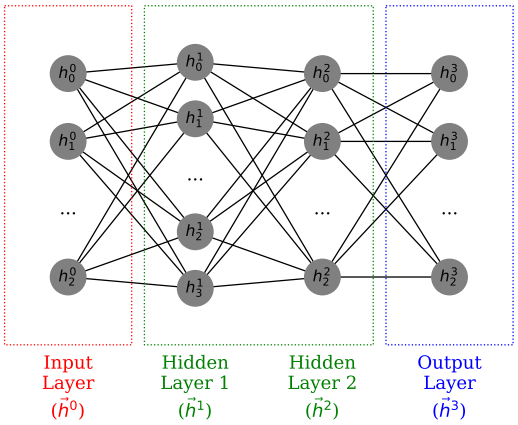


Figure 1. Visualization of a multilayer perceptron with two hidden layers. Lines are weighted connections directed from left to right. Circles are the neurons which schematically hold the values, pass the weighted sum of inputs through the activation function, and send this final value to the next layer.

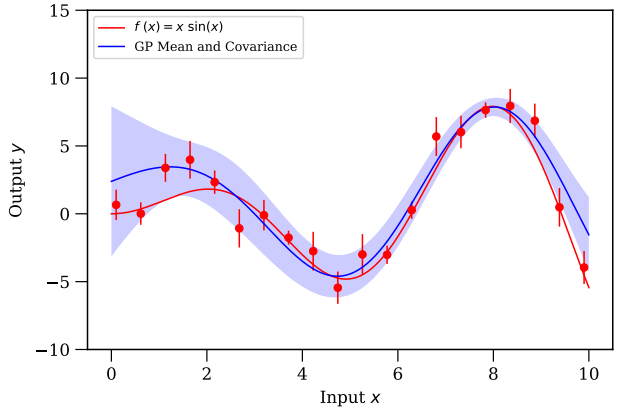


Figure 2. Example of Gaussian process regression on noisy data $y_n = x_n \sin(x_n) + \mathcal{N}(0, \epsilon_n)$, with the noise amplitude on each data point ϵ_n being randomly drawn randomly from the interval [0.5, 1.5]. The mean function (solid blue line) and covariance kernel (shaded blue region) are found which best match the training data (red points).

pass the weighted inputs through the activation function. We use the *scikit-learn* package from Pedregosa et al. (2011) for all our MLPs, using the following default inputs: a constant learning rate of 0.001; batches of size 200; the rectified linear unit function ('relu') as our activation function.

MLP training involves finding the weight values $W_{ij}^{(l)}$ which minimize the objective function,

$$\text{MLP Objective} = \frac{1}{2N}\sum_{n=1}^{N}\left(f(\vec{x_n}) - y_n\right)^2 - \frac{\alpha}{2}\sum_{i,j,l}\left(W_{ij}^{(l)}\right)^2 \quad (5)$$

for training data $(\vec{x_n}, y_n)$. The weights are initialised using a different random seed for each model. The function evaluation $f(\vec{x_n})$ in Equation (5) follows the procedure given in the previous subsection: passing the input values $\vec{x_n}$ through multiple layers of weighted sums and activation function evaluations. Before training, one must fix the number of hidden layers and the number of neurons in each hidden layer. We use a fixed L2 regularization parameter value of $\alpha = 0.0001$ to reduce the effect of overfitting. The *scikit-learn* class for MLP uses backpropagation algorithm Werbos (1982) for efficient calculation of the gradient of the objective function, see Rumelhart et al. (1986) for a more detailed description of this algorithm. We use the 'adam' optimization method (Kingma & Ba 2014) which terminates when the objective function falls below a tolerance of 10^{-10} for at least two consecutive iterations.

3.3 Gaussian processes regression

Gaussian process regression (GPR) is a fitting process for a function whose values are drawn from a Gaussian process. A Gaussian process is a set of random variables, any subset of which follow a jointly multi-variate Gaussian. For a finite set of D random variables stored in a vector $\vec{f} = [f_1, \ldots, f_D]$, the probability density function $P(\vec{f})$ of a multi-variate Gaussian has the form

$$\log P(\vec{f}) = -\frac{1}{2}\sum_{i,j=1}^{D}\left(f_i - \mu_i\right)K_{ij}\left(f_j - \mu_j\right) + \text{constant}. \quad (6)$$

Fitting this finite distribution involves finding the elements $\vec{\mu} = [\mu_1, \ldots, \mu_D]$ of the mean vector, and the elements K_{ij} of the covariance matrix. A Gaussian process extends the concept of a multi-variate Gaussian to infinite dimensions, by replacing the finite-dimensional forms $[\vec{f}, \vec{\mu}, K_{ij}]$ with functional forms $[f(\vec{x}), m(\vec{x}), k(\vec{x_i}, \vec{x_j})]$. A Gaussian process can then be thought of as a distribution over functions, and training involves finding the optimal forms for the mean function $m(\vec{x})$ and a covariance kernel $k(\vec{x_i}, \vec{x_j})$. Predictions are made by finding the function values which maximize the joint posterior of the training data and the new input values, all of which are assumed to be drawn from the same Gaussian processes. The choice of covariance kernel reflects the expected properties of the underlying process, such as smoothness or periodicity. Figure 2 shows an example of fitting a Gaussian process, where both the fitted mean function and covariance kernel have been shown.

Gaussian process regression involves finding the likelihood distributions of the mean function $m(\vec{x})$ and covariance function $k(\vec{x_i}, \vec{x_j})$ which result analytically from the noisy training data. These likelihood distributions are combined with input prior distributions, to give the final posterior distributions from which predictions can be made. Our prior for the mean function is

$$m(\vec{x}) = \vec{A} + b\vec{x} \quad (7)$$

with trainable parameters $\vec{A}$ and b (initialised to zeros) specifying a linear relationship to each of the five input dimensions. Our prior for the covariance function is the Matern32 kernel,

$$k_{\text{M32}}(\vec{x_i}, \vec{x_j}) = \sigma^2\left(1 + \frac{\sqrt{3}|\vec{x_i} - \vec{x_j}|}{\rho}\right)\exp\left(-\frac{\sqrt{3}|\vec{x_i} - \vec{x_j}|}{\rho}\right) \quad (8)$$

with trainable parameters for the kernel variance σ^2 and kernel

length-scale ρ (both initialised to unity). The Matern32 is used to represent data with a moderate level of smoothing. Both of these kernel parameters control over-fitting of this model. For instance, a smaller value of ρ allows the mean function to change more rapidly as a function of the inputs, which can cause the model to overfit the training data.

Training this model involves finding the matrix elements $K_{ij} = k(\vec{x}_i, \vec{x}_j)$ of the training data. The expected mean and variance for a new prediction test location $\vec{x^*}$ are then given by

$$f(\vec{x^*}) = \sum_{i,j=1}^{N} k(\vec{x}_i, \vec{x^*})\left(K_{ij} + \sigma^2\delta_{ij}\right)^{-1} y_j\,, \tag{9}$$

$$\mathrm{Var}(f(\vec{x^*})) = k(\vec{x^*}, \vec{x^*}) - \sum_{i,j=1}^{N} k(\vec{x}_i, \vec{x^*})\left(K_{ij} + \sigma^2\delta_{ij}\right)^{-1} k(\vec{x^*}, \vec{x}_j)\,, \tag{10}$$

from Rasmussen & Williams (2006). Note that these equations involve inverting the large matrix $(K_{ij} + \sigma^2\delta_{ij})$, which in our case has 91000^2 elements. Using python 8-byte *float64* values, simply storing a single object instance of this matrix takes 60GB of RAM. Our computer architecture with 128GB of RAM is not large enough to invert such a matrix, since inversion requires much more RAM than a single matrix instance. Sparse Gaussian process regression (SGPR) is an approximation of GPR for huge data sets. SGPR approximates the matrix inversion by using only a subset of m observed data points and inverting this smaller matrix instead. These m 'inducing points' are effectively an additional set of fitting parameters. Our SGPR model uses the *gpflow* package[8] which implements the methods in Titsias (2009) using TensorFlow (Abadi et al. 2015). The *gpflow* package uses the *scipy.optimize.minimize* function with the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS-B) method to find the best set of inducing points. The minimization method uses the default termination method, i.e. when the maximum component of the objective function's gradient falls below a tolerance of 10^{-5}.

3.4 Support vector machine

Support vector machine (SVM) models are often used for classification, but can also be used for regression. In SVM classification, training involves finding a set of hyperplanes which separate the training data into their labelled classes while at the same time maximizing the distance between the hyperplanes and the nearest training data points. SVM regression extends this concept to functional forms, so that training the model involves finding a function $f(\vec{x})$ whose evaluations at the training points $\vec{x_n}$ are most similar to the observed training values y_n, while at the same time ensuring that the function is as simple as possible. We use the *scikit-learn* package (Pedregosa et al. 2011) for our support vector machine models.

SVM training involves finding the functional form $f(\vec{x})$ such that the residual errors between the training data $(\vec{x_n}, y_n)$ and the function evaluations $f(\vec{x_n})$ all lie within some tolerance $-\epsilon \leqslant f(\vec{x_n}) - y_n \leqslant \epsilon$. This stringent constraint usually makes it impossible to find any such form $f(\vec{x})$. To weaken the condition and allow a solution, the slack variables (ξ_n, ξ_n^*) are introduced so that the residual fitting error $f(\vec{x_n}) - y_n$ for the training point $(\vec{x_n}, y_n)$ obeys $-\epsilon - \xi_n^* \leqslant f(\vec{x_n}) - y_n \leqslant \epsilon + \xi_n$. This optimization problem is more easily solved in the dual form, with objective function

$$\begin{aligned}\text{SVR Objective} = &\sum_{i,j=1}^{N} \left(\alpha_i - \alpha_i^*\right) k(\vec{x}_i, \vec{x}_j) \left(\alpha_j - \alpha_j^*\right) \\ &+ \epsilon \sum_{i=1}^{N} \left(\alpha_i + \alpha_i^*\right) - \sum_{i=1}^{N} \left(y_i(\alpha_i - \alpha_i^*)\right).\end{aligned} \tag{11}$$

Training involves finding the values (α_i, α_i^*) which minimize this objective function, subject to margin constraints

$$\sum_{i=1}^{N} \left(\alpha_i - \alpha_i^*\right) = 0 \quad \text{and} \quad 0 \leqslant \alpha_i, \alpha_i^* \leqslant C\,. \tag{12}$$

The kernel function $k(\vec{x}_i, \vec{x}_j)$ in Equation (11) controls the functional form $f(\vec{x})$. We try three different kernel functions: radial basis function (RBF), polynomial, and sigmoid. As discussed in Section 5.2.2 later, the only kernel which gives rise to reasonable accuracy predictions (with MSE less than 500 %) is the RBF kernel,

$$k_{\mathrm{RBF}}(\vec{x}_i, \vec{x}_j) = \exp\left(-\gamma \left|\vec{x}_i - \vec{x}_j\right|^2\right). \tag{13}$$

The RBF kernel is infinitely differentiable, hence is often used to model data from smooth distributions. Before training, one must set the penalty term C, the kernel influence range γ (hereafter written *gamma* to match the python class parameter), and the margin tolerance ϵ (written *epsilon*). Overfitting for SVR models is discouraged by C the penalty term.

4 EMULATOR TRAINING

In this section we describe how we create the training data and the specific choices we make in training our emulators. Standard practice is to use a large training dataset and then check that the trained emulators make valid predictions for unseen validation data. The training results are given later in Section 5. All emulators are trained on the same architecture, each on a single node using 16 Xeon E5-2650 cores and 128GB RAM.

4.1 *SimFast21* simulations

We run 2000 *SimFast21* simulations in total, retaining only the three input reionization parameters and the final output spherically averaged power spectra for each simulation. We use 1000 simulations for training, 500 for validation, and another 500 for testing the emulators which have highest prediction accuracy on the validation data. Each simulation generates three-dimensional realisations of the δT_b field in a cube of size 500Mpc resolved into 512^3 pixels (smoothed from density fields resolved into 1536^3 pixels). This gives power spectra values for seven redshift values: $\{8.0, 9.5, 11.0, 12.5, 14.0, 15.5, 17.0\}$ and thirteen k-values in the range $\{0.02, 3.0\}$ hMpc^{-1}. This corresponds to 91000 overall training data points, and 45500 data points each for validation and testing. The power spectra data have size of 335MB for all 2000 simulations, compared to 7TB size of all δT_b boxes.

[8] http://gpflow.readthedocs.io/en/latest/intro.html

4.2 Training set design

Our emulators map five input values to a single output target value. The target value is the δT_b power spectrum value for the given inputs. The first three input values are the three reionization parameters (see Section 2.2), which are different for each simulation. The final two inputs are the redshift z and the k-value, the values for which are constant across all simulations and are given in Section 4.1. The function $f(\vec{x})$ which the models are fitting is then the spherically averaged 21cm power spectrum $P_{\Delta T_b}(M_{\rm min}, \zeta_{\rm ion}, R_{\rm max}, z, k)$.

We use the Latin Hypercube method designed by McKay (1979) to choose the reionization parameter values for our simulations. The Latin Hypercube method provides a way to sample the three-dimensional input space in a more efficient way than naive exhaustive grid-search. We use the following ranges and scalings for the reionization parameters:

(i) $M_{\rm min}$ in the logarithmic range $[10^{7.8}, 10^{9.8}]$
(ii) $\zeta_{\rm ion}$ in the linear range $[5, 100]$
(iii) $R_{\rm max}$ in the linear range $[5, 20]$

These ranges match those used by the semi-numerical simulation authors, see for example Greig & Mesinger (2015). The lower $M_{\rm min}$ limit comes from the lowest temperature at which atomic hydrogen can cool and accrete onto halos, and the upper limit from observations of high-redshift Lyman break galaxies (Greig & Mesinger 2015). The $\zeta_{\rm ion}$ range roughly corresponds to ionizing photon escape fractions of 5% to 100%. The $R_{\rm max}$ range arises from recombination models of Sobacchi & Mesinger (2014), and only has an effect near the end of reionization when the ionized bubble sizes are comparable to $R_{\rm max}$ (Alvarez & Abel 2012, McQuinn et al. 2007). See Figures 8 and 9 later for example power spectra across these ranges for $\zeta_{\rm ion}$ and $M_{\rm min}$ values.

We also test three different scaling types for the target values to determine which gives the most accurate emulation. These three are a linear function $y = P_k$, a logarithmic function $y = \log[P_k]$, and a pseudo-logarithmic function $y = \sinh^{-1}[P_k]$ sometimes called luptitude after Lupton et al. (1999). We test logarithmic scaling as an attempt to exploit the fact that power spectra appear more naturally spaced in logarithmic space $\log[P_k]$ than in linear space P_k. However a few percent of the power spectra data are zero-valued, especially at early and late redshifts where the ionization field $x_{\rm HII}(\vec{r})$ becomes uniform and δT_b is effectively zero everywhere (Pritchard & Loeb 2012, pages 12-13). Our motivation for luptitude scaling is to retain as much data as possible: a purely logarithmic scaling would require us to throw away all zero-valued data points and reduce the size of our training data set. We comment on the effects of including or excluding these zero-valued data in Section 6.2.

4.3 k-range restriction

We exclude the largest and smallest scales from our validation and testing data, including only $0.1 \leqslant k \leqslant 2.0$ values. On large scales ($k < 0.1$ hMpc^{-1}), the power spectrum is affected by foregrounds Datta et al. (2010). The finite resolution of our simulations means that there is little information in the power spectrum on very small scales ($k > 2.0$ hMpc^{-1}). These restrictions are common for semi-numerical simulations, see for example Greig & Mesinger (2015).

4.4 Goodness of fit evaluations

For validation and testing, we measure the goodness of fit between predicted target values $y^*(k, z)$ and measured target values $y(k, z)$ using the mean squared error

$$\mathrm{MSE}\big[y(k,z), y^*(k,z)\big] = \frac{1}{N_z N_k} \sum_{z}^{N_z} \sum_{k}^{N_k} \left(\frac{y(k,z) - y^*(k,z)}{y(k,z)} \right)^2 \quad (14)$$

along with the percentage mean squared error, $100 \times$ MSE. The MSE is averaged over all N_z redshifts values and all N_k scale values in the range $0.1 \leqslant k \leqslant 2.0$, unless explicitly mentioned otherwise. For comparability, we use this same error function for all different emulators during validation and testing, although the models use different error metrics for determining their training convergence (see Section 3 for the training objective functions for each model).

5 EMULATOR TRAINING RESULTS

After training each emulator, we test its accuracy by generating predictions for a set of unseen validation data. By calculating the MSE value in Equation (14) between the predicted outputs and the true outputs, we determine which emulator makes the most accurate predictions. A low MSE means a high prediction accuracy.

5.1 Target value scaling

Here we compare the prediction accuracy for the three scaling methods of the target power spectra values: linear, logarithmic, and pseudo-logarithmic $\sinh^{-1}(x)$. As expected, the linear function has poor prediction accuracy because the power spectra values are more naturally spaced in logarithmic space than in linear space. The logarithmic function works fairly well at intermediate redshifts for this same reason, but all of the zero-valued power spectrum values had to be discarded as $\log(0)$ is undefined. The pseudo-logarithmic function $\sinh^{-1}(x)$ has the highest prediction accuracy over all redshifts and allows us to retain all training data points (with zero-valued outputs or otherwise). We use the pseudo-logarithmic function in all our emulators from here on.

5.2 Hyperparameter searching

Each model has a set of trainable values referred to as fitting parameters. Many models have an additional set of values which must be fixed even before starting to train, referred to as hyperparameters. Here we describe which hyperparameters (if any) we vary for each model type, and which hyperparameters give rise to the best prediction accuracy. For each model, we restrict the total training time for all hyperparameter searching to 156 CPU hours. The interpolation models involve no hyperparameters, and for the SGPR model we simply increase the number of inducing points m until the individual model's training time reaches 156 CPU hours. Increasing m should always increase the SGPR model's accuracy and so the value of m is not treated as a hyperparameter when considering the total training time. Including models with smaller m values in the total training time would give a smaller maximum value of m, making an unfair comparison with the other models.

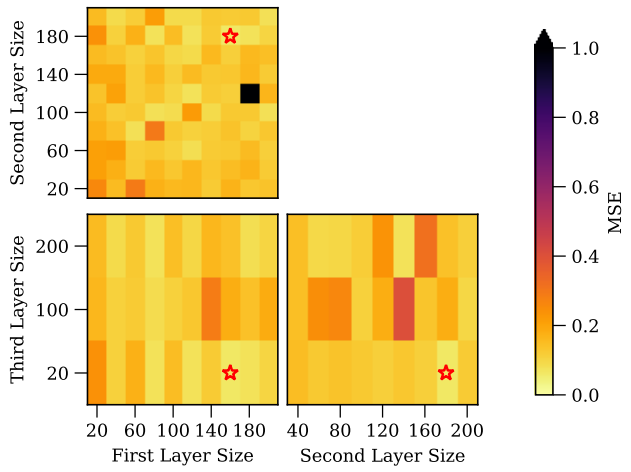


Figure 3. Mean squared error on the validation data for three-layer multilayer perceptron models, as a function of the sizes of each hidden layer. The red star shows the layer sizes of the MLP emulator with the highest prediction accuracy: 160 neurons in the first hidden layer, 180 neurons in the second hidden layer, and 20 neurons in the final hidden layer.

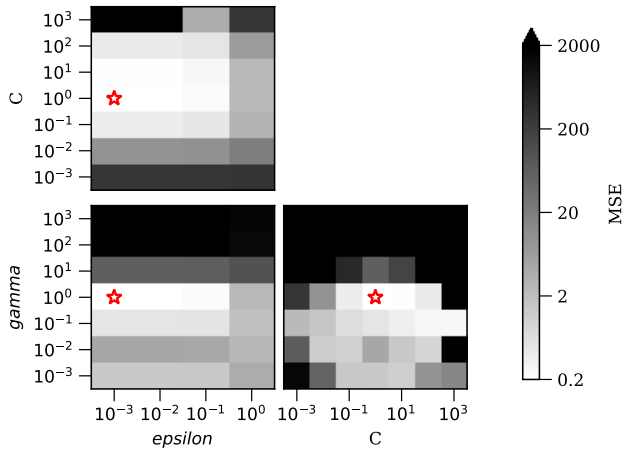


Figure 4. Mean squared error on the validation data as a function of the model hyperparameters, for support vector machine emulators using the RBF kernel. The hyperparameters are the penalty term C, margin tolerance *epsilon* and influence range *gamma*. The spread of MSE values is much larger for SVM models, indicated by the logarithmic colour scale between MSE values of 10^{-1} and 10^{3}. The hyperparameters of the highest prediction accuracy SVM model are indicated by the red star: $C = 1.0$, *epsilon* $= 10^{-3}$ and *gamma* $= 1.0$.

5.2.1 *MLP layer sizes*

We use MLP models with one, two and three hidden layers. The sizes of the hidden layers were varied linearly in the range $[0, 200]$ using a simple grid-search method. Generally, the emulator models with more hidden layers have higher prediction accuracy. The validation MSE values for the best one-, two- and three-layer MLP emulators are 13%, 2.3%, and 1.6% respectively. The validation MSE values for three-layer MLP emulators are shown in Figure 3 as a function of the sizes of each of the three hidden layers. Most three-layer MLP models have a low validation MSE near 10%. The best emulator has hidden layers sizes $160 - 180 - 20$, the hyperparameters for which are indicated by the location of the red star. Our MLP models end training when the objective function changes more slowly than a threshold tolerance for several training epochs. Most of our MLP models achieved this in fewer than 400 training epochs, with some 1-layer models lasting up to 800 epochs.

5.2.2 *SVM margin hyperparameters*

We test a range of SVM emulators with different values for three hyperparameters controlling the margin. We vary the penalty parameter C logarithmically in the range $[10^{-3}, 10^{3}]$; the tolerance *epsilon* logarithmically in the range $[10^{-3}, 10^{0}]$; and the kernel influence range *gamma* logarithmically in the range $[10^{-3}, 10^{3}]$. These hyperparameters are the suggested ranges by *sklearn* and we use a simple grid-search to find the best hyperparameters. We also test three kernel functions: RBF, sigmoid, and polynomial. Figure 4 shows how the validation MSE of emulators using the RBF kernel depends on the SVM hyperparameters. The different colour-map is used to emphasise that the colour range is logarithmic and has a much larger spread of MSE values between 0.2 and 2000 (or between 20% and 2×10^{5}%). The best SVM emulator has validation MSE of 20%, using hyperparameters $C = 1.0$, *epsilon* $= 10^{-3}$, *gamma* $= 1.0$ and the RBF kernel. All SVM emulators with kernels other than RBF have much worse validation MSE: the best polynomial and sigmoid SVM emulators have validation MSEs of 50000% and 500% respectively.

5.3 Overfitting tests

For each model we determine the best hyperparameters by trying a range of values and selecting the emulator which shows the highest prediction accuracy on the validation data. By trying different hyperparameter values we can usually find a closer fit to the data. However, this process is sensitive to over-fitting: the model might fit the training data more closely, but it may not extend well to new data. We test for overfitting by training a series of emulators with increasing training dataset sizes, keeping the hyperparameters fixed at the proposed best values. Providing more training data should give rise to improved predictions for the unseen validation data. If providing more training data instead leads to a decrease in validation prediction accuracy, then overfitting has occurred: the model makes good predictions for the training data, but does not extend well to new input values. Figure 5 shows the results of these tests, giving the mean square error on the validation data for each model, using differently sized training datasets. All mean squared errors generally decrease with increased training set size, implying that none has been overfitted.

5.4 Performance on testing data

Here we test the performance of the best emulator for each model type using all 500 simulations in our testing set. Table 1 shows the accuracy and speed of each emulator for making predictions on the entire testing dataset. The global MSE percentage is averaged across the entire testing data set. In Section 6.2 later, we discuss the fact that our emulators have worse accuracy at lower redshifts. We include a column in Table 1 for the percentage MSE averaged across the testing data with higher redshifts ($z \geqslant 10$). Figure 6 shows an

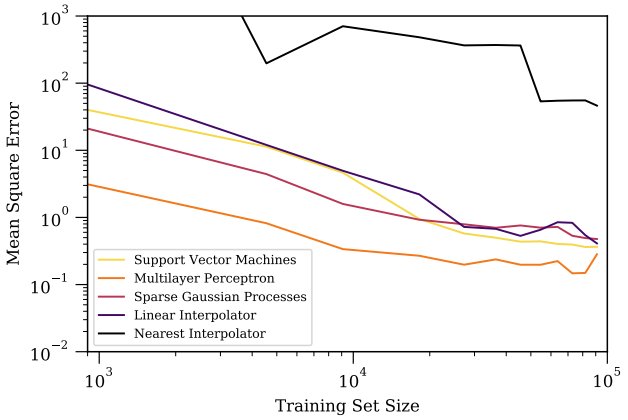


Figure 5. Mean squared error on the validation data as a function of training set size. The best hyperparameters were fixed for each model, and the emulator retrained with more training information. The MSE curves generally improve with more training data, implying that none has been overfitted.

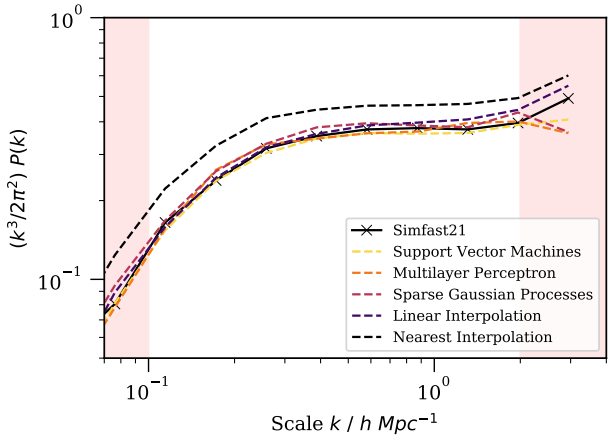


Figure 6. Predicted δT_b power spectra of a canonical simulation with reionization parameters $\{5 \times 10^8 M_\odot, 30.0, 10$ Mpc$\}$. Dotted lines show the predictions from the best emulator of each type. Solid line shows the power spectrum from an actual *SimFast21* simulation. The red shaded areas indicate the k-values that were excluded from our validation and testing. This test simulation was chosen from the testing data as the nearest to the canonical reionization parameters. The model using nearest-neighbour interpolation has significantly different predictions, likely owing to the underfitting processes discussed in Section 6.1.

example of the power spectra outputs from the best emulator of each model type, showing the predictions for a single test simulation near the canonical reionization parameters at $z = 9.5$.

6 EMULATOR TRAINING DISCUSSION

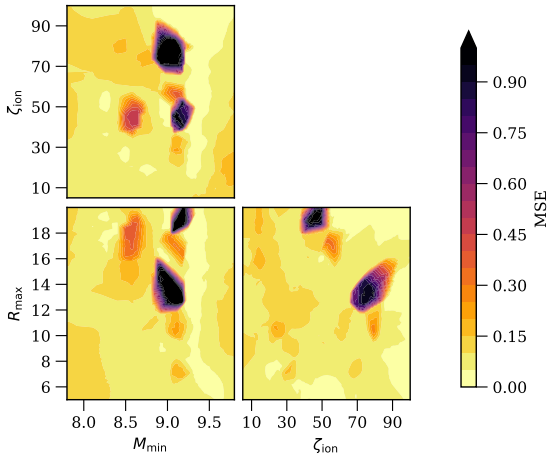
Figure 7 shows the prediction MSE of each emulator as a function of location in parameter space. The dark regions indicate the regions of parameter space which are most difficult to emulate. All panels in Figure 7 show a region of poorer prediction accuracy for inputs near $M_{\rm min} = 10^9$. This is likely due to the finite mass resolution of our *SimFast21* simulations. For values of $M_{\rm min}$ near the mass resolution, the simulation switches between containing both resolved and unresolved halos (if $M_{\rm min} < 5 \times 10^9$), and containing only resolved halos (if $M_{\rm min} > 5 \times 10^9$). The change in behaviour appears to be difficult to emulate for all model types.

6.1 Speed and accuracy performance

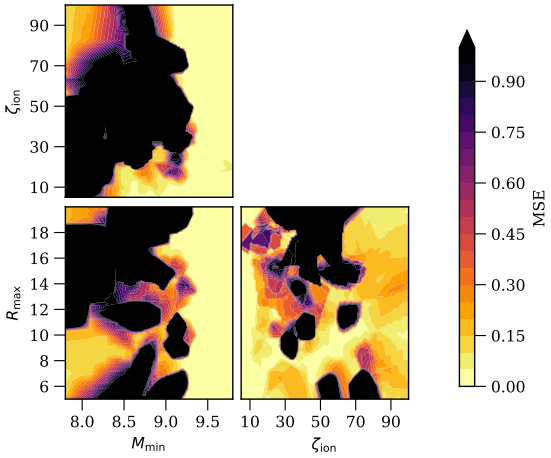
The three-layer multilayer perceptron is the best candidate for emulating *SimFast21* behaviour. Table 1 shows that this emulator makes fast and accurate predictions for the test dataset, taking less than a second to match the true simulation outputs within 4% mean squared error averaged across the whole input parameter space. Figure 7a shows that the emulator makes accurate predictions across a wide range of input parameters. Worse performance is seen for MLP emulators using fewer hidden layers: increasing the number of layers allows MLP models to be more flexible, and our results indicate that one- and two-layer MLP models are not flexible enough to fit the simulation outputs as accurately as three-layer models. Figures 8 and 9 show several example power spectra for a range of $\zeta_{\rm ion}$ and $M_{\rm min}$ values, also showing the predicted power spectra from this best emulator. The shaded red regions in these figures indicate the ranges of excluded k-values. Given the benefit of most three-layer models over two-layer models, it seems likely that models using four or more layers could provide even closer fit to the training data. We do not investigate such models, given our fixed upper limit on training time. Additionally, the benefit of adding more layers would likely be minimal as there is a clear case of diminishing returns for each additional layer: the best MSE for one layer was 27%; two layers gave 4.5% MSE; and three layers gave 3.8%.

The two interpolation models are the worst candidates for emulating *SimFast21* behaviour. The nearest-neighbour interpolation model has poor prediction accuracy both in terms of the global MSE value of 290% from Table 1, and the local MSE across parameter space shown in Figure 7b. The model uses the nearest-neighbour lookup method of *scipy.spatial.KDTree* which is fast but makes no account of noise or smoothness in the simulation behaviour. The linear interpolation model emulates the *SimFast21* behaviour more closely: the global MSE is 17% and the local MSE in Figure 7c shows larger regions of good accuracy. This accuracy is at the expense of much slower prediction times. The nearest neighbour model makes predictions for the whole testing dataset in less than a second, whereas the linear interpolation model takes several hours. Our results indicate that interpolation methods cannot capture the complicated behaviour of *SimFast21*, justifying the need for more complex machine learning techniques.

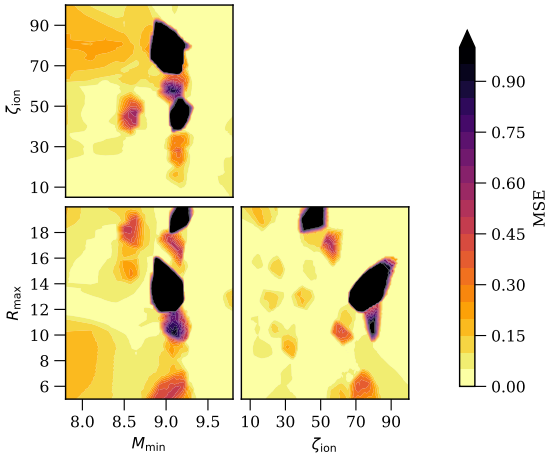
Our sparse Gaussian processes model is a poor emulator candidate. Both the local MSE in Figure 7d and the global MSE of 36% are poor. The accuracy of the model would almost certainly be improved by increasing the number of inducing points, which would lessen the matrix inversion approximation. However, training models with $m > 2730$ would require more than the allowed CPU time. Our value of $m = 2730$ is chosen as the largest number of inducing points whose model training time does not exceed 156 hours. A hard upper limit of $m < 18000$ is found for our 128GB RAM architecture, since values of m larger than this cause a *ResourceError* in *tensorflow*. Moreover, increasing the number of inducing points also increases the prediction time: using $m = 910$ takes 16 seconds to make predictions for the testing dataset; using $m = 2730$ takes 116 seconds. Increasing the number of inducing points gives better accuracy at the expense of much slower prediction times.



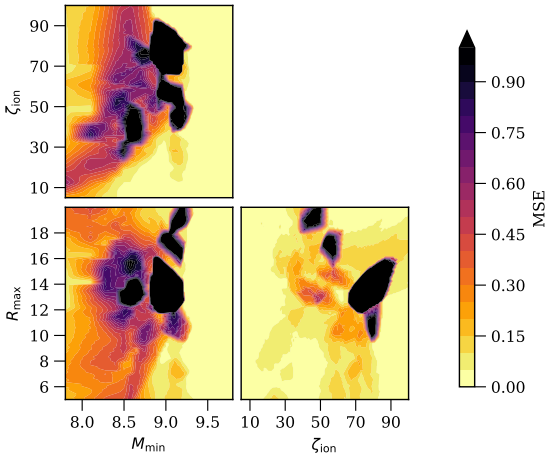
(a) Three-layer Multilayer Perceptron



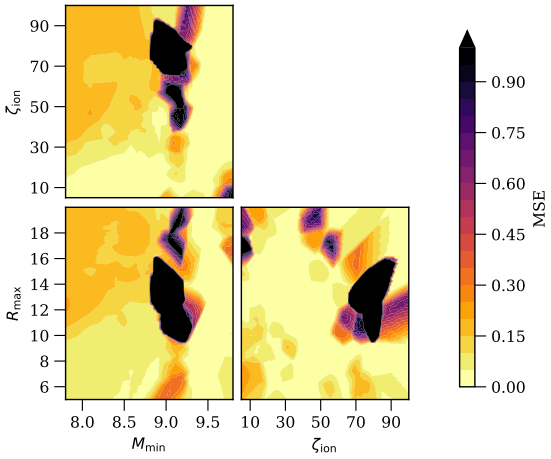
(b) Nearest ND Interpolation



(c) Linear ND Interpolation



(d) Sparse Gaussian Processes



(e) Support Vector Machine

Figure 7. Mean squared error on testing dataset for the five model types, as a function of prediction location in the three-dimensional reionization parameter space. Each panel shows the MSE values marginalized over all but two input dimensions. For instance, the $\zeta_{\rm ion}$-$M_{\rm min}$ panels show the MSE values as marginalised over $\{R_{\rm max}, z, \text{and } k\}$ dimensions.

Model Type	Global Test MSE % (all z)	Test MSE % for $z \geqslant 10$	Prediction Time
Nearest Neighbour Interpolation	290 % *	5.1 %	0.20s
Sparse Gaussian Processes $m = 2730$	36 %	0.6 %	116s
Support Vector Machine	32 %	2.1 %	27s
1-layer Multilayer Perceptron	27 %	9.2 %	0.07s
Linear Interpolation	17 %	1.6 %	4.1 hours *
2-layer Multilayer Perceptron	4.5 %	2.3 %	0.14s
3-layer Multilayer Perceptron	3.8 %	1.4 %	0.27s

Table 1. Speed and accuracy performance of the best emulator for each technique, using the testing data set. The percentage MSE values here are $100 \times$ MSE. The rows are sorted in order of global prediction accuracy, from highest MSE (least accurate) at the top to lowest MSE (most accurate) at the bottom. We give global MSE values averaged across the entire dataset and also the MSE for a subset of the testing data with $z \geqslant 10$ to demonstrate that most of the poor accuracy occurs at later redshifts. See Section 6 for a discussion on the extreme (*) values for the two naive interpolation methods. For all models except SGPR, the total time for hyperparameter searches is 156 CPU hours. For SGPR model, we run a single model with the largest possible number of inducing points m without exceeding 156 hours training time.

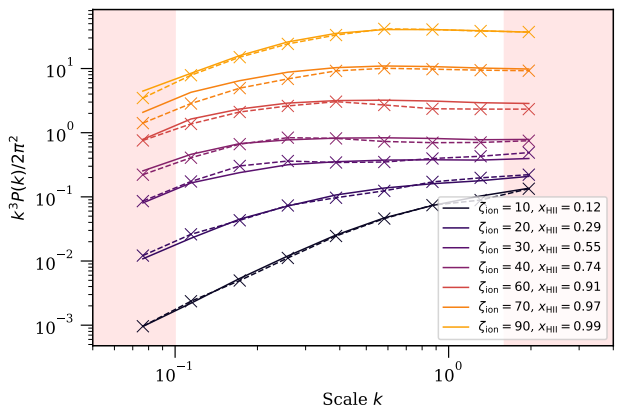


Figure 8. Example emulated and simulated power spectra for a range of $\zeta_{\rm ion}$ values at $z = 9.5$, for fixed $M_{\rm min} = 5 \times 10^8$ and $R_{\rm max} = 10$. Solid line shows the simulated power spectra, dotted line shows the predicted power spectra from our best emulator. The ionization fraction for each line is given in the legend.

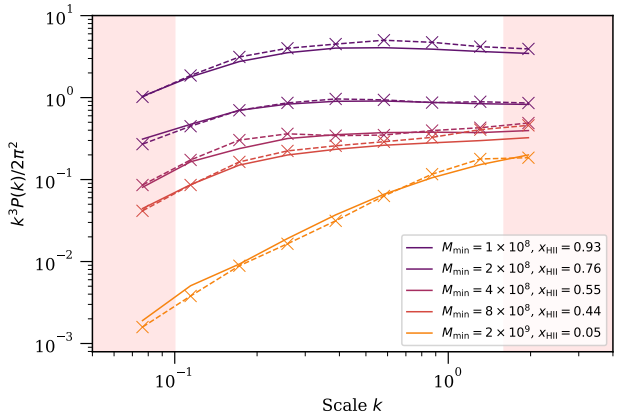


Figure 9. Example emulated and simulated power spectra for a range of $M_{\rm min}$ values at $z = 9.5$, for fixed $\zeta_{\rm ion} = 30$ and $R_{\rm max} = 10$. Solid line shows the simulated power spectra, dotted line shows the predicted power spectra from our best emulator. The ionization fraction for each line is given in the legend.

The support vector machine model is also a poor candidate for a *SimFast21* emulator. The global MSE of 32% from Table 1 is one of the worst. This model also has slow prediction speeds, taking 27 seconds to make predictions of the testing data (100 times slower than the best MLP model). It is possible that using other kernels and doing deeper hyperparameter searches would give better accuracy. Given the long prediction times for these models, we find it unlikely that any SVM models would outperform our best MLP emulator, either in terms of speed or accuracy.

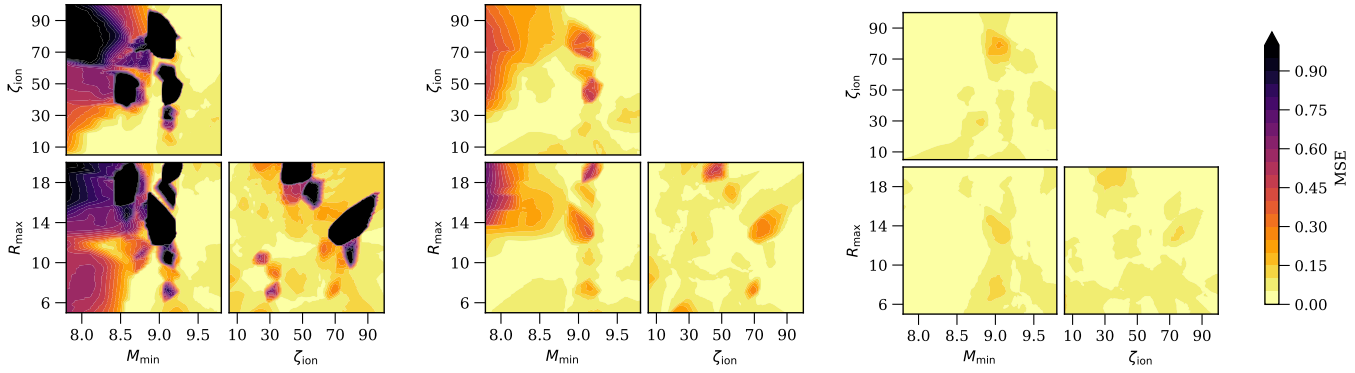
6.2 Low redshift performance

The prediction accuracy of our emulators is worse for lower redshifts than for higher redshifts. If data for $z < 10$ are excluded from performance testing, then all our emulators improve significantly: for instance, the three-layer multilayer perceptron's percentage MSE improves from 3.8% to 1.4%. The improved values using only high-redshift power spectra are presented in the third column of Table 1. There are two effects that could be causing the worse accuracy at lower redshift, which we discuss here.

First, our emulators differ from those of Kern et al. (2017) and Schmit & Pritchard (2018) in that our models are trained using the redshift and k-scales as extra input dimensions. Our motivation for including redshift and k-scales was to allow for immediate predictions at any redshift or k-scale. Without including these input dimensions the trained models would only make prediction at the fixed redshifts and k-scales of the training data. Making predictions at other input values with such fixed-input emulators would require further interpolation afterwards. Although using z as an input allows for more flexible predictions, this flexibility is likely a cause of poorer emulation at lower redshifts. Without more computing power or faster training algorithms we would suggest that future attempts to emulate the power spectrum should be done for fixed z inputs.

Secondly, a feature of the actual simulated power spectra could be another source of poor emulator accuracy. The amplitude of the power spectrum in Equation (3) is highly sensitive to the global 21cm brightness temperature $\langle \delta T_{\rm b} \rangle$. At low redshifts near the end of reionization, $\langle \delta T_{\rm b} \rangle$ approaches zero (see Pritchard & Loeb 2012 for a review). This low value of $\langle \delta T_{\rm b} \rangle$ amplifies even small fluctuations in $\delta T_{\rm b}$, causing a sharp increase in the fluctuation field $\Delta T_b(\vec{r})$ from which the power spectrum is calculated. Soon after, reionization finishes and $\Delta T_b(\vec{r}) = 0$ everywhere so that the amplitude of $P_{\Delta T_{\rm b}}(k)$ jumps suddenly from high-amplitude to zero-amplitude. These two sudden features are difficult to emulate: the sharp increase in power spectrum amplitude near the end of reionization, and the sudden drop thereafter from high-amplitude to zero-amplitude.

We investigate the low redshift behaviour in two ways. First,



(a) Three-layer MLP for $z = 8.0$ (b) Three-layer MLP for $z = 9.5$ (c) Three-layer MLP for $z = 11.0$

Figure 10. Mean squared error on testing dataset for the best MLP model as a function of prediction location, similar to Figure 7 but without averaging over redshift. The prediction quality is much worse at low redshift than it is at high redshift, shown by the large darker regions at low redshift. The percentage MSE for $z \geqslant 11.0$ is better than 5% across almost all of the input parameter space. We omit plotting panels for each $z > 11.0$ here as they all look similar to that for $z = 11.0$.

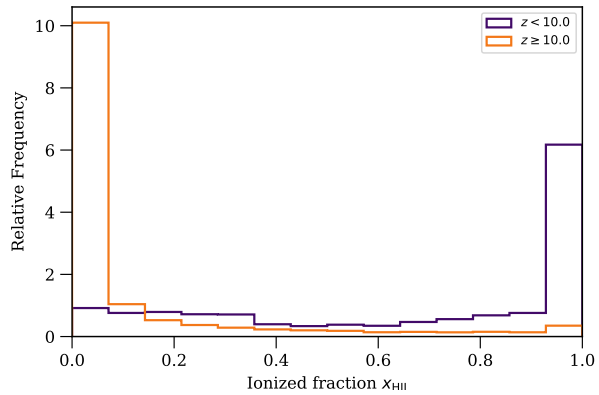


Figure 11. Normalised histogram of the ionization fraction for all simulations, for low redshifts ($z < 10$) and for higher redshifts ($z \geqslant 10$). The 21cm power spectrum is sensitive to the neutral fraction. The fact that many simulations are fully ionized for $z < 10$ could be one reason for the poor performance of our emulators at low redshifts.

we plot the local prediction performance of our best emulator at each redshift in the training data. Figure 10 shows the best emulator's local MSE separately for the three lowest redshifts $z = \{8.0,\ 9.5,\ 11.0\}$. For $z \geqslant 10$, very few of the simulations in our training data have completed reionization and thus very few contain the associated problematic zero-valued power spectra. In Figure 10c and for higher redshifts, the local performance is consistently good across the parameter space. At $z = 9.5$ some simulations are near the end of reionization, and the local performance in Figure 10b begins to show regions of poorer predictions. By $z = 8$, a large number of simulations have completed reionization. These regions have zero-amplitude power spectra and show much worse performance, in particular those with high $\zeta_{\rm ion}$ and low $M_{\rm min}$ in Figure 10a.

Our second method to examine the low redshift behaviour is to observe how many simulations have finished reionization at each redshift. Figure 11 shows normalised histograms of the global ionization fraction values for all simulations in our data, separating into lower redshifts ($z < 10$) and higher redshifts ($z \geqslant 10$). For higher redshifts, most simulations are near the start of reionization, with ionization fractions $x_{\rm HII} < 0.4$. For lower redshifts, many models have finished reionization with ionization fractions nearing $x_{\rm HII} = 1.0$. This difference causes the low redshift power spectra to contain sudden features which are difficult to emulate. Training using $\delta T_{\rm b} - \langle \delta T_{\rm b} \rangle$ as the target values rather than $\Delta T_{\rm b}$ could remove the sudden changes in power spectra magnitudes and would be easier to emulate. We note for instance that Kern et al. (2017) were able to emulate down to $z = 5$ without reporting any issues for emulating these redshifts.

7 USE CASE: MAPPING BETWEEN *SimFast21* AND *21cmFAST*

21cmFAST and *SimFast21* are two common semi-numerical simulations for generating predicted 21cm maps during the Epoch of Reionization. In this section we use our best emulator to investigate the extent to which the two simulations give similar outputs for similar inputs. Our motivation for this is to demonstrate a method for creating a mapping between any two simulations which generate the same output statistic. In particular, this method could be extended to give a mapping between *SimFast21* power spectra and those from more accurate (but slower) three-dimensional radiative transfer simulations, such as C^2-RAY (Mellema et al. 2006). Although numerical simulations have different input parameters to semi-numerical simulations, it would still be possible to map between them by finding the parameters which best match their output power spectra. When analysing huge datasets, *SimFast21* could be used to give coarse constraints on reionization parameters. Using the mapping between *SimFast21* and the more accurate numerical simulation, the coarse contours could be mapped to their equivalent regions of the numerical simulation inputs. This would allow more detailed exploration of this smaller region of parameter space with the numerical simulations.

7.1 *21cmFAST*

Here we describe the default procedure of *21cmFAST*, in particular highlighting how it differs from the *SimFast21* algorithms described earlier in Section 2.2. In the following section we discuss which of these differences we retain when creating the mapping. The linear and non-linear density fields in *21cmFAST* are seeded in the same way as in *SimFast21*.

The first difference between the simulations is the method for calculating collapse fractions from the non-linear density field. *21cmFAST* does not resolve individual halos, but rather calculates the collapse fraction directly from the non-linear density field following the model of spherical collapse from Press & Schechter (1974). In order to match the more accurate ellipsoidal collapse model from Sheth et al. (2001), *21cmFAST* afterwards normalises the spherical collapse fractions so that their average value matches that expected from ellipsoidal collapse.

The second difference is the method for calculating the ionization fraction. Both simulations calculate the ionization fraction by determining whether the collapsed matter in a region emits enough photons to ionize the surrounding matter. If there are enough photons, then *SimFast21* paints the entire spherical region as ionized using the fully overlapping-spheres method in Mesinger & Furlanetto (2007), whereas the default *21cmFAST* algorithm is to paint only the central pixel of the region Zahn et al. (2007). The latter method is much faster but the algorithms give a considerably different reionization history for the same inputs (see Hutter 2018). *21cmFAST* has an option to match the method of *SimFast21*.

The final difference is in the evolution of the parameter $M_{\rm min}$. The default *21cmFAST* implementation allows the minimum halo mass $M_{\rm min}$ to evolve with redshift by setting a minimum virial temperature $T_{\rm vir}$ for ionizing photons.

7.2 Matching reionization histories

Using identical reionization and cosmological parameters, and keeping all other input parameters at their default values from the *GitHub* packages, *21cmFAST* version 1.2[9] and *SimFast21* version 1.0 result in different reionization histories, as expected due to the different default bubble-finding algorithms. Our motivation in this section is to demonstrate a method for mapping between the input parameters of two similar (but not identical) simulations. Using the default implementations, the output power spectra of the two simulations at a single fixed redshift are not comparable because the two simulations have reached different stages of reionization. Before making the mapping, we chose input parameters of *21cmFAST* which more closely matched the *SimFast21* algorithm, so that the output power spectra are similar enough that making a mapping is meaningful, but not so similar that they give identical results. The following is a list of significant input parameters in *21cmFAST* that we adjusted from the default values:

(i) FIND_BUBBLE_ALGORITHM = 1
(ii) ION_Tvir_MIN = −1, instead using ION_M_MIN
(iii) INHOMO_RECO = 0

Appendix A lists all parameters used in both simulations for repeatability. The most significant change from default was in the algorithm for finding ionized bubbles, setting FIND_BUBBLE_ALGORITHM = 1. Without making a judgement on which method is more realistic we used the *SimFast21* algorithm of painting the entire sphere as ionized, rather than painting only the central pixel. We fix the minimum mass $M_{\rm min}$ for collapse using ION_M_MIN, rather than using the default *21cmFAST* functionality of a fixed virial temperature $T_{\rm vir}$ using ION_Tvir_MIN. We also turn off calculations involving inhomogeneous recombinations by setting INHOMO_RECO = 0, since the version of *SimFast21* that we use does not have this option (although later versions do, see Hassan et al. 2016). Figure 12 shows the resulting ranges of reionization histories from a spread of minimum halo mass scenarios between $10^8 M_\odot - 10^9 M_\odot$. Each minimum mass scenario is averaged across five realisations. The histories are shown for *SimFast21* (dotted) and for *21cmFAST* with both bubble-finding algorithms: ionising the central pixel only (darker red region) and ionising the full sphere (lighter orange region) to match *SimFast21*. The only remaining major differences between the default *SimFast21* simulation and the changed *21cmFAST* simulation are in the specifics of implementation discussed above. Figure 12 shows that the differences in implementation still result in different reionisation histories even after matching the bubble-finding algorithms, although the bubble-finding algorithm is the most dominant effect.

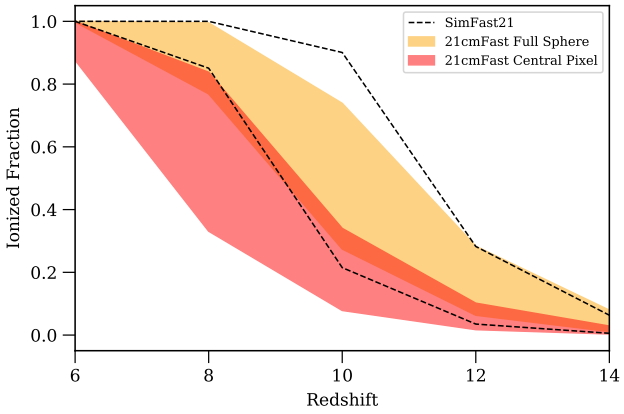


Figure 12. Ranges of reionization histories that result from *SimFast21* and *21cmFAST*, with $M_{\rm min}$ varying from $10^8 M_\odot$ to $10^9 M_\odot$. The region between the black dotted curves indicates the range of histories from *SimFast21*. The two coloured regions show the range of histories from *21cmFAST*, both before (darker red) and after (lighter orange) matching the algorithms. The other reionization parameters are fixed at $\zeta_{\rm ion} = 30.0$ and $R_{\rm max} = 10.0$. The bubble-finding algorithm has a significant impact on the resulting reionization history, and even after matching algorithms there is a slight difference between *SimFast21* and *21cmFAST*.

7.3 Determining a mapping between simulations

Here we describe how we use our best emulator to determine a mapping between the inputs of our modified *21cmFAST* and the inputs of *SimFast21*. We use the same k-space restrictions as in Section 4.3, using only $0.1 \leqslant k \leqslant 2.0$ since the large scales are subject to foregrounds and the small scales are subject to shot noise from the finite simulation resolution. We also restrict our comparisons to higher redshifts $z \geqslant 10$ for which our emulator exhibits higher prediction accuracy. We emphasise that this is a proof-of-concept method showing how to make a mapping between simulations solely using the output power spectra.

[9] https://github.com/andreimesinger/21cmFAST

Figure 13 shows an example of one such mapping. We explain how to interpret the mapping here. Suppose a reference *21cmFAST* simulation has already been run using the parameters specified by the white star: namely, a *21cmFAST* simulation with parameters $M_{\rm min} = 3 \times 10^8 M_\odot$, $\zeta_{\rm ion} = 30.0$, $R_{\rm max} = 10$Mpc. According to the mapping in Figure 13, any *SimFast21* simulation using parameters within the orange contour will result in power spectra which are similar to the reference *21cmFAST* spectra. We classify two simulations as similar if the mean-squared error between their output power spectra is lower than 30%. The orange contour thus shows the region of *SimFast21* parameters which should be used, if the desired result is to exhibit similar power spectra to the reference *21cmFAST* simulation. We generate the reference power spectra in Figure 13 by running five *21cmFAST* simulations, and taking the average to reduce the effect of sample variance. We then generate the contours by using our emulator to run a large number of simulations across the whole parameter space. For each emulated simulation, we calculate the mean-squared error of its power spectra compared to the reference simulation. Note that our emulator makes these predictions in seconds, rather than the several months that would be needed to run the same number of full simulations. We refer to this type of figure as a similarity plot. Most importantly, if the orange contour does not overlap with the white star, then this indicates that *SimFast21* and *21cmFAST* result in significantly different output power spectra for the same input parameters.

Two features of the orange contours are immediately apparent. First, the extended contours in the $R_{\rm max}$ direction. The $R_{\rm max}$ parameter is known to have little effect on the output power spectra for our high redshifts (Mesinger et al. 2011). This is an inherent property of the power spectrum, regardless of which simulation is used. A second clear feature is the large curved contour in the $M_{\rm min}$-$\zeta_{\rm ion}$ parameter space. We investigated both features, to confirm whether they arise as an inherent property of the power spectrum itself, or if they arise from differences in the two simulations. To do this, we perform the same similarity analysis as above, but using *SimFast21* itself as the reference simulation. The purple contours in Figure 13 then give the regions of *SimFast21* parameters which result in similar power spectra to the reference *SimFast21* simulation. The lighter purple contours use a MSE threshold of 30%. The darker purple contours use a stricter threshold of 15% MSE. The curved feature appears in both orange and purple contours, indicating that it is not due to a difference in the simulations. This curved degeneracy has been observed previously, see for example Greig & Mesinger (2015) and Schmit & Pritchard (2018). Note that we do not include a dark orange contour for the the stricter 15% MSE threshold because the power spectra for *21cmFAST* differ from those of *SimFast21* enough that no *21cmFAST* contours are visible for an MSE threshold of 15%.

Figure 14 shows similarity plots for several other reference simulations, where the parameters for each reference simulation is again indicated by the location of the white star. We show the contours in the two-dimensional $M_{\rm min}$-$\zeta_{\rm ion}$ space, ignoring the less interesting $R_{\rm max}$ direction. We find that the orange contour does not always lie on top of the white star. This indicates that *SimFast21* and *21cmFAST* do not always result in similar output power spectra. We use the same contour levels as in Figure 13, namely 15% and 30% for the darker and lighter purple contours, and 30% for the *21cmFAST* contours. Again, no 15% MSE contour is shown for *21cmFAST* because the *SimFast21* power spectra differ from the *21cmFAST* power spectra by more than 15% everywhere.

For several of these scenarios, there is an offset between the orange contour and the white star. The offset is small near the canonical parameters in the central panels, but gets larger at lower $M_{\rm min}$ and higher $\zeta_{\rm ion}$. The most likely reason for this offset is the difference in the reionization histories. This offset would mean that the choice of using *SimFast21* or using *21cmFAST* would affect the outcome of parameter estimation methods, such as maximising χ^2 values in Shimabukuro & Semelin (2017), or using MCMC methods as in Schmit & Pritchard (2018) and Kern et al. (2017). Note that the two simulations in this comparison needn't share the same types of input parameters. For instance, it would be possible to generate the reference power spectra using a numerical radiative transfer simulation, and determine how its inputs map to *SimFast21*.

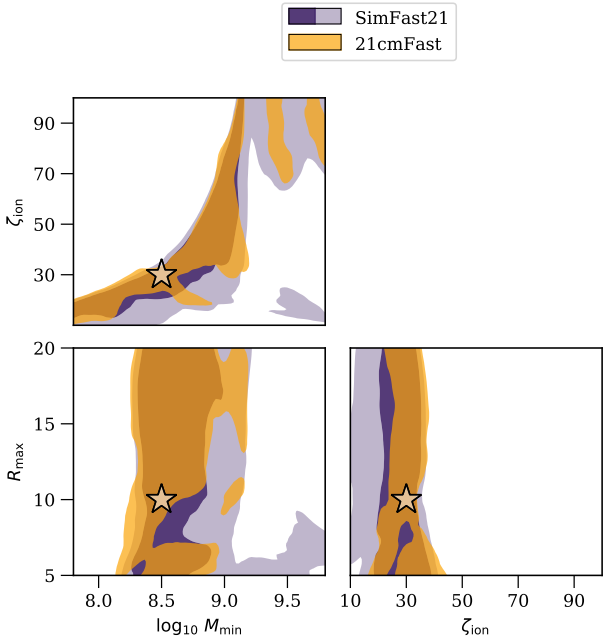


Figure 13. Mean squared error between emulated *SimFast21* power spectra and measured power spectra from both simulations. The star indicates the fixed simulation parameters. The orange contour indicates the regions where emulated *SimFast21* power spectra are within 30% MSE of the fully-simulated *21cmFAST* power spectra. For comparison, the purple contours indicate the same regions for comparing emulated *SimFast21* power spectra with fully-simulated *SimFast21* power spectra, using 30% MSE (lighter contour) and 15% MSE (darker contour).

8 CONCLUSIONS

Fast modelling of the 21cm signal will become a significant problem in analysing the huge datasets from upcoming radio interferometry experiments. Ideally, we would be able to compare numerical radiative transfer simulations with these data. Current numerical simulations are too slow to sample the input parameter space efficiently. Semi-numerical simulations are faster but can still only be used to constrain a small number of parameters. One potential solution to this problem is to replace current semi-numerical simulations with emulated models, reproducing the simulation outputs in a fraction of the original simulation time.

In this paper, we train and compare emulators using five different machine learning techniques. The two naive interpolation methods are not feasible as emulators, since they have either slow prediction times (linear interpolation model) or poor accuracy (near-

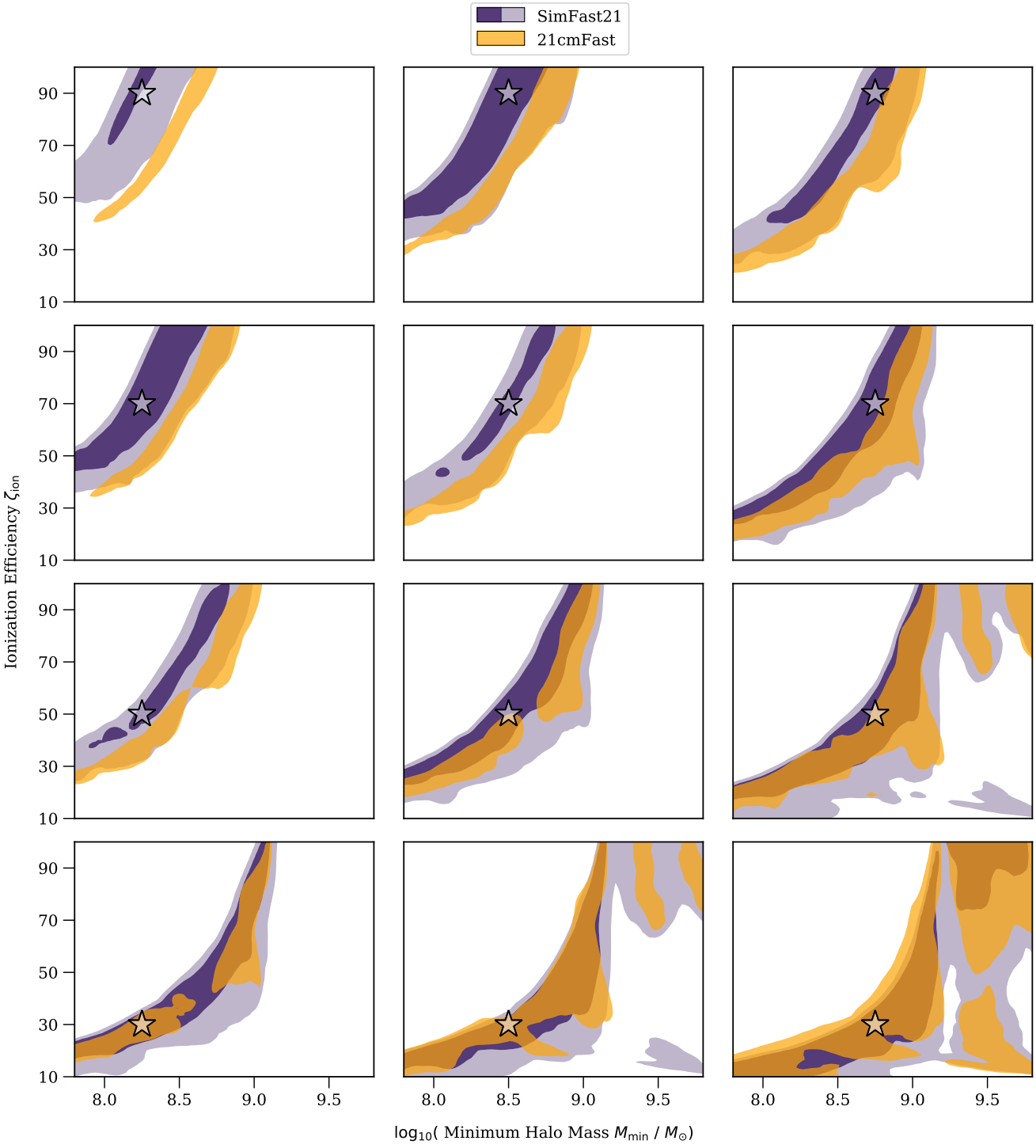


Figure 14. Similarity plots between emulated *SimFast21* power spectra and fully-simulated power spectra from *21cmFAST* (orange contours) and *SimFast21* (purple contours). In each panel, the white star indicates the scenario parameters of the fully-simulated power spectra. The orange contour shows the regions in which emulated *SimFast21* power spectra differ by less than 30% from the fully-simulated *21cmFAST* power spectra. The lighter- and darker-purple contours show the equivalent regions for comparing emulated *SimFast21* power spectra to fully-simulated *SimFast21* power spectra, within 30% and 15% MSE respectively. An offset can be seen for several of these different scenarios.

est neighbour interpolation model). Of the three more sophisticated models, one model performs much better than the others: the multilayer perceptron. This trained model makes predictions of the outputs from 500 *SimFast21* simulations to within 4% mean squared error averaged across all output points, reducing the modelling time from around 3000 hours to less than a second. If CPU training time is not a factor, then the accuracy of the sparse Gaussian processes regression or support vector machine models could potentially be improved with deeper hyperparameter searches. However, given their already relatively long prediction times and the accurate performance of the multilayer perceptron, these models are unlikely to give an improvement over the three-layer multilayer perceptron.

Our emulators use redshift and k-scales as extra input dimensions. This makes the models more flexible but gives rise to less accurate emulation especially near the end of reionization at lower redshifts. We also use $\Delta T_\mathrm{b} = \frac{\delta T_\mathrm{b}}{\langle \delta T_\mathrm{b} \rangle} - 1$ as the target values of our emulators. This gives rise to sudden features in the power spectra near at the end of reionization and is harder to emulate than using $\delta T_\mathrm{b} - \langle \delta T_\mathrm{b} \rangle$.

We use our best emulator to determine a relationship between two different reionization algorithms, using *SimFast21* and a version of *21cmFAST* with non-default inputs. We find some noticeable offsets in which input parameters match the power spectra outputs of *SimFast21* with those of *21cmFAST*. We provide a graphical description of how this offset depends on location in parameter space, so that users could roughly determine which *SimFast21* input parameters should be used if the desired result is to match the 21cm power spectrum of an existing *21cmFAST* simulation. Although our results are for a version of *21cmFAST* with non-default inputs, this method has potential for bridging between fast semi-numerical simulations and more accurate three-dimensional radiative transfer code such as C^2-RAY (Mellema et al. 2006). However, Majumdar et al. (2014) noted that there can be a 25% difference between the power spectrum outputs of C^2-RAY and semi-numerical codes. Given this discrepancy, it is likely that mapping between numerical and semi-numerical simulations will be considerably more challenging and it may be necessary to emulate numerical codes directly.

ACKNOWLEDGEMENTS

Many thanks to Mario Santos and Andrei Mesinger for their helpful comments on a draft version of this paper. WDJ is supported by the Science and Technology Facilities Council (ST/M503873/1) and from the European Community through the DEDALE grant (contract no. 665044) within the H2020 Framework Program of the European Commission. CAW acknowledges financial support from the European Research Council under ERC grant number 638743-FIRSTDAWN (held by Jonathan Pritchard). FBA acknowledges support from the DEDALE grant, from the UK Science and Technology Research Council (STFC) grant ST/M001334/1, and from STFC grant ST/P003532/1.

REFERENCES

Abadi M., et al., 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org/
Ali Z. S., et al., 2015, ApJ, 809, 61
Alvarez M. A., Abel T., 2012, ApJ, 747, 126
Barber C. B., Dobkin D. P., Huhdanpaa H., 1996, ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, 22, 469
Barkana R., Loeb A., 2001, Phys. Rep., 349, 125
Datta A., Bowman J. D., Carilli C. L., 2010, ApJ, 724, 526
DeBoer D. R., et al., 2017, PASP, 129, 045001
Furlanetto S. R., Zaldarriaga M., Hernquist L., 2004, The Astrophysical Journal, 613, 1
Furlanetto S. R., Oh S. P., Briggs F. H., 2006, Phys. Rep., 433, 181
Gillet N., Mesinger A., Greig B., Liu A., Ucci G., 2018, preprint, (arXiv:1805.02699)
Greig B., Mesinger A., 2015, MNRAS, 449, 4246
Greig B., Mesinger A., 2018, in Jelić V., van der Hulst T., eds, IAU Symposium Vol. 333, Peering towards Cosmic Dawn. pp 18–21 (arXiv:1705.03471), doi:10.1017/S1743921317011103
Greig B., Mesinger A., Pober J. C., 2016, MNRAS, 455, 4295
Hassan S., Davé R., Finlator K., Santos M. G., 2016, MNRAS, 457, 1550
Hassan S., Davé R., Finlator K., Santos M. G., 2017, MNRAS, 468, 122
Hutter A., 2018, MNRAS, 477, 1549
Jones E., Oliphant T., Peterson P., et al., 2001, SciPy: Open source scientific tools for Python, http://www.scipy.org/
Kern N. S., Liu A., Parsons A. R., Mesinger A., Greig B., 2017, ApJ, 848, 23
Kingma D. P., Ba J., 2014, preprint, (arXiv:1412.6980)
Kulkarni G., Choudhury T. R., Puchwein E., Haehnelt M. G., 2016, MNRAS, 463, 2583
Liu A., Pritchard J. R., Allison R., Parsons A. R., Seljak U., Sherwin B. D., 2016, Phys. Rev. D, 93, 043013
Lupton R. H., Gunn J. E., Szalay A. S., 1999, AJ, 118, 1406
Majumdar S., Mellema G., Datta K. K., Jensen H., Choudhury T. R., Bharadwaj S., Friedrich M. M., 2014, MNRAS, 443, 2843
Majumdar S., Pritchard J. R., Mondal R., Watkinson C. A., Bharadwaj S., Mellema G., 2018, MNRAS, 476, 4007
McKay M. D. e. a., 1979, Technometrics, pp vol. 21, no. 2, pp. 239âĂŞ245
McQuinn M., Lidz A., Zahn O., Dutta S., Hernquist L., Zaldarriaga M., 2007, MNRAS, 377, 1043
Mellema G., Iliev I. T., Alvarez M. A., Shapiro P. R., 2006, New Astron., 11, 374
Mellema G., et al., 2013, Experimental Astronomy, 36, 235
Mesinger A., Furlanetto S., 2007, The Astrophysical Journal, 669, 663
Mesinger A., Furlanetto S., Cen R., 2011, MNRAS, 411, 955
Patil A. H., et al., 2017, ApJ, 838, 65
Pedregosa F., et al., 2011, Journal of Machine Learning Research, 12, 2825
Pober J. C., Greig B., Mesinger A., 2016, MNRAS, 463, L56
Press W. H., Schechter P., 1974, ApJ, 187, 425
Pritchard J. R., Loeb A., 2012, Reports on Progress in Physics, 75, 086901
Rasmussen Williams 2006, Gaussian Processes for Machine Learning. The MIT Press
Rumelhart D. E., Hinton G. E., Williams R. J., 1986, Nature, 323, 533
Santos M. G., Ferramacho L., Silva M. B., Amblard A., Cooray A., 2010, MNRAS, 406, 2421
Schmit C. J., Pritchard J. R., 2018, MNRAS, 475, 1213
Semelin B., Eames E., Bolgar F., Caillat M., 2017, MNRAS, 472, 4508
Sheth R. K., Mo H. J., Tormen G., 2001, MNRAS, 323, 1
Shimabukuro H., Semelin B., 2017, MNRAS, 468, 3869
Shimabukuro H., Yoshiura S., Takahashi K., Yokoyama S., Ichiki K., 2016, MNRAS, 458, 3003
Sobacchi E., Mesinger A., 2014, MNRAS, 440, 1662
Tingay S. J., et al., 2013, Publ. Astron. Soc. Australia, 30, e007
Titsias M., 2009, in Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics. PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, pp 567–574, http://proceedings.mlr.press/v5/titsias09a.html
Watkinson C. A., Majumdar S., Pritchard J. R., Mondal R., 2017, MNRAS, 472, 2436
Watkinson C. A., Giri S. K., Ross H. E., Dixon K. L., Iliev I. T., Mellama G., Pritchard J. R., 2018, preprint, (arXiv:1808.02372)
Werbos P. J., 1982, in Drenick R. F., Kozin F., eds, System Modeling and Optimization. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 762–770

Zahn O., Lidz A., McQuinn M., Dutta S., Hernquist L., Zaldarriaga M., Furlanetto S. R., 2007, The Astrophysical Journal, 654, 12
Zel'dovich Y. B., 1970, A&A, 5, 84

Appendices

A SIMULATION PARAMETERS

We list all relevant user-changeable parameters used for all *21cmFAST* and *SimFast21* simulations in this paper. For further descriptions of these parameters see Mesinger et al. (2011) and Santos et al. (2010). We exclude parameters relating to spin temperature calculations since we did no use this functionality.

A.1 Cosmology

Parameter	Value
σ_8	0.810
Hubble h	0.710
$\Omega_{\rm M}$	0.270
Ω_Λ	0.730
$\Omega_{\rm b}$	0.046
$\Omega_{\rm n}$	0.0
$\Omega_{\rm k}$	0.0
$\Omega_{\rm R}$	0.0
$\Omega_{\rm tot}$	1.0
$Y_{\rm He}$	0.245
$n_{\rm s}$	0.960
Sheth-Tormen b	0.34
Sheth-Tormen c	0.81
Helium II $z_{\rm reion}$	3
Maximum Redshift	17.00
Minimum Redshift	8.00
Redshift Step	1.50
Simulation Length	500.00
Star Formation Rate	0.025
Velocity Component	3
Critical Overdensity	1.680

A.2 *SimFast21* reionization parameters

Parameter Name	Value
use_camb_matterpower	False
use_fcoll	True
halo_Rmax	40
halo_Mmin	Various
Ion_eff	Various
bubble_Rmax	Various
use_Lya_xrays	False

A.3 *21cmFAST* reionization parameters

Parameter Name	Value
ION_M_MIN	Various
ION_Tvir_MIN	-1 (off)
HII_EFF_FACTOR	Various
EFF_FACTOR_PL_INDEX	0
R_BUBBLE_MAX	Various

A.4 Other *21cmFAST* parameters

Parameter Name	Value
P_CUTOFF	0
M_WDM	2
g_x	1.5
INHOMO_RECO	0
ALPHA_UVB	5
t_STAR	0.5
EVOLVE_DENSITY_LINEARLY	0
SMOOTH_EVOLVED_DENSITY_FIELD	1
R_smooth_density	0.2
SECOND_ORDER_LPT_CORRECTIONS	0
HII_ROUND_ERR	1e-3
FIND_BUBBLE_ALGORITHM	1
R_BUBBLE_MIN	L_FACTOR*1
USE_HALO_FIELD	0
N_POISSON	-1
T_USE_VELOCITIES	1
MAX_DVDR	0.2
DIMENSIONAL_T_POWER_SPEC	0
DELTA_R_FACTOR	1.1
DELTA_R_HII_FACTOR	1.1
R_OVERLAP_FACTOR	1.0
DELTA_CRIT_MODE	1
HALO_FILTER	0
HII_FILTER	1
OPTIMIZE	0
OPTIMIZE_MIN_MASS	1e11
SIZE_RANDOM_SEED	-23456789
LOS_RANDOM_SEED	-123456789
USE_TS_IN_21CM	0
CLUMPING_FACTOR	>50
Pop	2
Pop2_ion	4361

This paper has been typeset from a TEX/LATEX file prepared by the author.