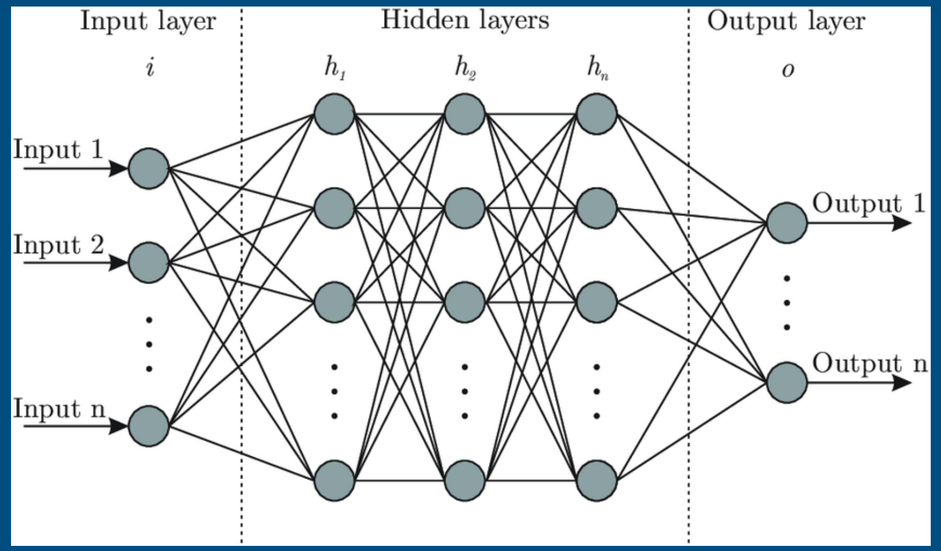
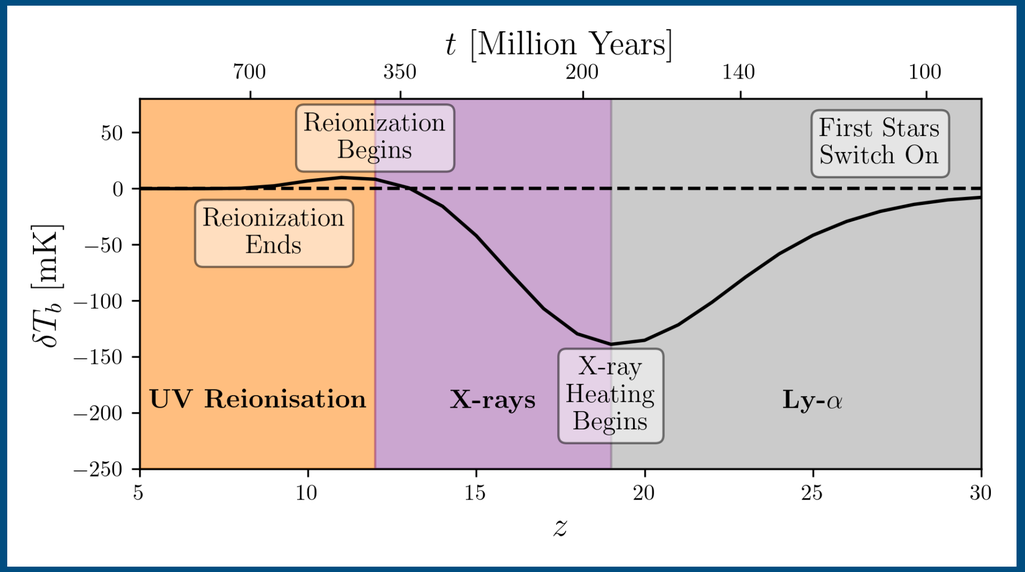
Machine Learning for 21-cm Cosmology

Harry Bevins
Final year PhD student at Cambridge

The Plan

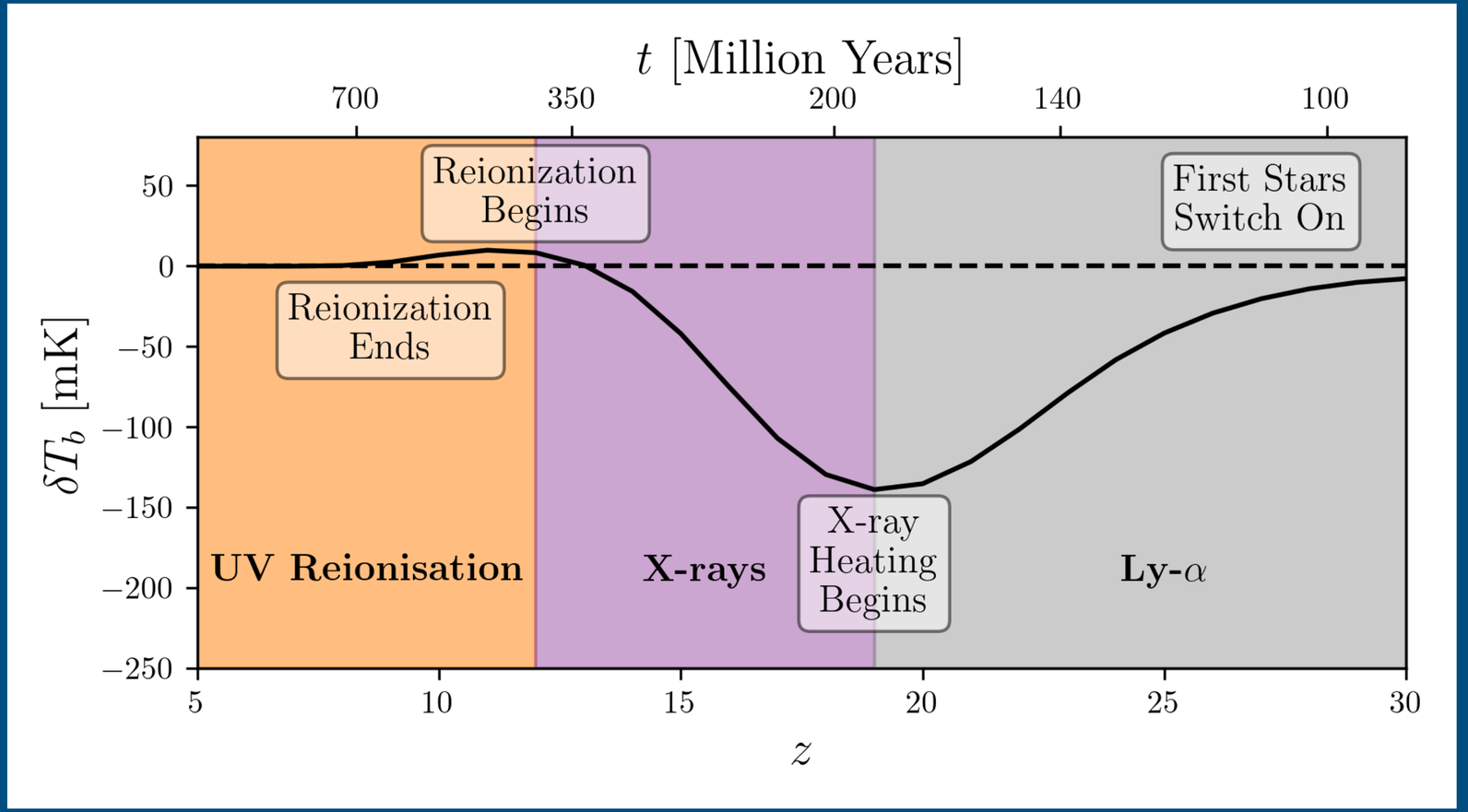
- Why are we interested in emulating the 21-cm signal?
- What is a neural network?
- Examples in 21-cm Cosmology





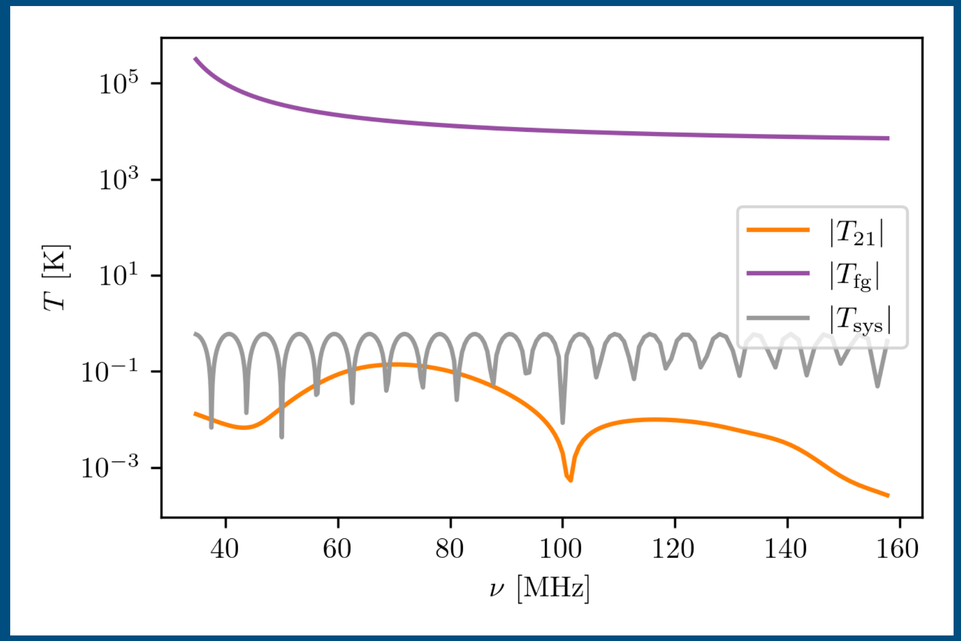
Why are we interested in emulating the 21-cm signal?

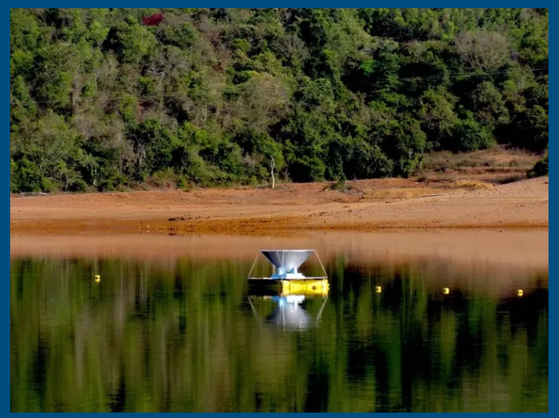
21-cm Cosmology



21-cm Cosmology

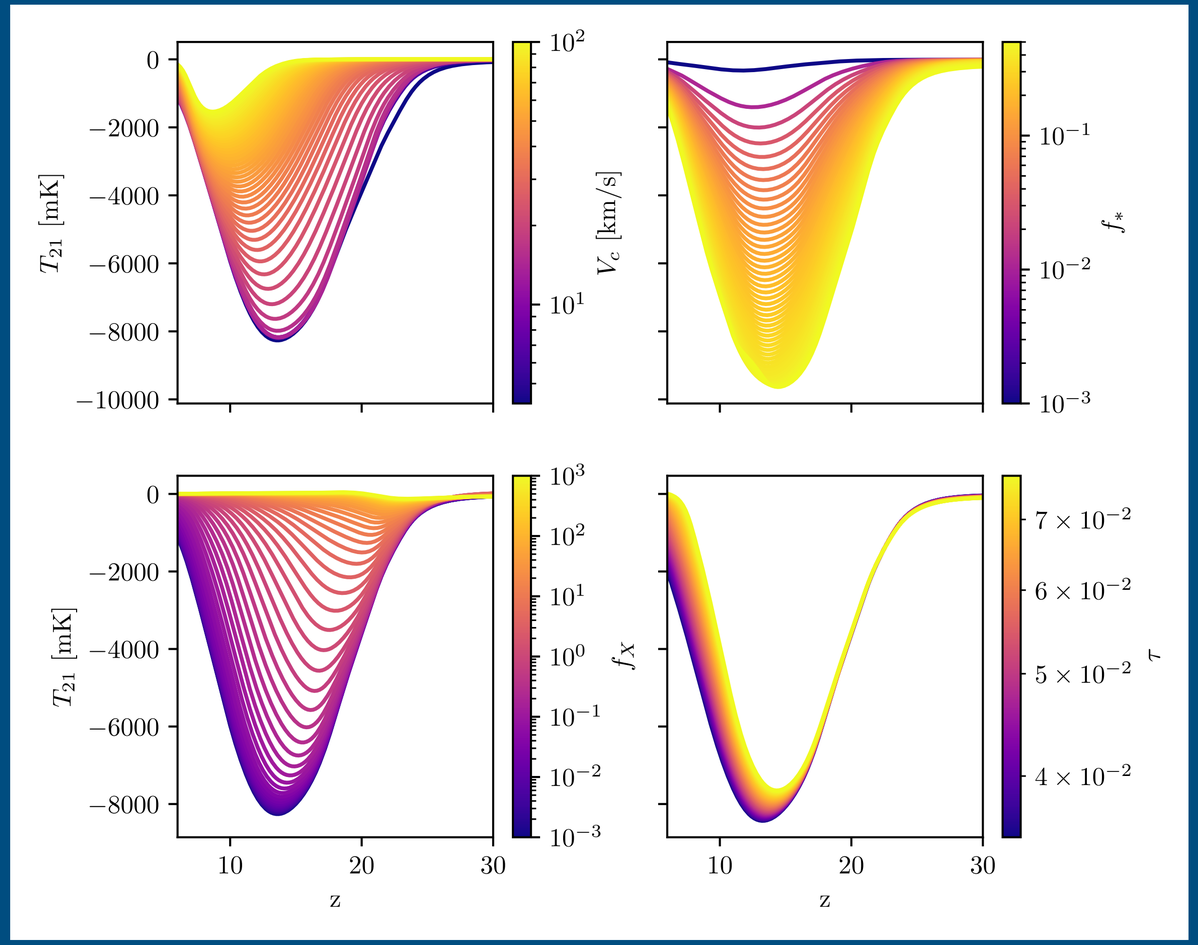
- Data from a 21-cm experiment includes emission from our own galaxy, other nearby galaxies, systematics, instrumental noise and radio frequency interference
- We need to be able to model each of these different components
- Often we can approximate the 21-cm signal with a gaussian absorption feature
- However, ideally we would like to physically motivate our model





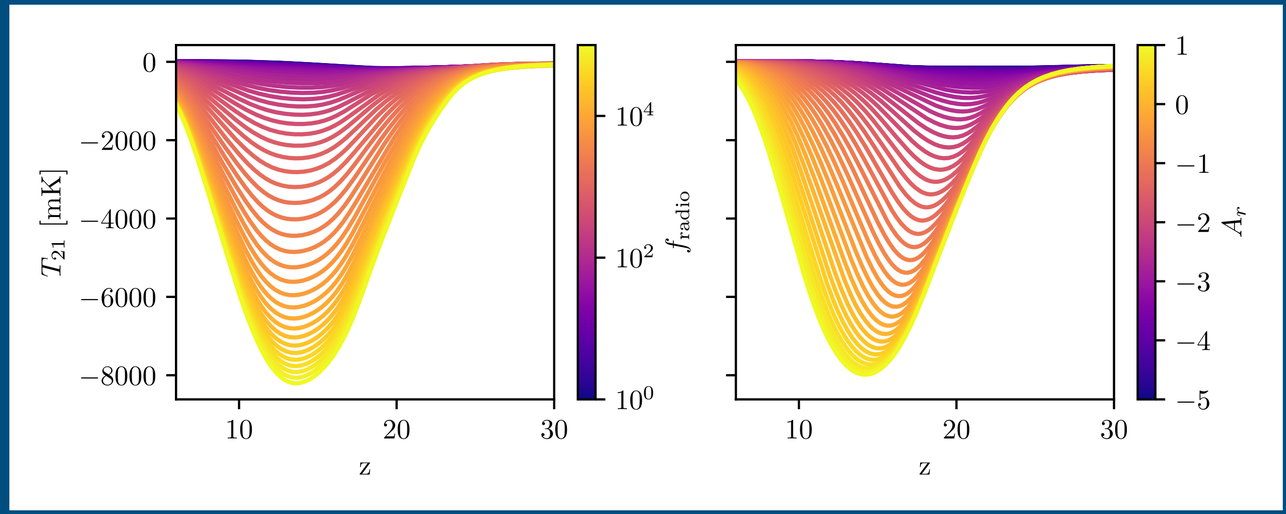
21-cm Cosmology

- The signal is mediated by complex astrophysical processes
- Star formation effects the depth and position of the minimum
- X-ray heating effects the timing of the minimum
- The CMB optical depth effects the timing of reionisation



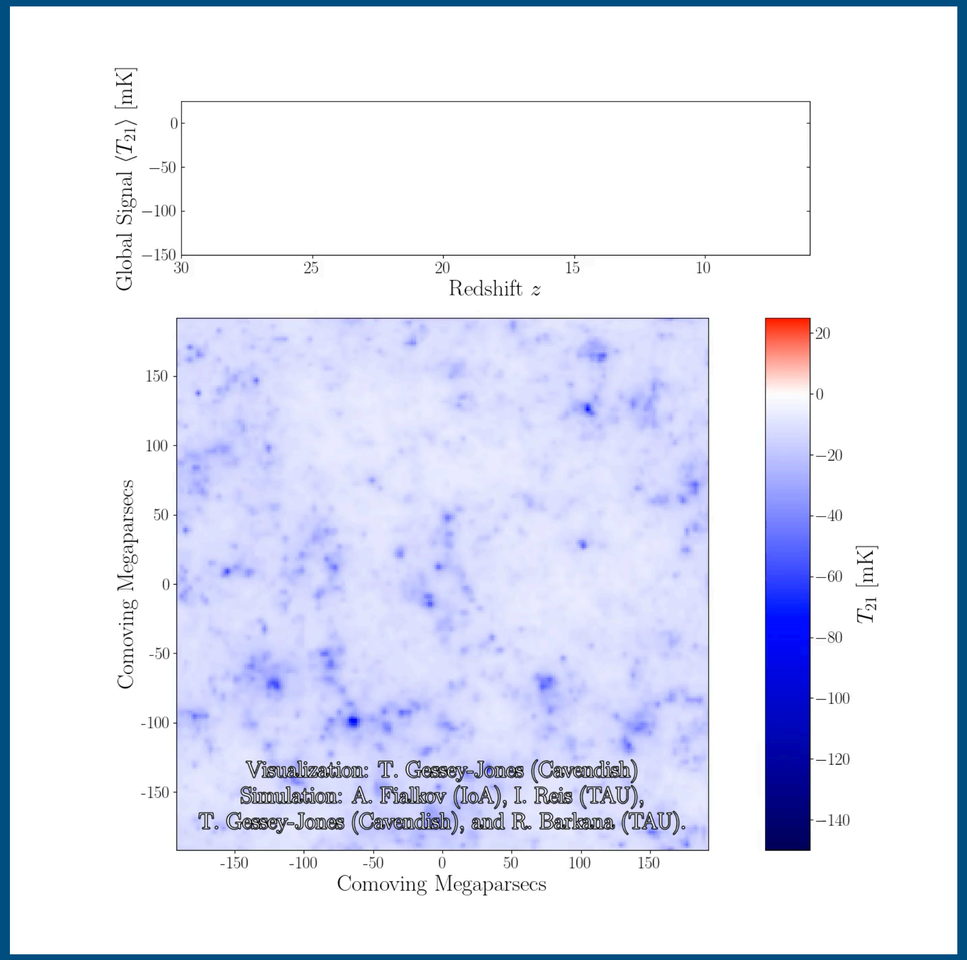
21-cm Cosmology

- Exotic processes
- Interactions between dark matter and baryons
- Or increased radio background from galaxies
- Increased radio from cosmic strings



The need for emulators

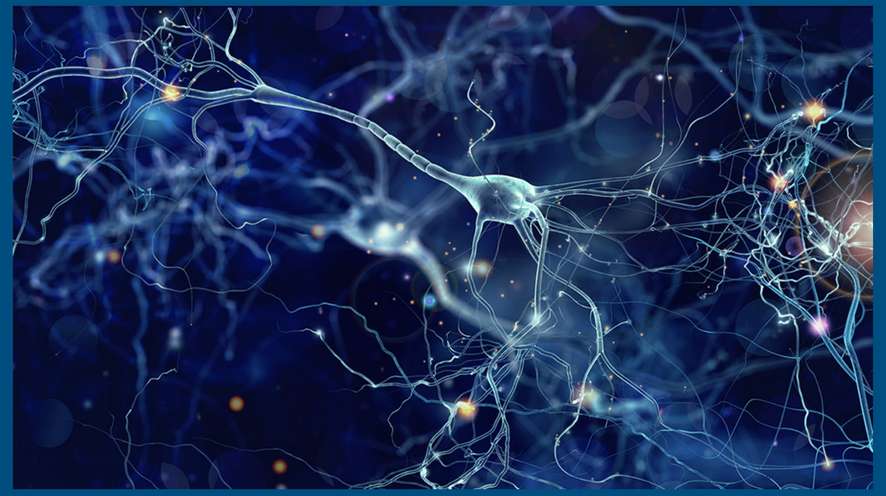
- Semi-numerical simulations are slow
- A few hours per signal
- This is impractical if we want to fit physically motivated models to data sets
- Look towards neural networks for fast signal emulation

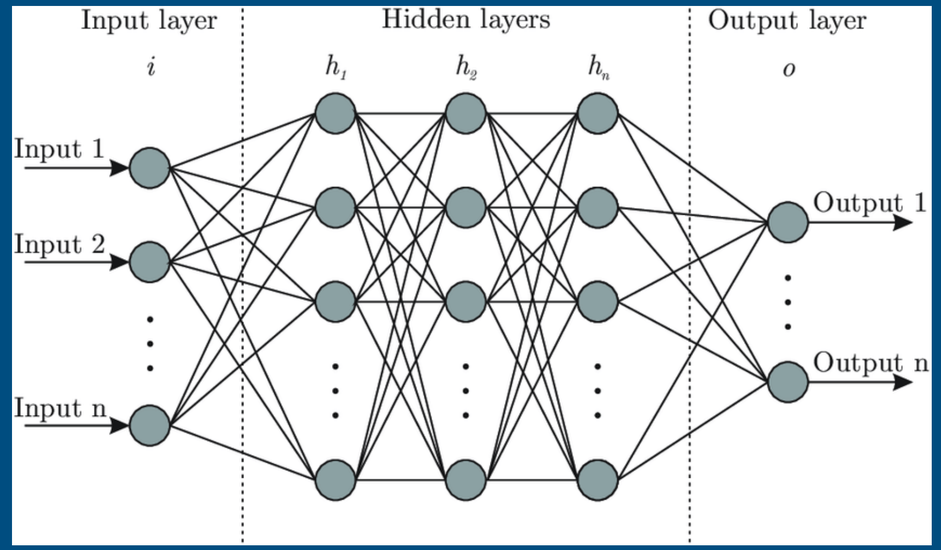


What is a neural network?

Artificial Neural Networks

- Motivated by the structure of the brain
- The brain consists of many different neurons
- These neurons feed into each other and perform calculations based on the outputs of proceeding neurons
- If we chain a bunch of neurons together we can make decisions and read these slides



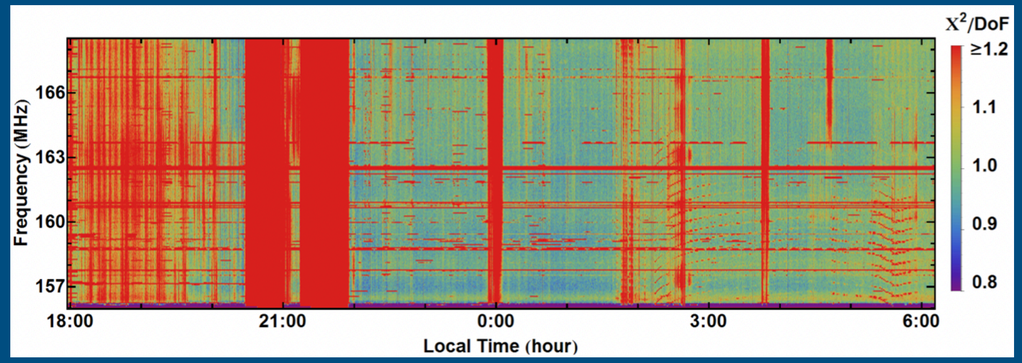


Artificial Neural Networks

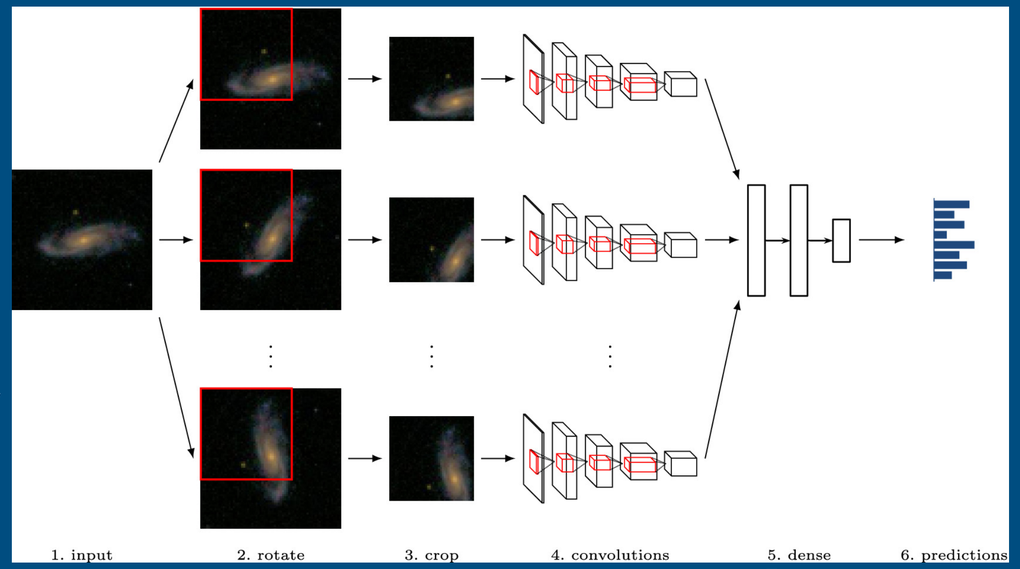
- Neural Networks (NNs) consist of artificial neurons that work together to solve complex problems

- NNs are used for lots of different things such as ***handwriting recognition*** and ***face detection***

- In radio astronomy different varieties of NN are used for ***signal modelling***, ***radio frequency interference filtering*** and ***galaxy classification*** among other applications



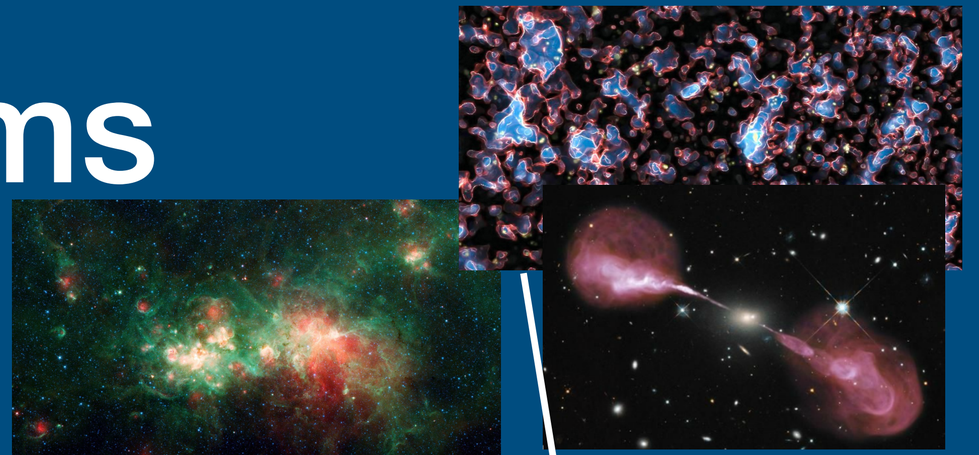
Liu and Shaw 2019 [1907.08211]



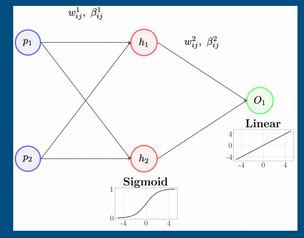
Dieleman et al 2015 [1503.07077]

Regression Problems

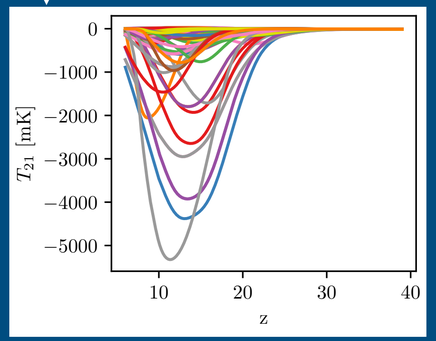
- We are interested in solving a regression problem with our NN to create a signal emulator
- In a regression problem we transform the inputs, θ, to our NN into a function of those features, $f(\theta)$
- Here $\theta = \{f_*, V_c, f_X, \tau, f_{\rm radio}\}$ and $f(\theta) = T_{21}$
- In contrast a classification problem outputs probabilities rather than functions e.g. what's the probability that a galaxy has spiral arms



$\theta = \{f_*, V_c, f_X, \tau, f_{\rm radio}\}$

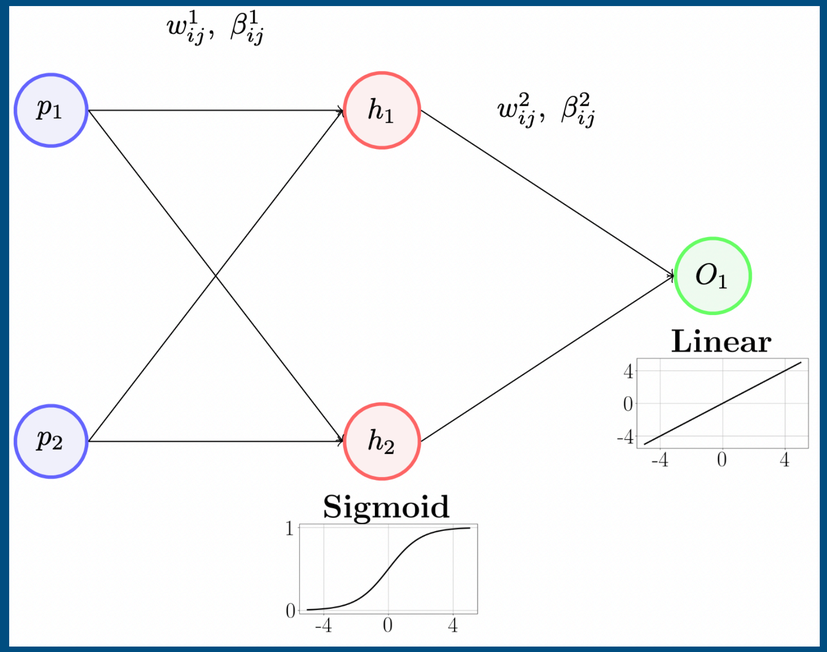


$f(\theta) = T_{21} =$



The components of a Neural Networks

- ***The inputs:*** these are our astrophysical parameters $\theta = \{p_1, p_2\}$
- ***The output:*** this is our function $f(\theta)$
- ***The hidden layers:*** intermittent steps between the inputs and outputs
- ***The weights and biases:*** these are tuneable parameters along each edge of the network that help the network transform the input to the output
- ***The activation functions:*** these introduce non-linear structure into the network



Here the superscript corresponds to the set of weights and biases between layers and the subscript to the nodes being connected

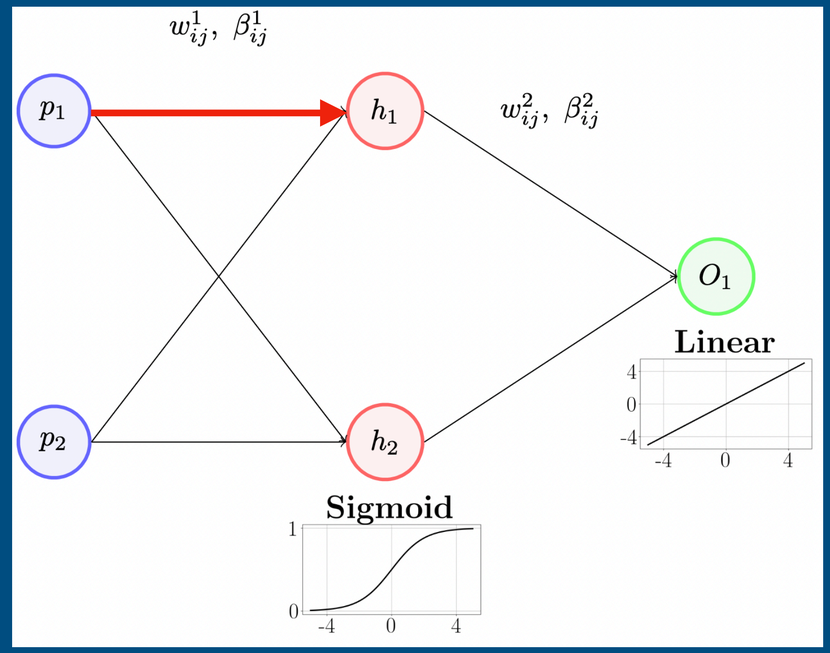
The path through the neural network

- If we consider the path highlighted in red between p_1 and h_1 then the input to h_1 is given by $w_{11}^1 p_1 + \beta_{11}^1$
- The value at h_1 is given by the weighted and scaled inputs from each proceeding node such that

$$h_1 = w_{11}^1 p_1 + w_{21}^1 p_2 + \beta_{11}^1 + \beta_{21}^1$$

- The input to h_1 is then passed through the sigmoid activation function giving

$$h_1' = \frac{1}{1 + \exp(-h_1)}$$



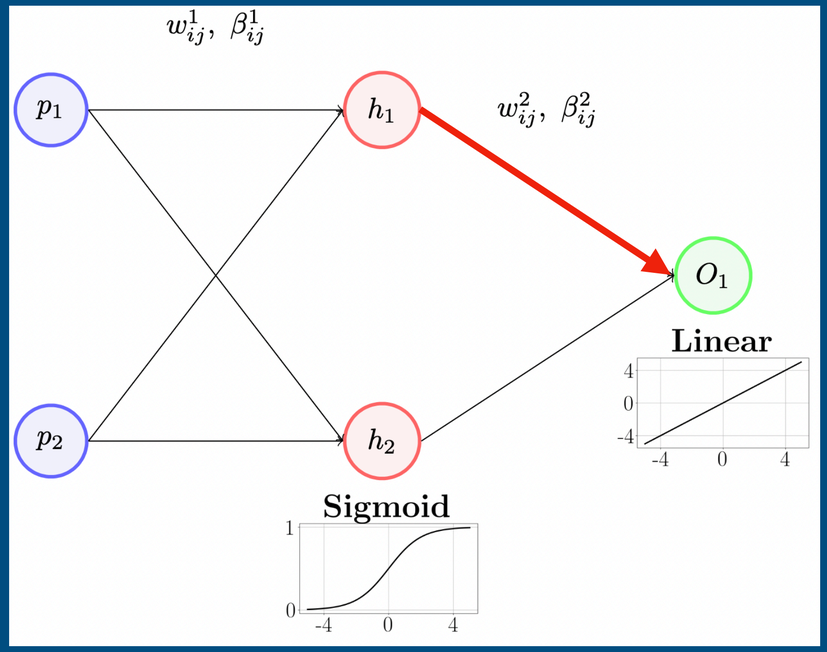
The path through the neural network

- h_1' is then passed to the output node and appropriately weighted, scaled and summed with the output of the other hidden nodes

$$o_1 = w_{11}^2 h_1' + w_{21}^2 h_2' + \beta_{11}^2 + \beta_{21}^2$$

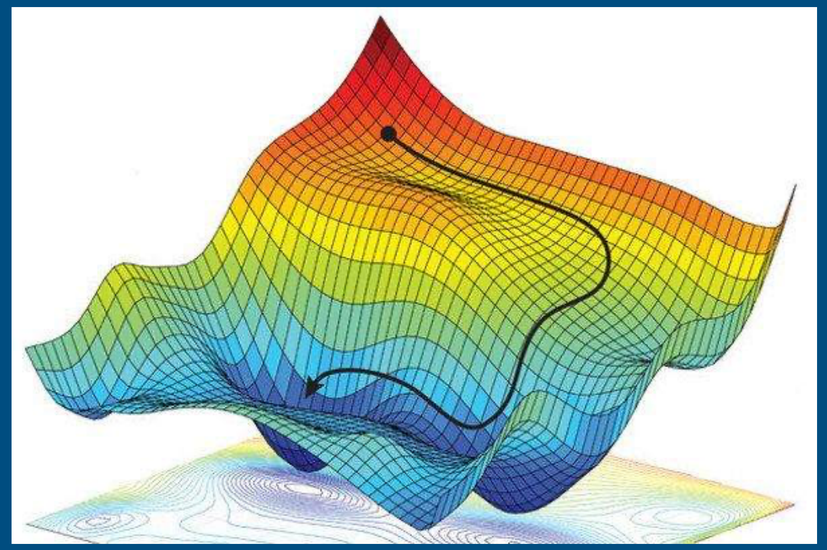
- The output is then passed through a linear activation function such that the network output is $o_1' = o_1$

- This quickly becomes more complicated as we add more and more inputs, outputs and hidden layers



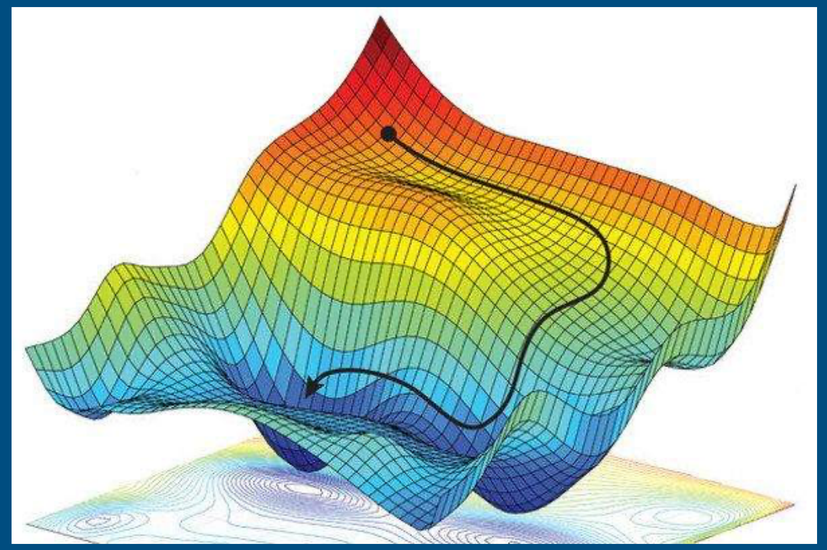
Supervised Learning

- The problem is how to set the weights and biases of the network
- In essence we have to teach our network how to make predictions by giving it training data
- Training data comprises a set of N example inputs with known outputs generated from semi-numerical simulations



Supervised Learning

- At each step (or epoch) in the training we provide the network with the inputs and evaluate the difference between what it outputs and what we expect
- We do this with a “loss function”
- Depending on the value of the loss function at each epoch we adjust the weights and biases to improve the accuracy of the emulation at the next epoch

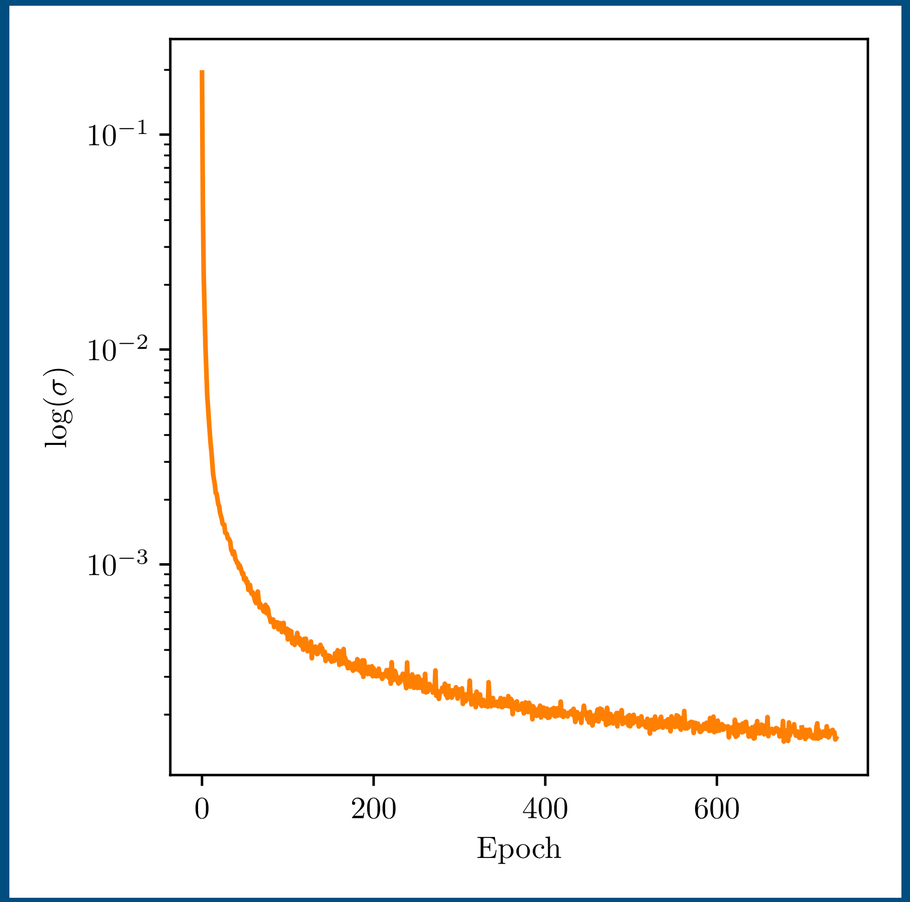


The loss function

- The loss function describes the surface we are optimising on and a common example is

$$\sigma(w,\beta) = \frac{1}{N}\sum_{k}(y_{\text{true,k}} - y_{\text{pred,k}}(w,\beta))^2$$

- Here $y_{\text{true,k}}$ is what we expect the network to predict given a set of inputs and $y_{\text{pred,k}}(w,\beta)$ is what it returns



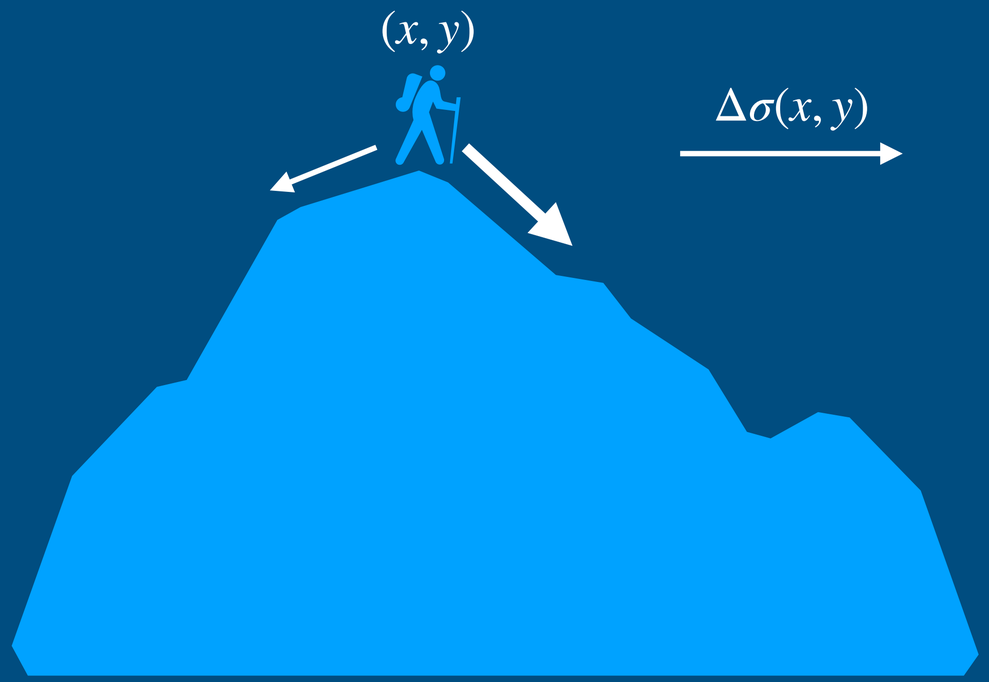
Gradient Descent

- In a gradient descent algorithm we optimise the loss by moving w and β in the direction of the negative gradient at each epoch e.g.

$$\mathbf{w}' = \mathbf{w} - \gamma \nabla \sigma$$

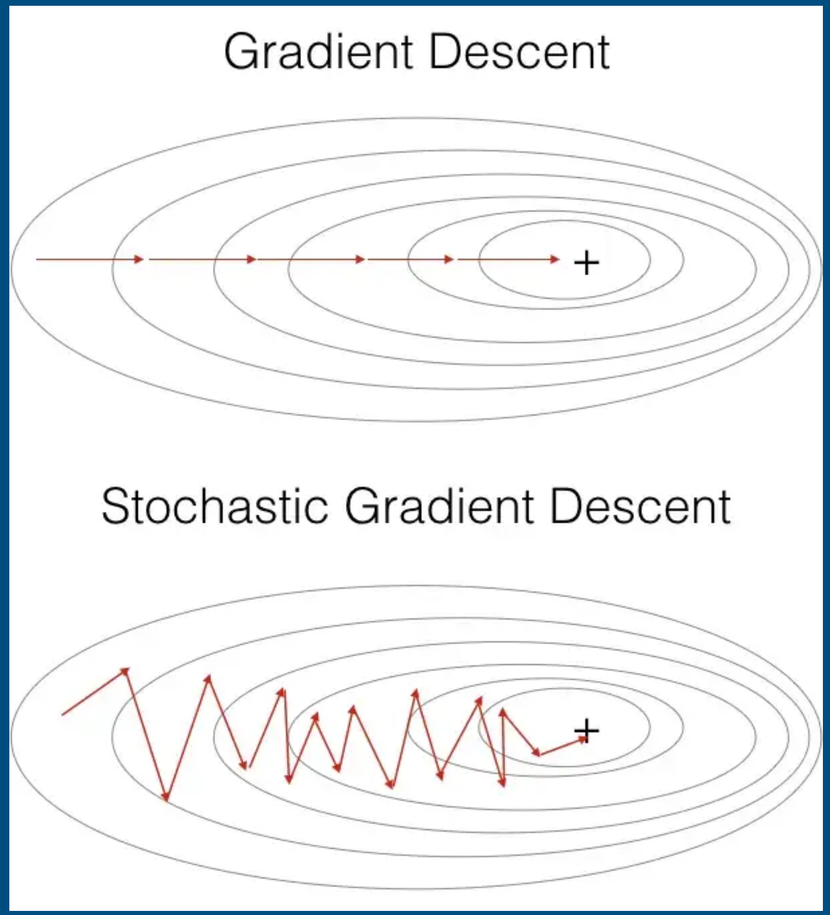
- γ is a tuneable parameter known as the learning rate

- Analogous to a mountaineer stuck at the summit in foggy weather and attempting to descend on the quickest possible path which is defined by the steepness of the slope



Stochastic Gradient Descent

- In practice it is often too time consuming and the parameter space is too complex to calculate the gradient of our loss function for all of our training data
- We therefore use ***stochastic*** gradient descent and train on batches of randomly chosen data at each epoch
- But how do we calculate the gradient of such a complex function?

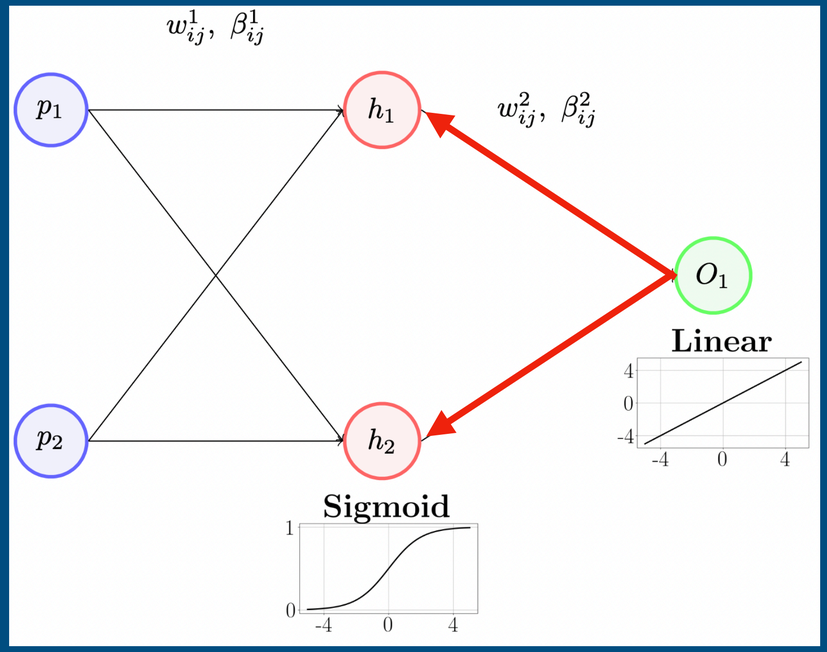


The backpropagation algorithm

- The output is given by $o_1 = w_{11}^2 h_1' + w_{21}^2 h_2' + \beta_{11}^2 + \beta_{21}^2$
- Knowing this and our loss function we can calculate via the chain rule

$$\frac{d\sigma}{dw_{11}^2} = \frac{d\sigma}{do_1}\frac{do_1}{dw_{11}^2}$$

- This then allows us to update according to $w_{11}^2 - \gamma \frac{d\sigma}{dw_{11}^2}$
- We can do a similar thing for $w_{21}^2, \beta_{11}^2, \beta_{21}^2$ and adjust them appropriately



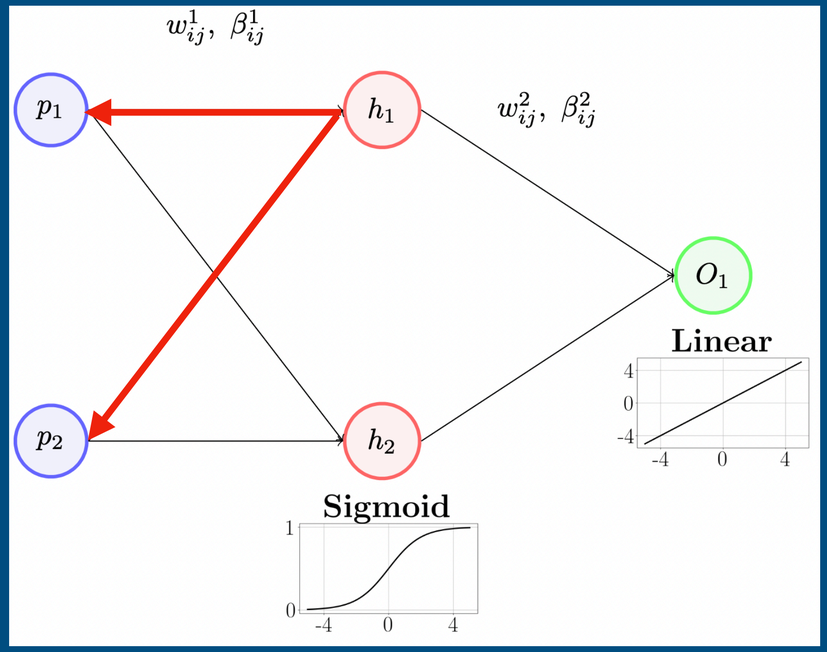
The backpropagation algorithm

- $w^1_{11} \rightarrow h_1 \rightarrow h'_1 \rightarrow o_1$
- So we can calculate

$$\frac{d\sigma_1}{dw^1_{11}} = \frac{do_1}{dh'_1}\frac{dh'_1}{dh_1}\frac{dh_1}{dw^1_{11}}$$

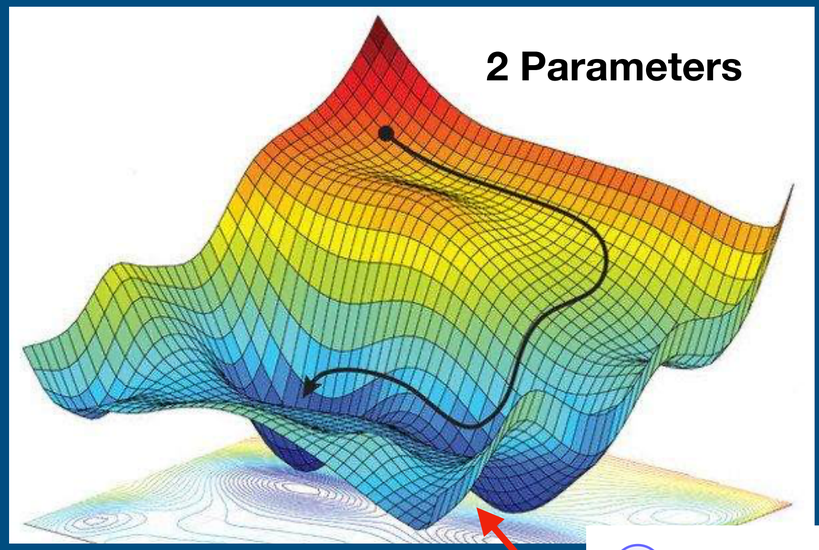
  and appropriately modify w^1_{11}

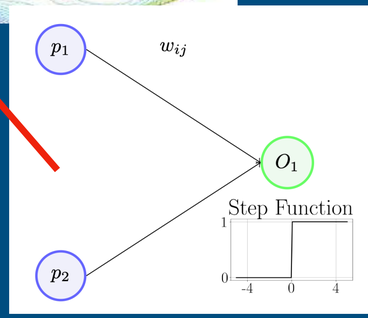
- Through this process we arrive at a way to modify all of the hyper-paremeters in our network to improve its accuracy at each epoch

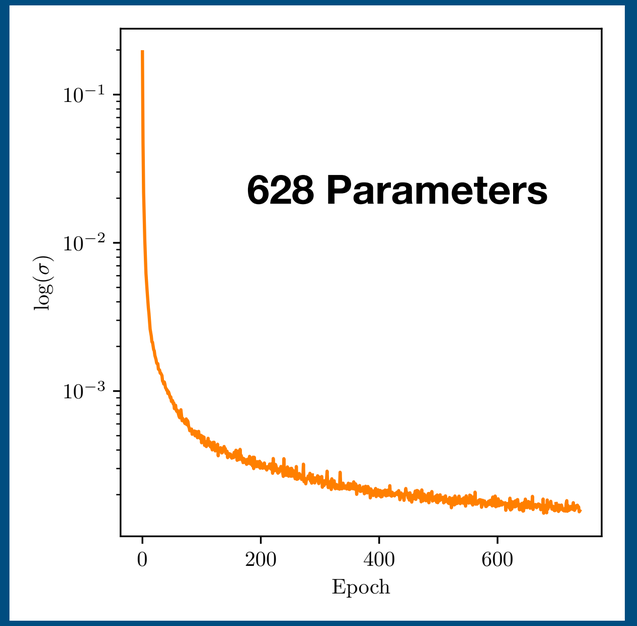


Training the network

- We are trying to find the global minimum (or a good enough local minimum) in a very high dimensional space
- We therefore have to repeat the process described many many times stepping through the parameter space to minimise $\sigma(w, \beta)$

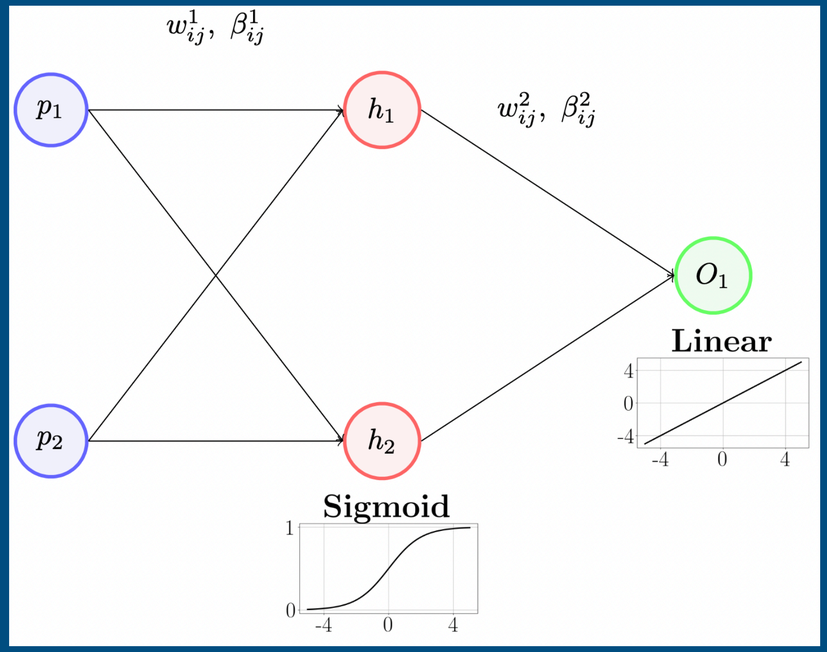






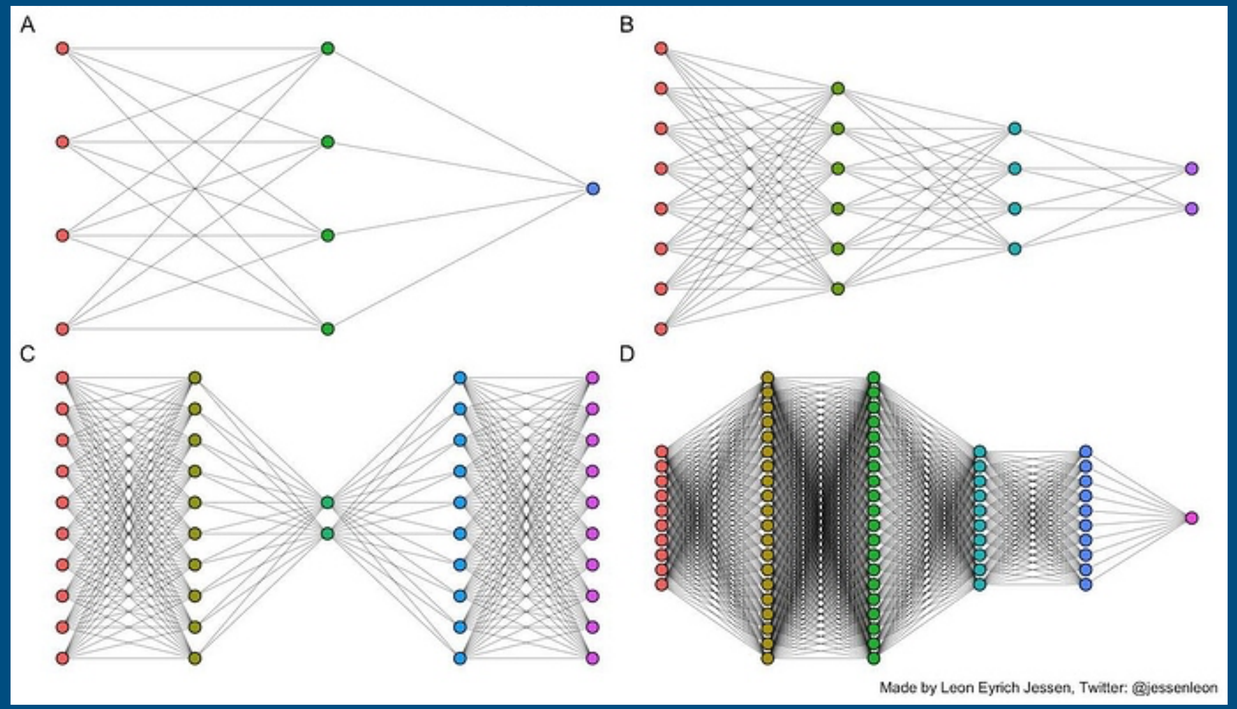
The summary

- Neural networks transform inputs, θ, into outputs which are functions, $f(\theta)$
- The NNs are comprised of hidden layers connected by weighted and scaled transformations
- Each node in each layer in the network has an activation function that is designed to introduce non-linearity into the estimator
- We adjust the weights and biases in the network using the backpropagation algorithm, stochastic gradient descent and a loss function



Other considerations

- The network size is often an important consideration: too big and we can loose the ability to generalise but too small and it won’t be accurate enough
- The learning rate is often a non-trivial choice that needs to be tested
- The choice of loss function can effect the emulation
- The inputs and outputs need to be preprocessed to improve the network accuracy



Examples in 21-cm Cosmology

21cmGEM

Cohen et al. 2020 [1910.06274]

- 21cmGEM uses a series of neural networks and a decision tree to model the 21-cm signal
- A series of networks estimate different astrophysical quantities and the decision tree works out the number of expected turning points in the signal
- The final network estimates the signal based on the outputs of the other network
- The emulator uses a Principle Component Analysis
- Written in Matlab
- Code is available here https://people.ast.cam.ac.uk/~afialkov/

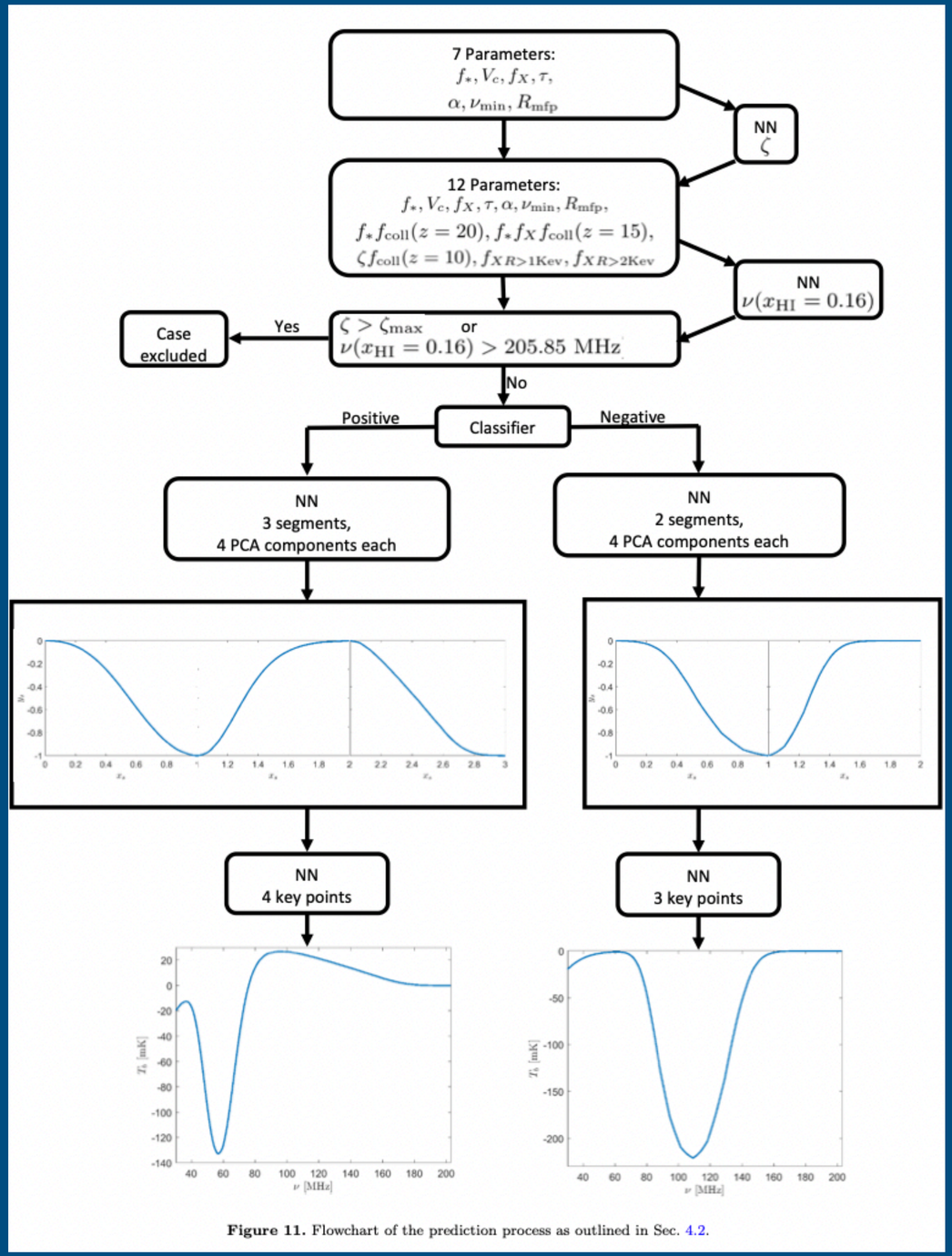
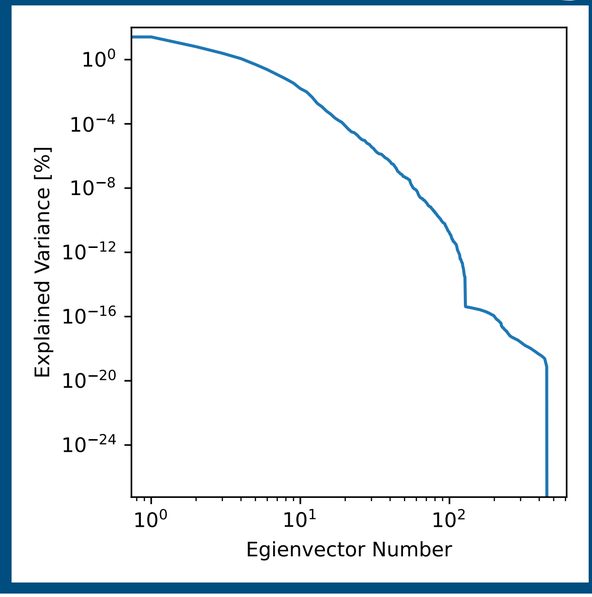
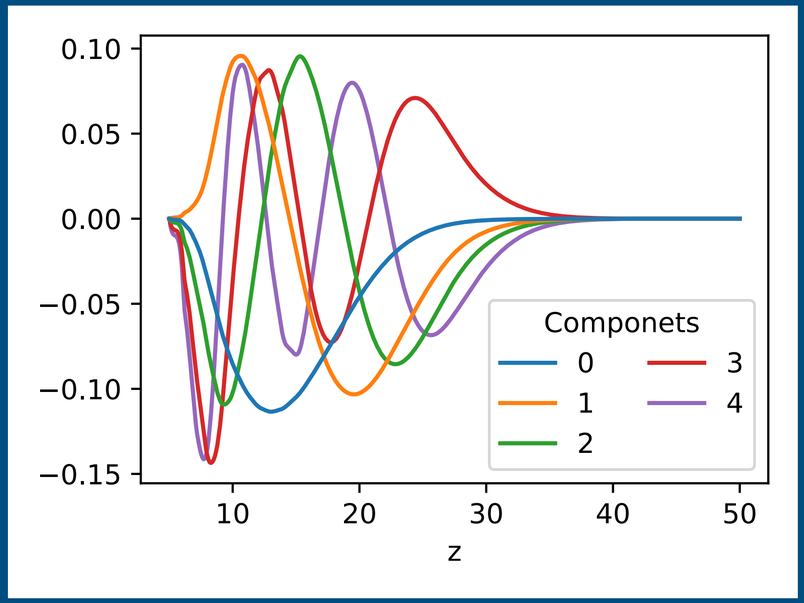


Figure 11. Flowchart of the prediction process as outlined in Sec. 4.2.

21cmGEM - Principle Component Analysis

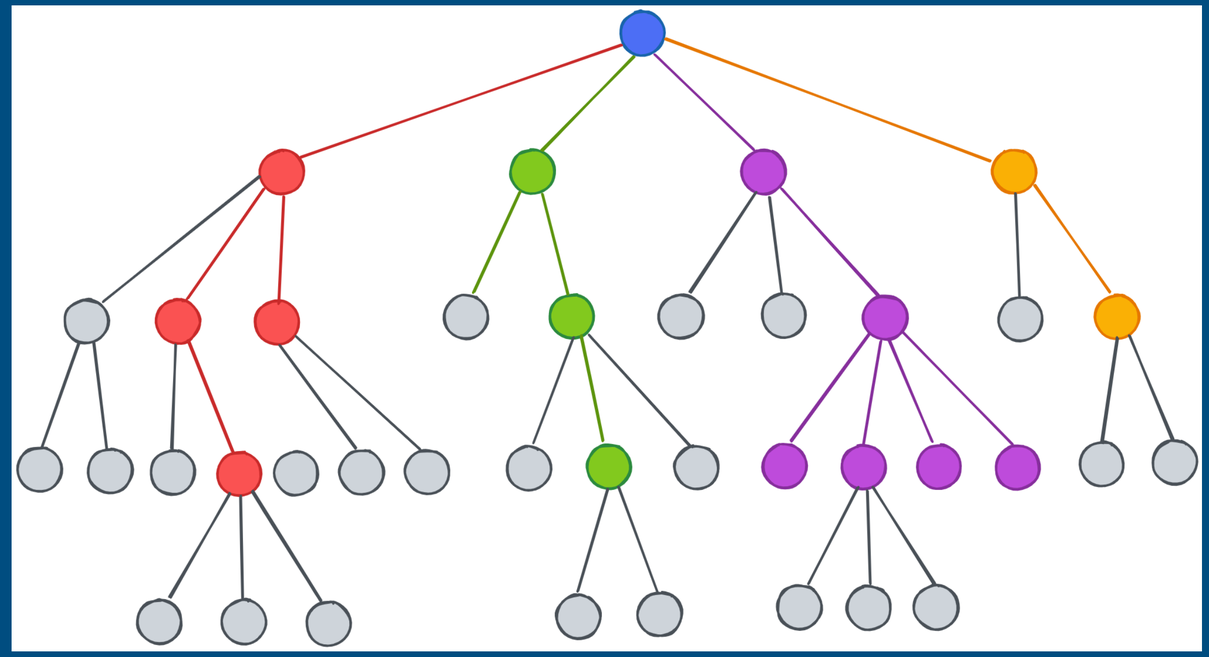
- Form of dimensionality reduction which means that the network only needs to learn a few outputs rather than a large set of temperatures
- Decompose the training data into a series of eigenvectors
- We find that only a handful of these eigenvectors account for the structure in the 21-cm signal
- We can largely reconstruct each signal from this handful of vectors multiplied by some coefficients
- 21cmGEM estimates these coefficients for the principle axis rather than estimating directly $T_{21}(z)$





21cmGEM - Decision Trees

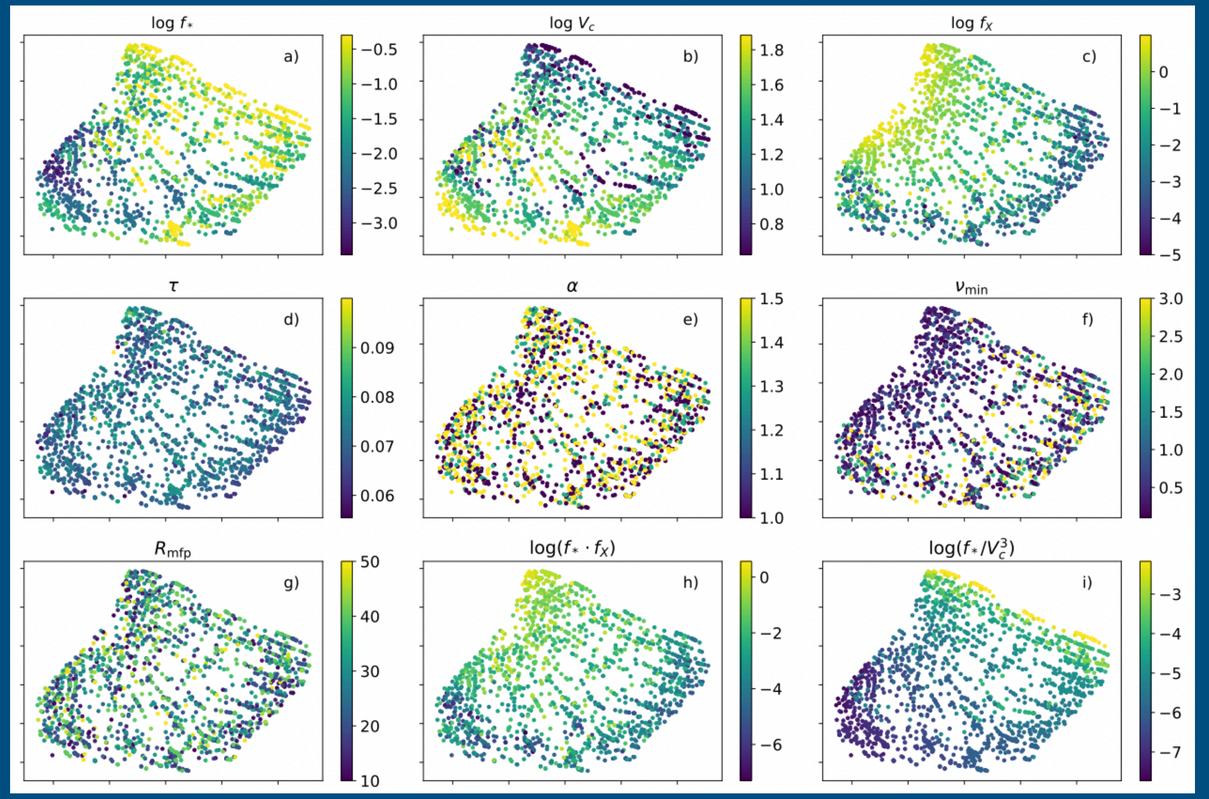
- Essentially a series of if-else statements that make classifications
- In 21cmGEM decision tress are used to determine whether the signals feature emission or not based on θ
- The decision ultimately effects the preprocessing of the data



21cmVAE

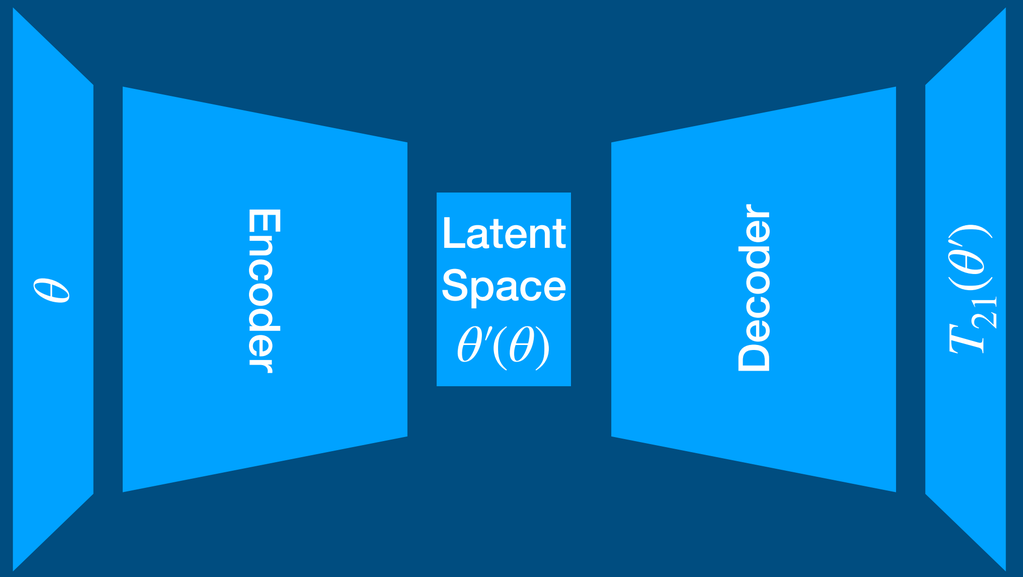
Bye et al. 2022 [2107.05581]

- 21cmVAE features two different types of emulator: a fully connected neural network and an autoencoder
- Takes seven astrophysical parameters as inputs
- Outputs the 21-cm temperature at fixed redshifts



21cmVAE - Autoencoders

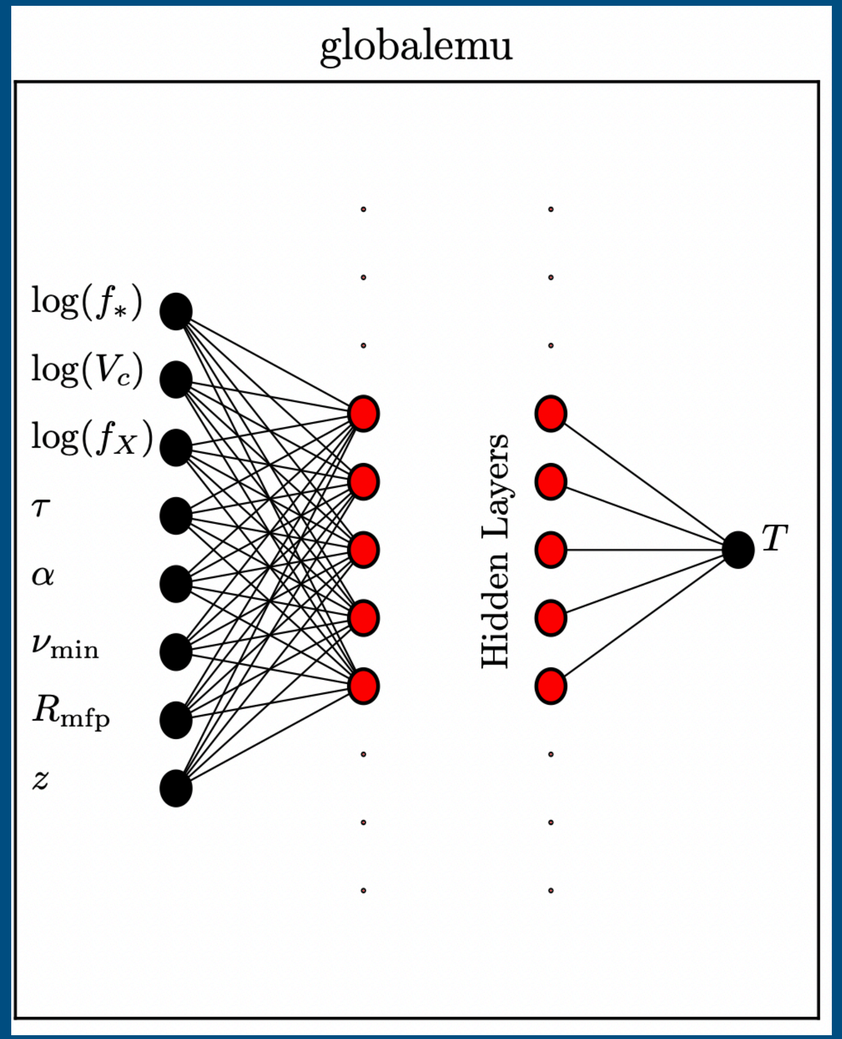
- A similar idea to PCA for performing dimensionality reduction
- Using trained neural networks to work out the best way to summarise the inputs into a lower dimensional space and then transform them to T_{21}
- The idea is that the ‘encoder’ learns the most important features of the astrophysical parameter space and the ‘decoder’ effectively transforms these into the signal



globalemu

Bevins et al. 2022 [2104.04336]

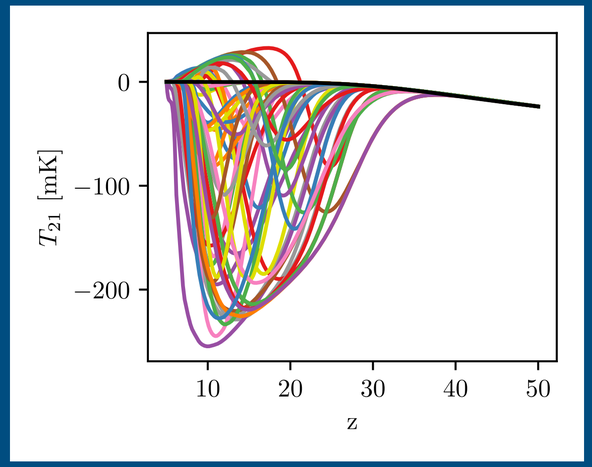
- globalemu uses a signal fully connected neural network
- It takes redshift as an input and outputs a single corresponding temperature
- Multiple calls have to made to produce a signal but its compact, fast and accurate
- The emulator leverages some novel preprocessing of the data

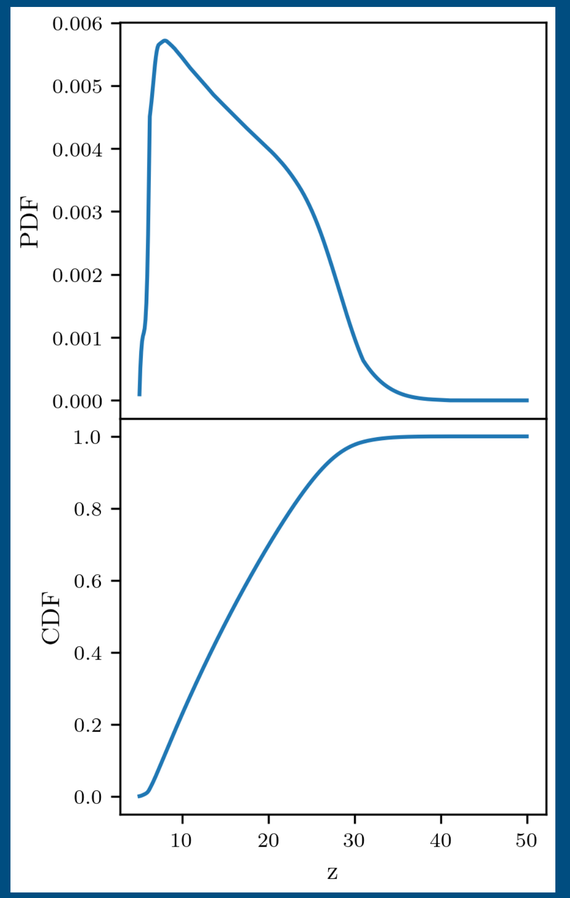


globalemu - preprocessing

Bevins et al. 2022 [2104.04336]

- The 21cm signal has a common structure at high redshift that is independent of the astrophysical inputs
- We model this structure and remove it from the data
- Then to better capture the variation in the 21-cm signal we resample the signals as a function of redshift to a higher degree around regions of high variation in the training data

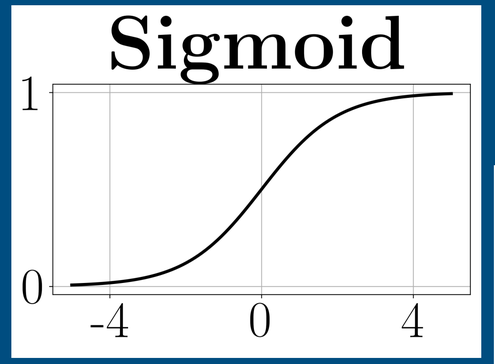


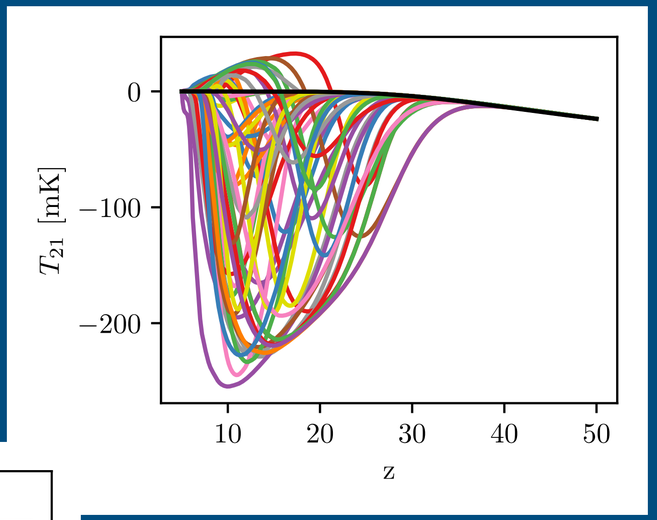


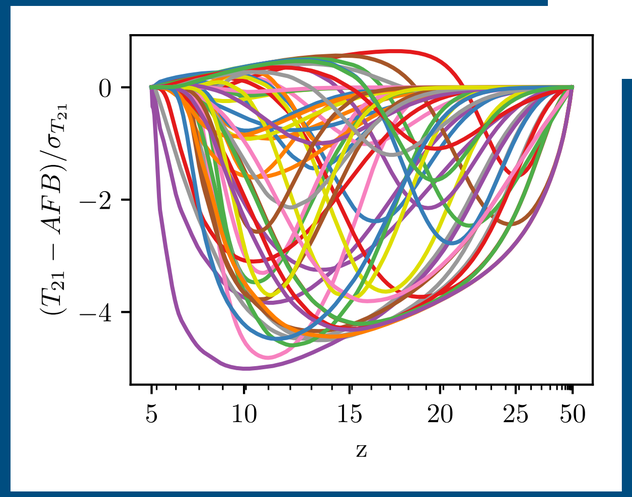
globalemu - preprocessing

Bevins et al. 2022 [2104.04336]

- Neural networks tend to work better if the inputs and outputs are of order unity
  - Activation functions behave better
  - The changes in the weights and biases at each epoch are smaller
- In globalemu the inputs are scaled between 0 and 1 and the output 21-cm signals are divided by the standard deviation of the training data



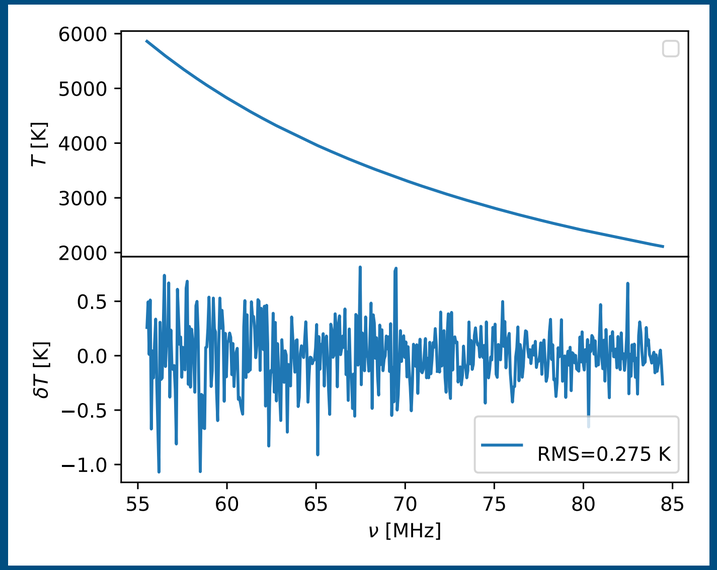


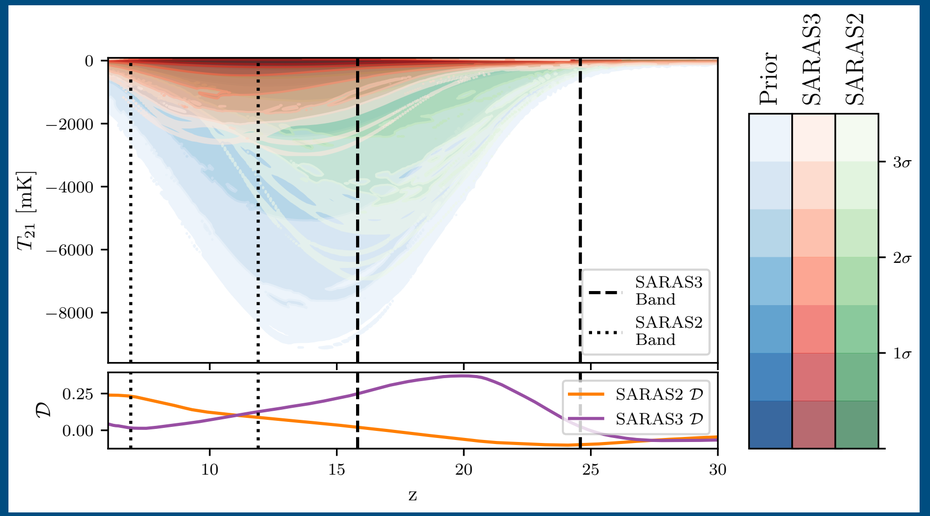


Applications to real data

Bevins et al. 2022a and 2022b [2201.11531, 2212.00464]

- In practice we can use 21-cm signal emulators to physically model the 21-cm signal in real data
- There are many examples of this in the literature
- The idea is we guess some set of astrophysical parameters compare the resultant prediction from our trained emulator to our data and adjust our guess appropriately
- We typically use Bayesian inference to do this (see Dominic's talk tomorrow)





Additional Resources

- Tensorflow: Python machine learning library (with a steep learning curve)
  - Tutorials at https://www.tensorflow.org/tutorials
- Pytorch: Another python library (with a not so steep learning curve)
  - Tutorials at https://pytorch.org/tutorials/
- Lots of good books! e.g. Data Science from Scratch by Joel Grus
  - Has some nice introductory discussion of different ML algorithms including NNs and decision trees

Summary

- Fast access to physical models of the 21-cm signal
- Chain together a series of artificial neurons
- Tuneable weights and biases along connections
- Activation functions to introduce non-linear structure
- Train with backpropagation algorithm and stochastic gradient descent
- Whole array of different applications of NNs in astrophysics