

# Lecture 15

# Signal Emulation for

# Astrophysics and Cosmology

Lecturer: Dr Harry Bevins (htjb2)



# Overview

- Why we need signal emulators?
- Example of a 21-cm signal emulator
- Example Dimensionality Reduction
- Brief discussion of CNNs for SKA power spectrum measurements

Slides available at <https://github.com/htjb/Talks> (and Moodle!)

Example codes on github too!



# Why we need signal emulators?



# Recap: Bayes theorem

$$P(\theta|D)P(D) = P(D|\theta)P(\theta)$$

Outputs                            Inputs

Posterior:  $P(\theta)$       Evidence:  $Z$       Likelihood:  $L(\theta)$       Prior:  $\pi(\theta)$

For MCMC or Nested Sampling



# The likelihood function

- The likelihood function gives the probability of the *data given the model and parameters*  $L(\theta) = P(D|\theta, M)$
- Defined as an input to Nested Sampling and other MCMC algorithms
- A lot of research goes into the form of the likelihood which varies for different analysis problems (e.g. Scheutwinkel et al. [2204.04491], Lovick et al 2312.02075)

# An example likelihood

$$\log_e L(\theta) = \sum_i -\frac{1}{2} \log_e 2\pi |\Sigma| - \frac{1}{2} (D_i - M_i(\theta))^T \Sigma^{-1} (D_i - M_i(\theta))$$

- Work in log space to make things more computationally stable
- The functional form of the likelihood defines the noise distribution in your data
- Here we are assuming the noise is Gaussian and allowing for a non-diagonal covariance



# The issue

$$\log_e L(\theta) = \sum_i -\frac{1}{2} \log_e 2\pi |\Sigma| - \frac{1}{2} (D_i - M_i(\theta))^T \Sigma^{-1} (D_i - M_i(\theta))$$

- In a Nested Sampling run the likelihood function is evaluated for different  $\theta$  hundreds of thousands to millions of times
- This means we need to evaluate the model  $M(\theta)$  millions of times
- But the models are often computationally expensive taking minutes to hours to days per realization



# The issue

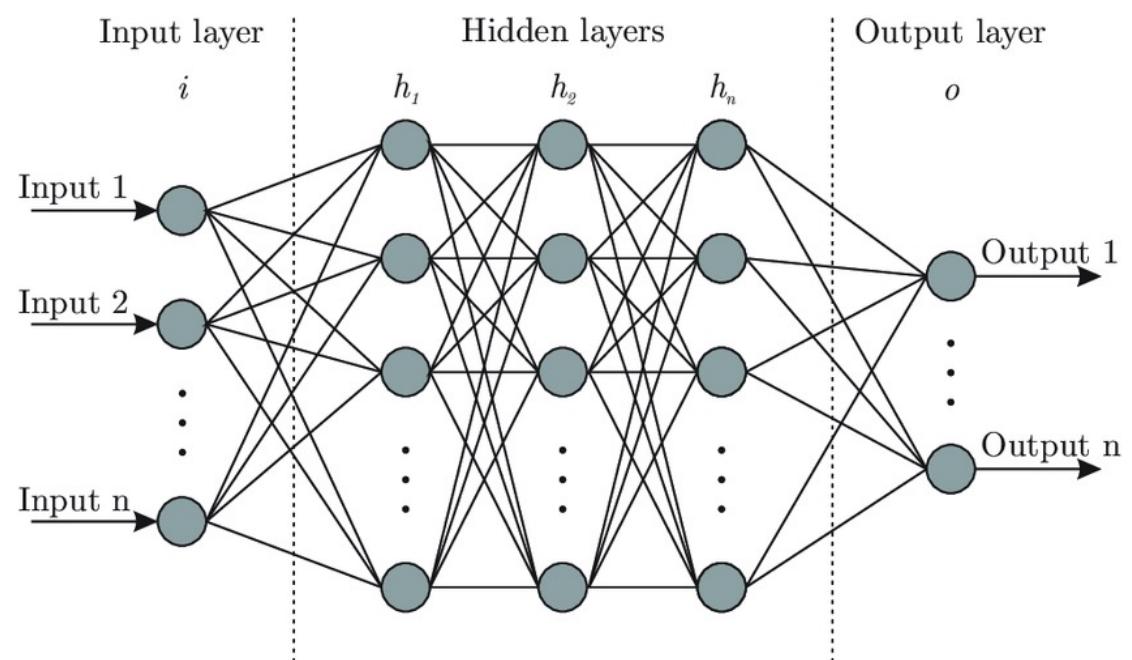
$$\log_e L(\theta) = \sum_i -\frac{1}{2} \log_e 2\pi |\Sigma| - \frac{1}{2} (D_i - M_i(\theta))^T \Sigma^{-1} (D_i - M_i(\theta))$$

- How do we get around this?
- Look towards emulators which can approximate  $M(\theta)$  in less than a few milliseconds
- Go from millions or hours to hours per Nested Sampling run



# What actually is an emulator?

- Some approximation of a complex astrophysical model (think N-body, hydrodynamical or semi-numerical simulations)
- Often built with machine learning tools
- Artificial neural networks and Convolutional neural networks

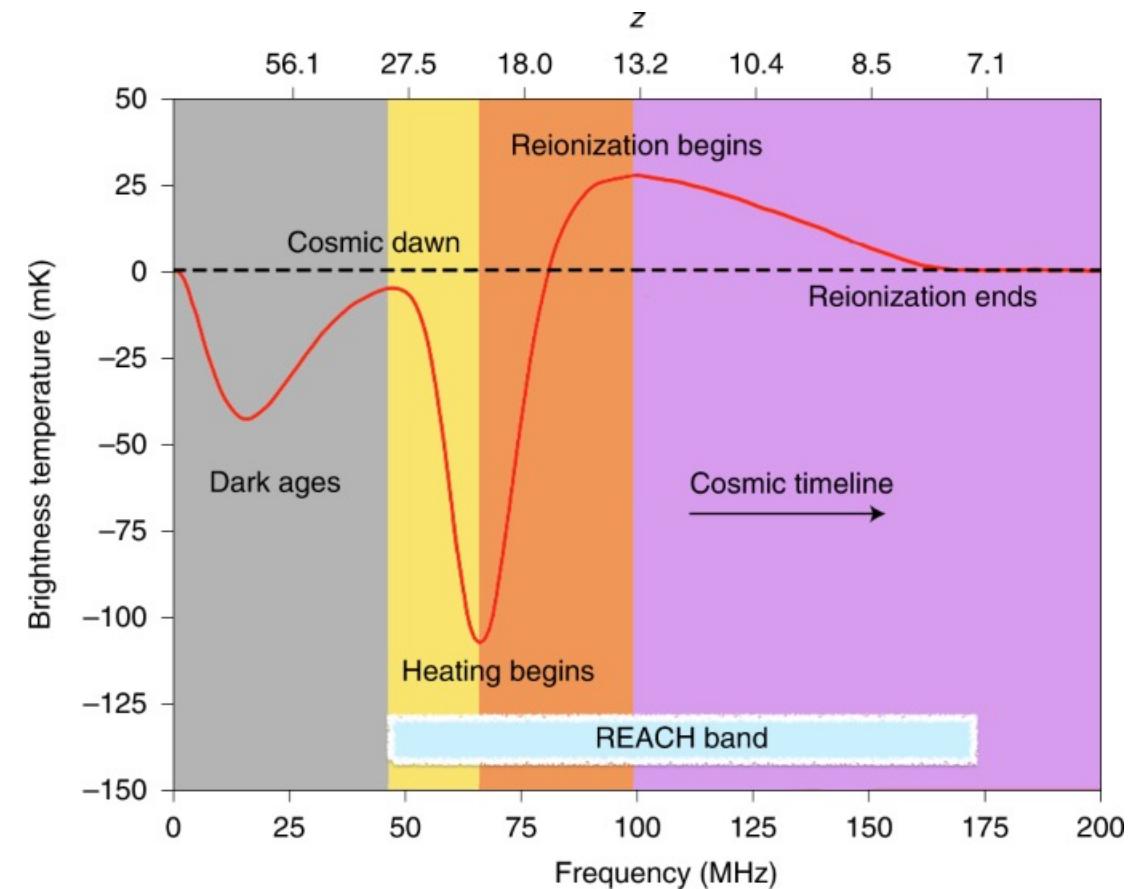


# Sky-averaged 21-cm Cosmology



# Sky averaged 21-cm Cosmology

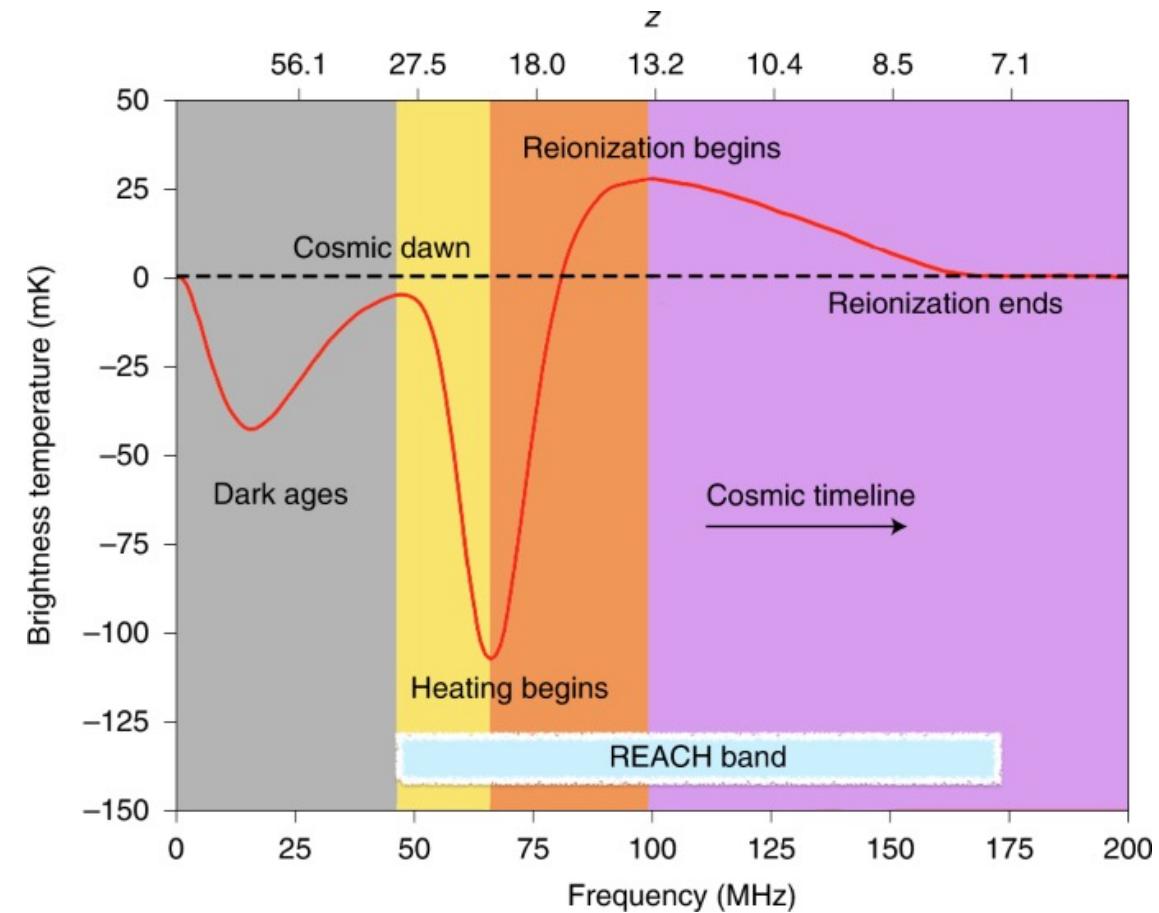
- Looking for a spectral distortion in the CMB temperature caused by neutral hydrogen
- A detection of the signal will help us understand
  - When the first stars formed ad how bright they were
  - The nature of dark matter
  - The abundance and brightness of X-ray emitting objects
  - When the universe transformed from neutral to ionized





# Sky averaged 21-cm Cosmology

- Complex dependence between  $T_{21}$  and the parameters of our models  $\theta$
- We use semi-numerical simulations to model and study how the signal varies over space and time for given  $\theta$
- One signal realization takes  $\approx 3$ hrs

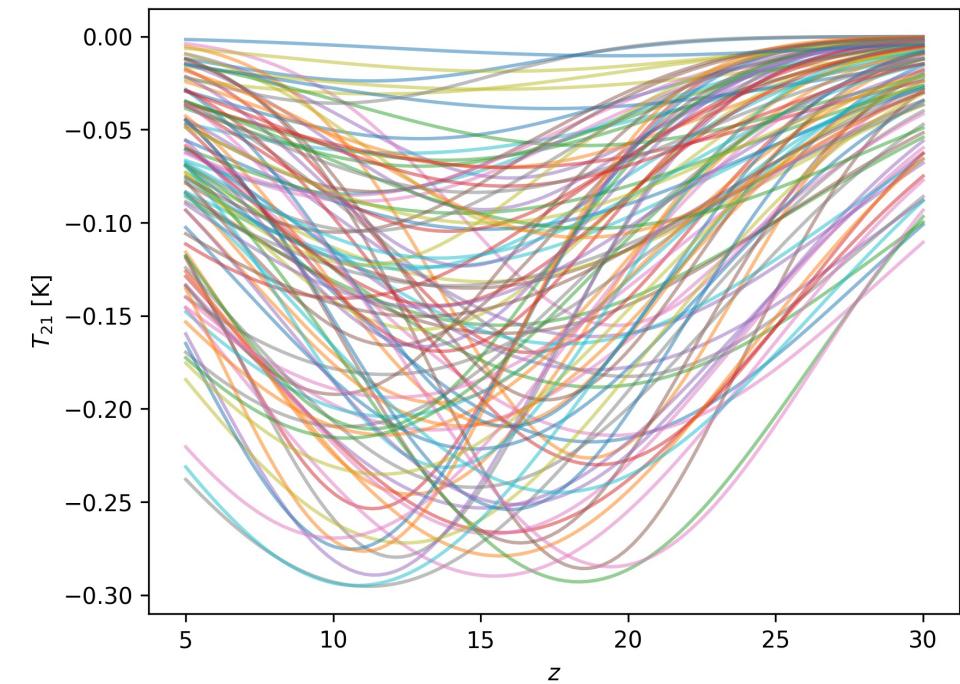




# A toy 21-cm Cosmology example

- We can approximate the 21-cm signal with a Gaussian absorption feature
- Parameterised by
  - An amplitude  $A$
  - A width  $\sigma$
  - A central redshift  $z_c$
- We want to approximate the simulation with a neural network  
 $T_{21}(\theta) \approx f_\phi(\theta)$

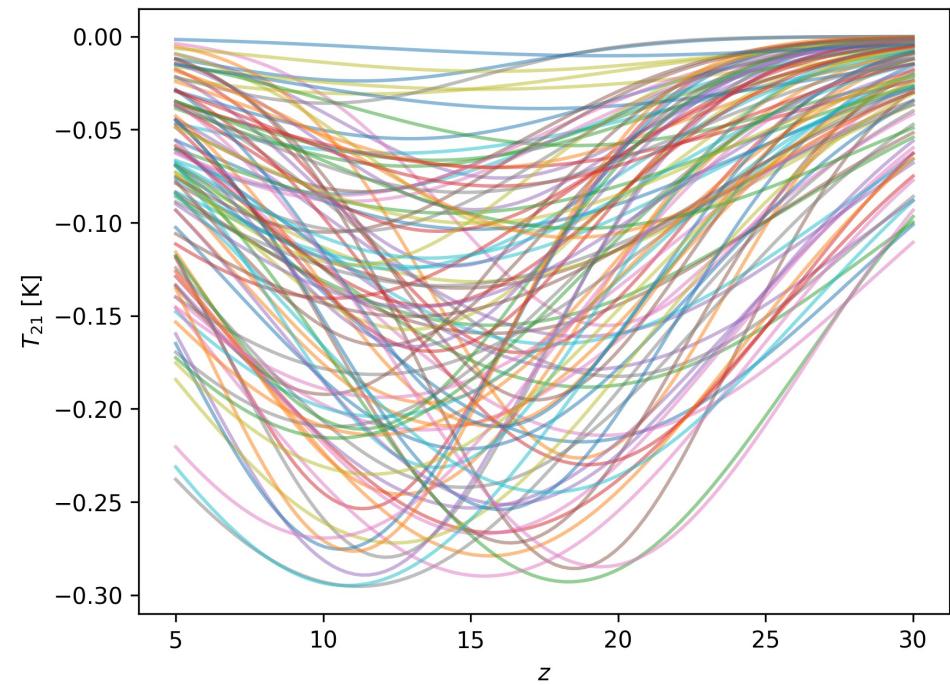
```
def gaussian(parameters):  
    """a simple Gaussian function"""\n    return -parameters[0] * \  
           np.exp(-0.5*(z - parameters[1])**2/\  
                  parameters[2]**2)
```





# Designing the emulator

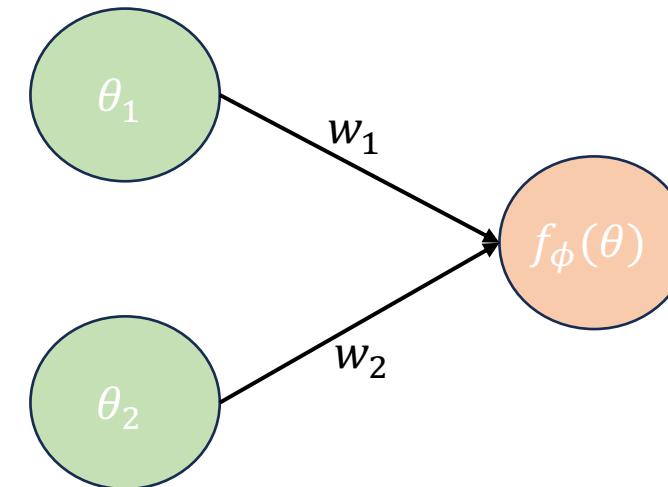
- Need to consider
  - Loss function and early stopping
  - Architecture and dimensionality reduction
  - Normalization
  - Activation functions
  - Measures of accuracy





# Emulators as an approximation

- In  $T_{21}(\theta) \approx f_\phi(\theta)$ ,  $\phi$  are the parameters of our neural network
- $\phi$  has to be optimized so that the approximation is accurate as possible
- Can say  $T_{21}(\theta) = f_\phi(\theta) + \epsilon_\phi(\theta)$  where we are attempting to minimize the error  $\epsilon_\phi(\theta)$



$$f_\phi(\theta) = \sigma(w_1\theta_1 + w_2\theta_2)$$

where

$$\phi = \{w_1, w_2\}$$

and  $\sigma$  is an activation function



# Loss Function

- In order to minimize the error  $\epsilon_\phi(\theta)$  we have to provide the network with training data  $\{T_{21}, \theta\}$
- For each example we evaluate some measure of error e.g.

$$L = \frac{1}{N} \sum_i |\epsilon_\phi(\theta)| \text{ or } L = \frac{1}{N} \sum_i (\epsilon_\phi)^2$$

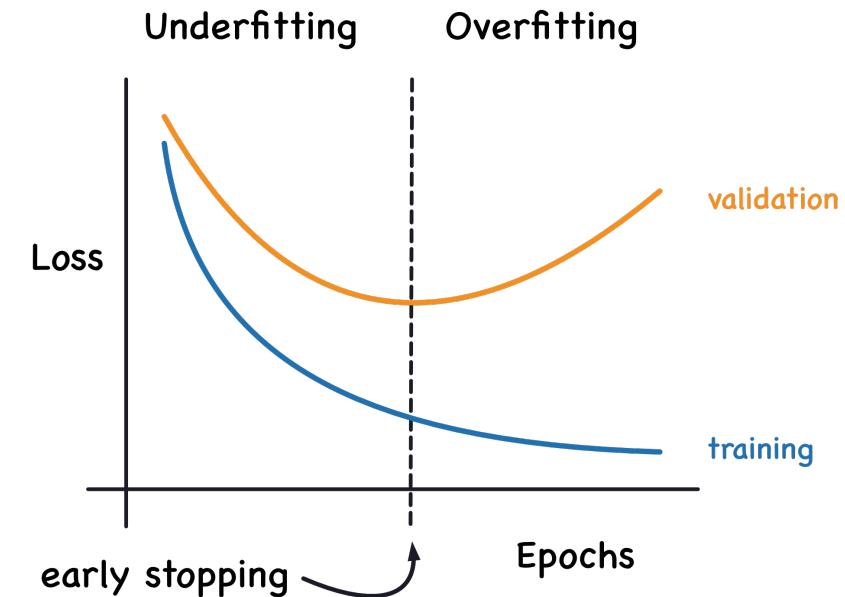
- And adjust  $\phi$  using Stochastic Gradient descent to minimize  $L$



# Training and test data

- We typically generate a few thousand simulations for training
- We reserve a proportion for testing emulator accuracy during training (early stopping)
- Prevents overfitting of the training data

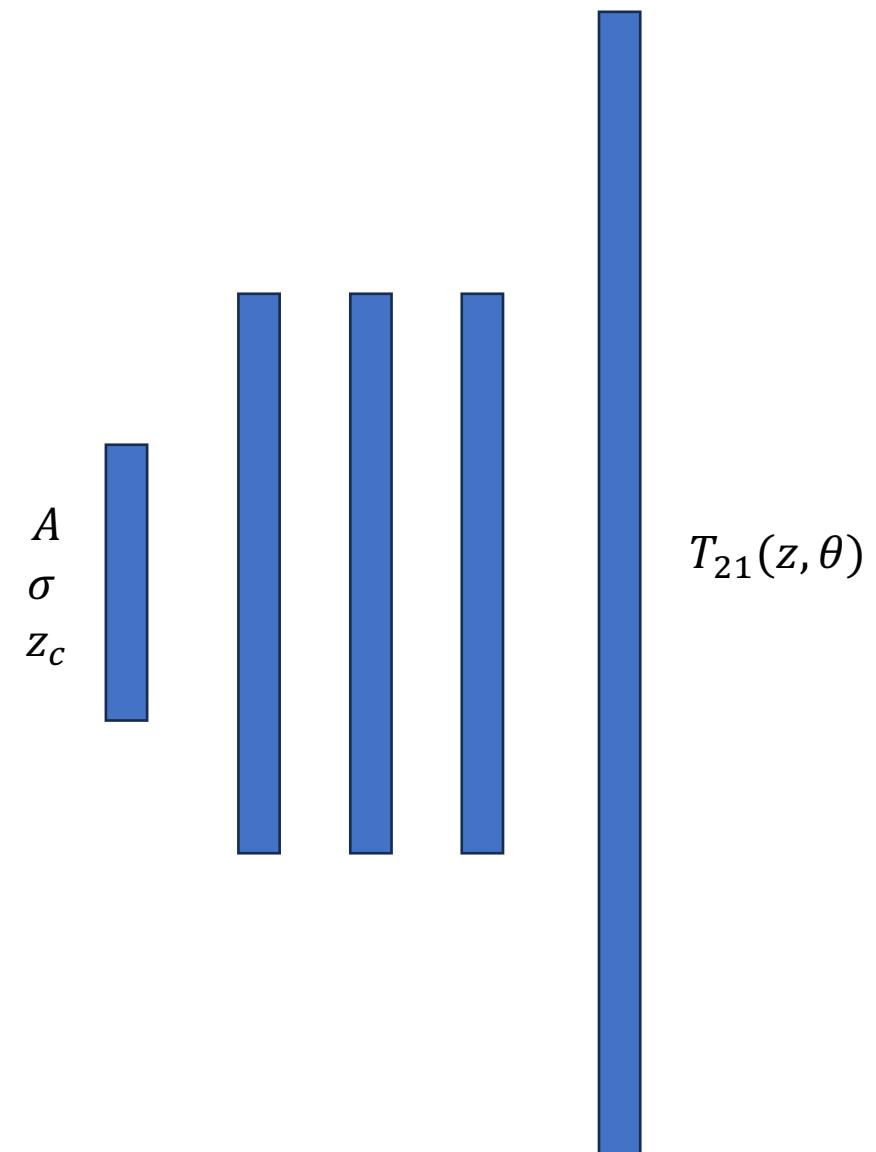
```
# split the data
idx = random.sample(range(n), int(n*0.8))
train_params_pretile = parameters[idx]
train_signals_pretile = signals[idx]
test_params_pretile = np.delete(parameters, idx, axis=0)
test_signals_pretile = np.delete(signals, idx, axis=0)
```





# Architecture choices

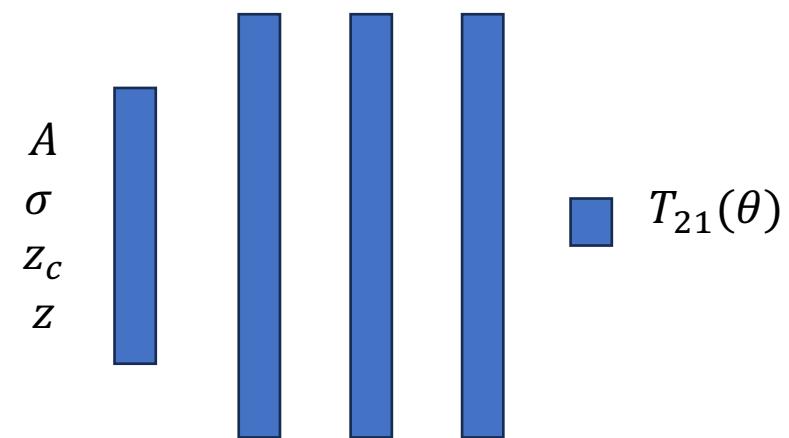
- We could attempt to train a network to go directly from  $\theta$  to  $T_{21}(z)$
- However, we might want to evaluate the 21-cm signal at a range of different redshifts
- Leads to a big network and lots of parameters to optimize





# Architecture choices

- We could perform some form of dimensionality reduction such as PCA (see later)
- But for 21-cm Cosmology we choose to make redshift (the independent variable) an input to our network
- Loop over the network to predict  $T_{21}(z, \theta)$

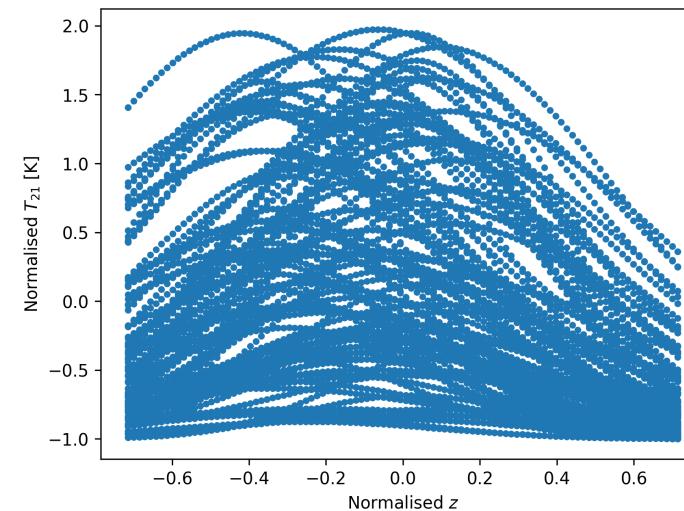
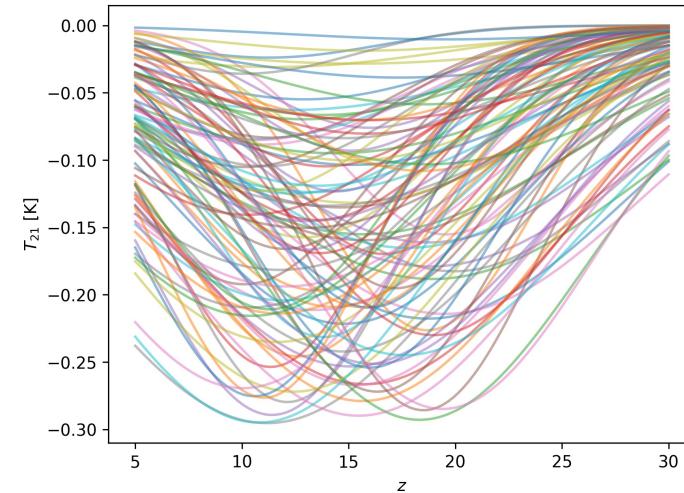




# Normalisation and data processing

- We have to carefully format our data so that we input  $\{A, \sigma, z_c, z\}$  and output  $T_{21}(z, \theta)$
- Perform standardization on training and test data sets

$$\tilde{A} = \frac{A - \bar{A}}{\sigma_A}$$



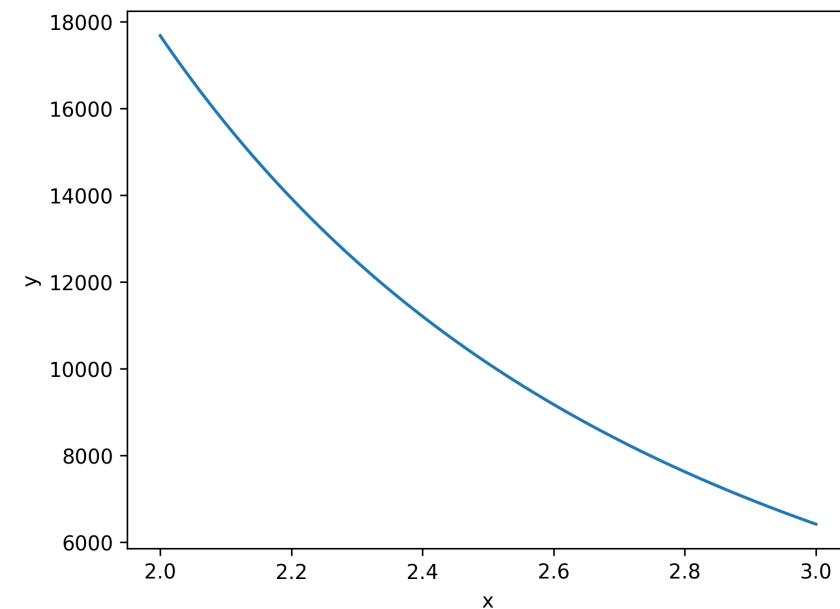


# Why do we normalize?

- Imagine fitting a polynomial function to a power law data set

$$M = \sum_{i=0}^N a_i x^i$$

- If  $x \sim 1, y \sim 10,000$  and  $N = 3$  then the individual  $a_i$  can take on values between  $-10,000$  to  $10,000$





# Why do we normalize?

- But if we normalize so that  $y \sim 1$  then our model is effectively

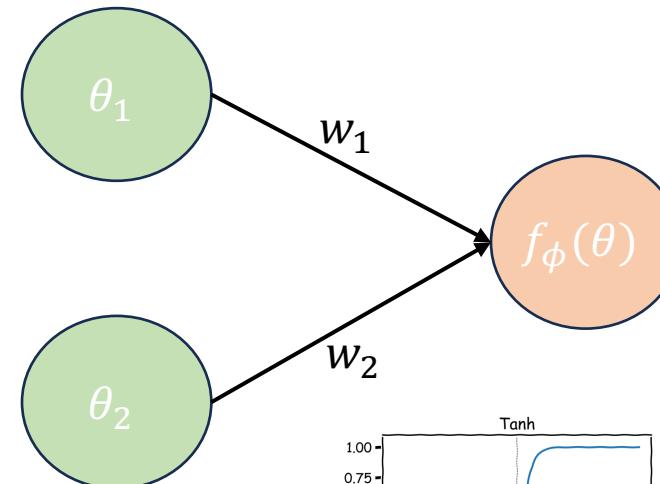
$$M = y_0 \sum_{i=0}^N a_i x^i$$

- And  $a_i \sim 1$  making the problem much easier to fit
- By normalising we reduce the range of values that  $\phi$  can take and make the problem easier to solve
- Activation functions are also usually set up to take in order unity values otherwise they saturate but this is by design i.e. not the reason we normalize



# Activation Functions

- Activation functions add non-linearity into our modelling
- We often want to make careful choices for our activation functions
- E.g. if we know our output (after normalization) should be between 0-1 we might choose a sigmoid activation

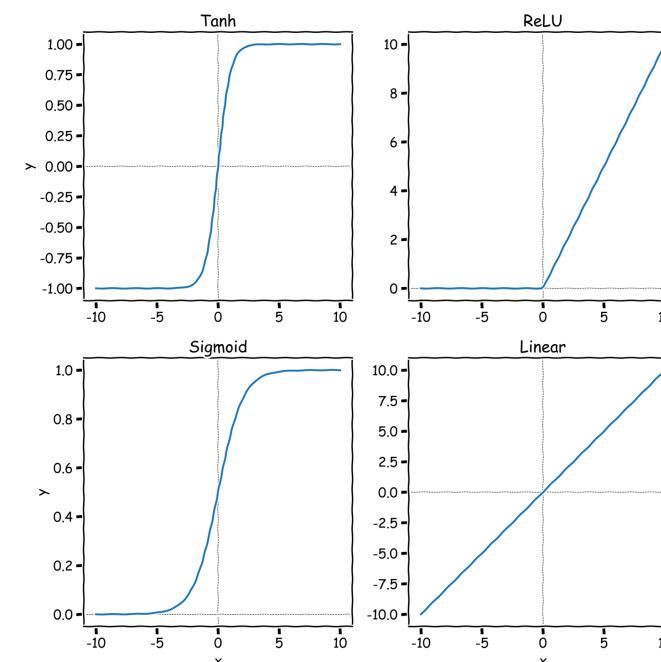


$$f_{\phi}(\theta) = \sigma(w_1\theta_1 + w_2\theta_2)$$

where

$$\phi = \{w_1, w_2\}$$

and  $\sigma$  is an activation function





# Building the neural network

```
# callback for early stopping
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

# neural network architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(4, activation='sigmoid'),
    tf.keras.layers.Dense(8, activation='sigmoid'),
    tf.keras.layers.Dense(8, activation='sigmoid'),
    tf.keras.layers.Dense(8, activation='sigmoid'),
    tf.keras.layers.Dense(8, activation='sigmoid'),
    tf.keras.layers.Dense(1, activation='linear'),
])

# building the model with the adam optimizer and mean squared error loss function
model.compile(optimizer='adam',
               loss='mse')

# training the model
model.fit(train_params, train_signals, epochs=200, batch_size=250,
           callbacks=[callback], validation_data=(test_params, test_signals))
```

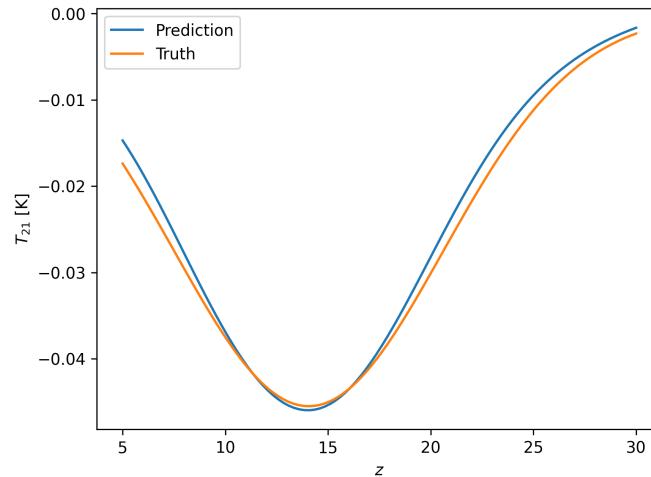
# Assessing the accuracy

- Once trained we want to assess whether the network is doing a good job before we use it in our Bayesian analysis
- The network will predict signals in the normalized space

```
def prediction(params):
    """
    This function takes in a set of unnormalized parameters (A, zc, sigma)
    and returns a predicted signal in the unnormalized space.
    """

    params = np.tile(params, len(z)).reshape(len(z), 3)
    params = np.hstack((params, z.reshape(-1, 1)))
    params = (params - norm_param_means) / norm_param_stds
    pred = model.predict(params, verbose=0)
    return pred*norm_signal_stds + norm_signal_means

pred = prediction(test_params_pretile[100])
plt.plot(z, pred)
plt.plot(z, test_signals_pretile[100])
plt.xlabel('z')
plt.ylabel('signal')
plt.show()
```

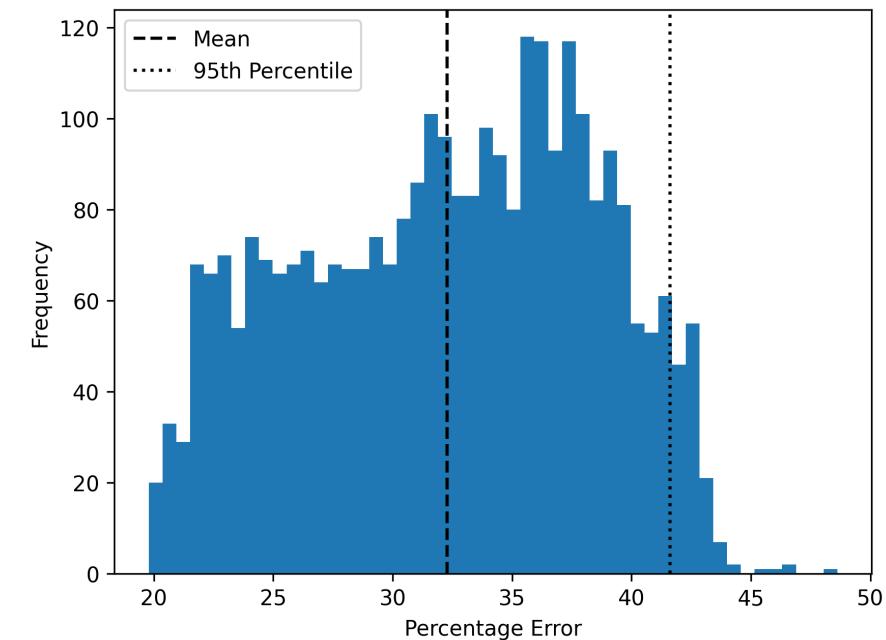




# Assessing the accuracy

- Often we want a more general measure of accuracy
- We assess on the whole test data set and take the average and 95<sup>th</sup> percentile values
- This network isn't very accurate!

```
error = []
for i in tqdm(range(len(test_params_pretile))):
    pred = prediction(test_params_pretile[i])
    error.append(100*np.mean(np.abs(pred - test_signals_pretile[i])) / np.max(np.abs(test_signals_pretile[i])))
error = np.array(error)
```





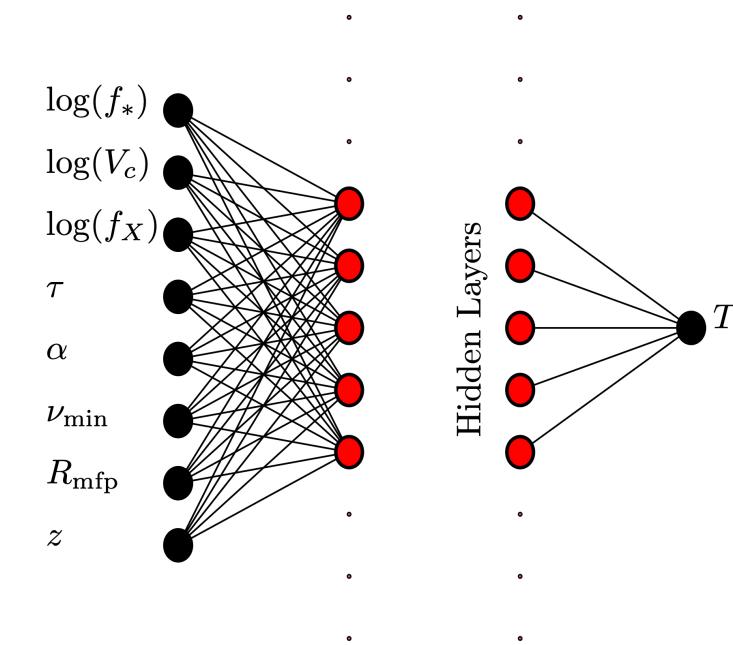
# Current state of the art

- There are many approaches for emulating the sky-averaged 21-cm signal
- `globalemu` [2104.04336] uses some of the ideas discussed here
- It is light weight, easy to retrain and very fast while maintaining a good level of accuracy

## `globalemu`: Robust and Fast Global 21-cm Signal Emulation

### Introduction

<code>globalemu</code> :	Robust Global 21-cm Signal Emulation
Author:	Harry Thomas Jones Bevins
Version:	1.8.2
Homepage:	<a href="https://github.com/htjb/globalemu">https://github.com/htjb/globalemu</a>
Documentation:	<a href="https://globalemu.readthedocs.io/">https://globalemu.readthedocs.io/</a>





# Current state of the art

- 21cmVAE [2107.05581] uses Variational Autoencoders
- More complicated architecture, harder to retrain and an order of magnitude slower than globalemu
- But more accurate than globalemu

## 21cmVAE: A Very Accurate Emulator of the 21-cm Global Signal

CHRISTIAN H. BYE,<sup>1,2</sup> STEPHEN K. N. PORTILLO,<sup>3</sup> AND ANASTASIA FIALKOV<sup>4,5</sup>

<sup>1</sup>Department of Astronomy, University of California, Berkeley, CA 94720, USA

<sup>2</sup>Department of Physics, McGill University, Montréal, QC H3A 2T8, Canada

<sup>3</sup>DIRAC Institute, Department of Astronomy, University of Washington, 3910 15th Ave. NE, Seattle, WA 98195, USA

<sup>4</sup>Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge CB3 0HA, United Kingdom

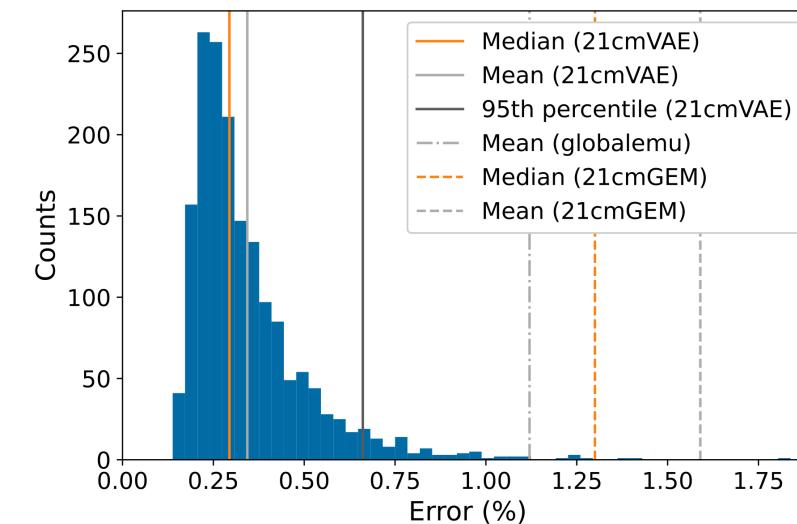
<sup>5</sup>Kavli Institute for Cosmology, Madingley Road, Cambridge CB3 0HA, UK

(Received Month Date, 2021; Revised Month Date, 2022; Accepted Month Date, 2022)

Submitted to ApJ

## ABSTRACT

Considerable observational efforts are being dedicated to measuring the sky-averaged (global) 21-cm signal of neutral hydrogen from Cosmic Dawn and the Epoch of Reionization. Deriving observational constraints on the astrophysics of this era requires modeling tools that can quickly and accurately





# Dimensionality Reduction



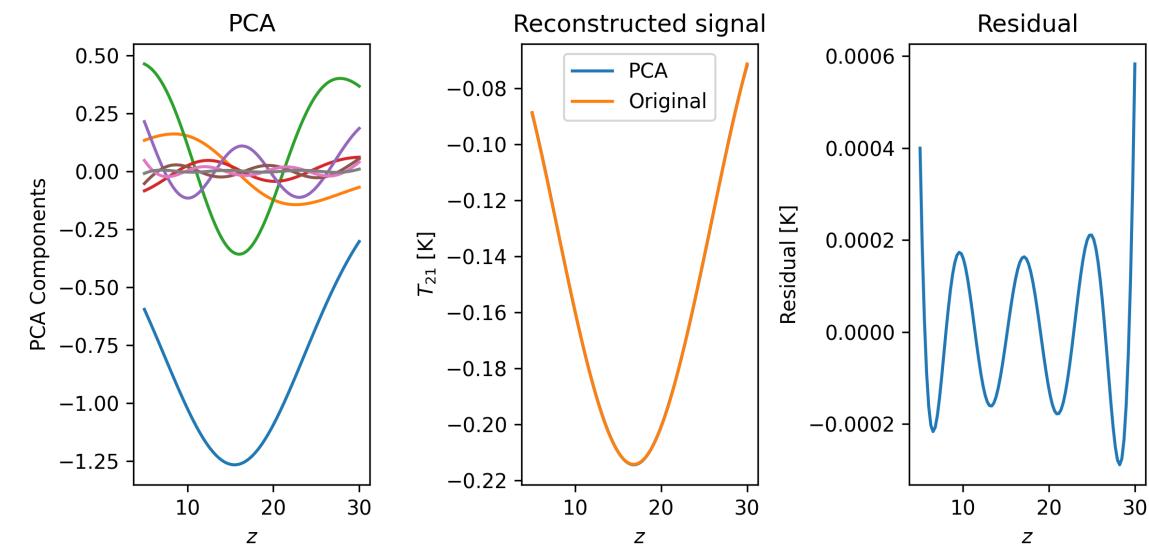
# Dimensionality Reduction

- The idea is to compress the data so that we can predict a few parameters
- And then reconstruct the signal from those parameters
- The issue is that these techniques are often result in information loss

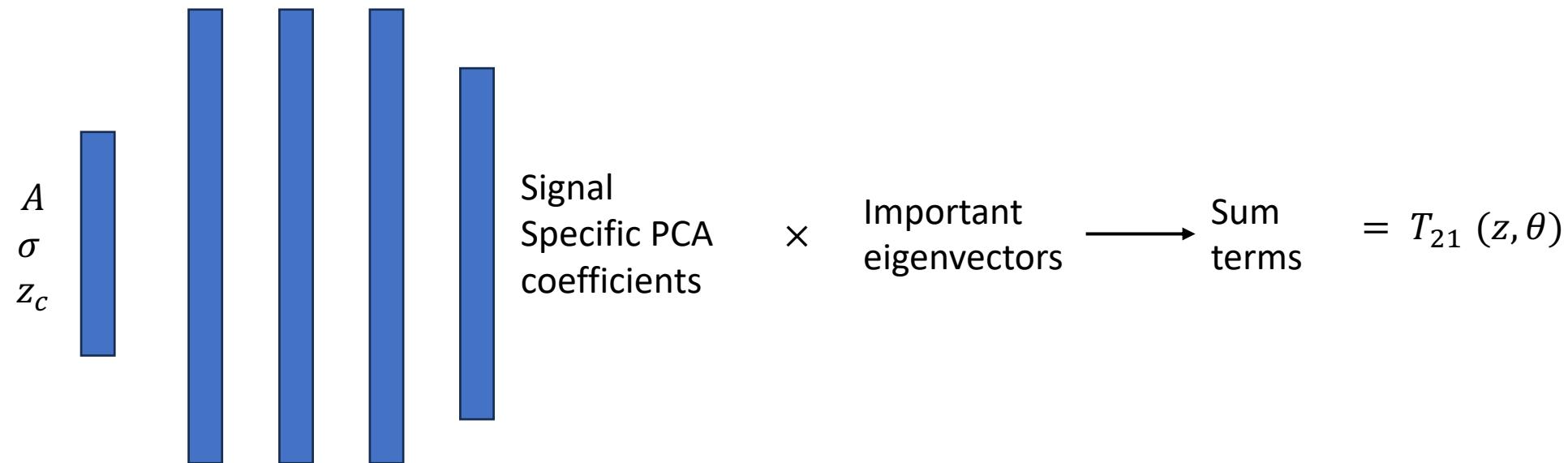


# Principle Component Analysis

- Decompose training data into eigenvectors and order based on eigenvalues
- Eigenvectors with largest eigenvalues describe the directions in the training data with the biggest variance
- Discard eigenvectors that correspond to low variance directions
- Reconstruct signals with weighted sum of most important eigenvectors where weights vary for each signal



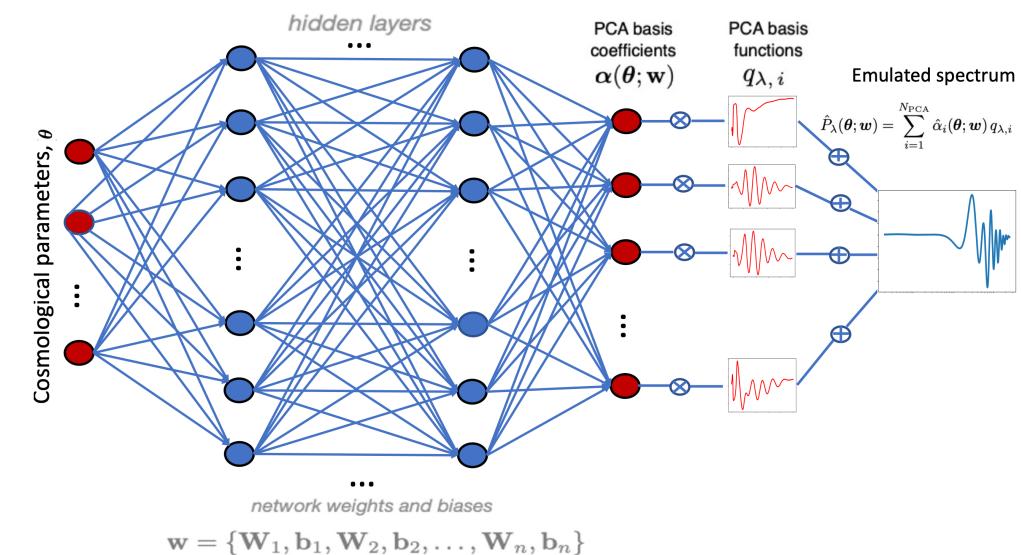
# Principle Component Analysis





# Cosmopower

- Emulating the CMB power spectrum [2106.03846] as a function of cosmological parameters
- Very accurate and easily retrainable
- Uses Principle Component Analysis



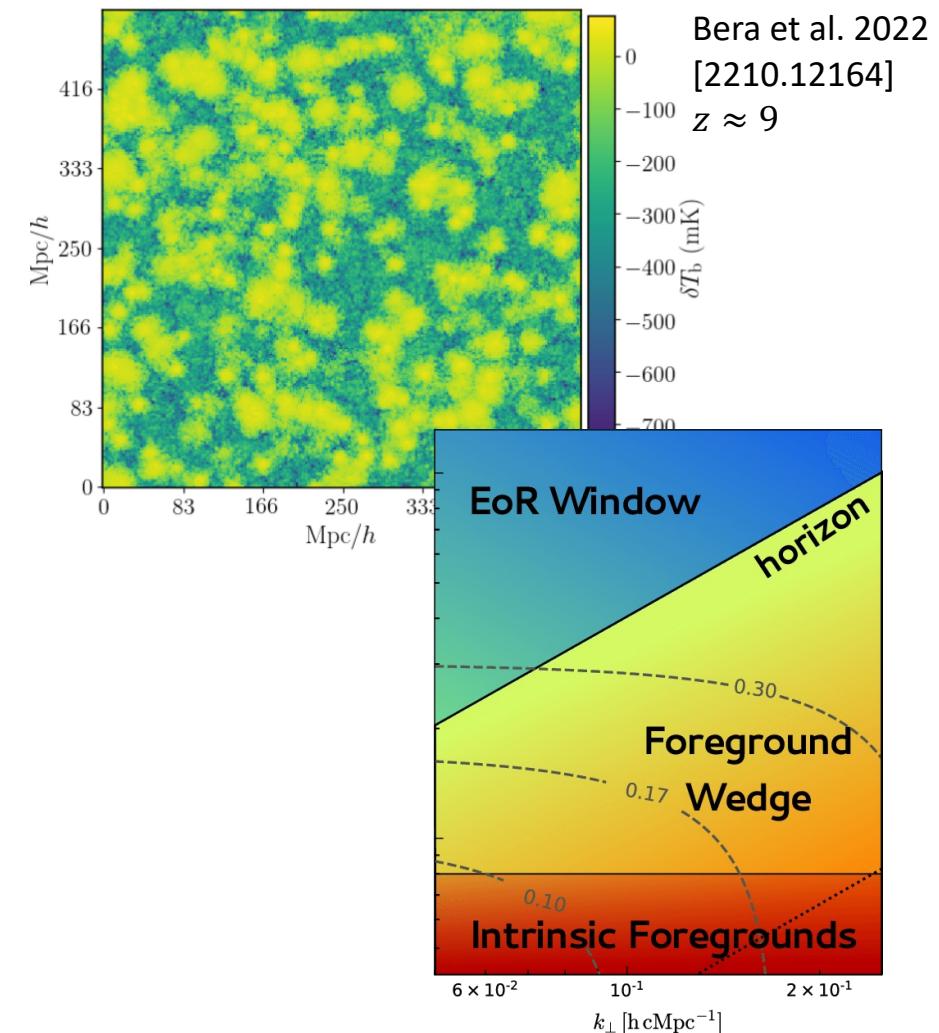


# 21-cm 2D Power Spectrum with the SKA



# The SKA and 21-cm Cosmology

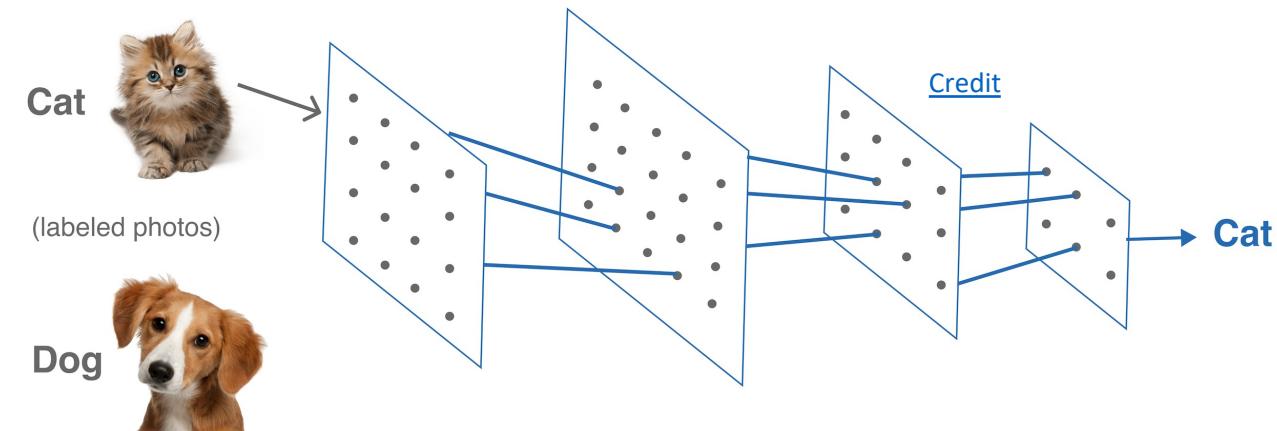
- The SKA will attempt to measure how the 21-cm signal varies spatially and temporally on the sky
- And extract science via the 2D power spectrum





# How do we emulate images?

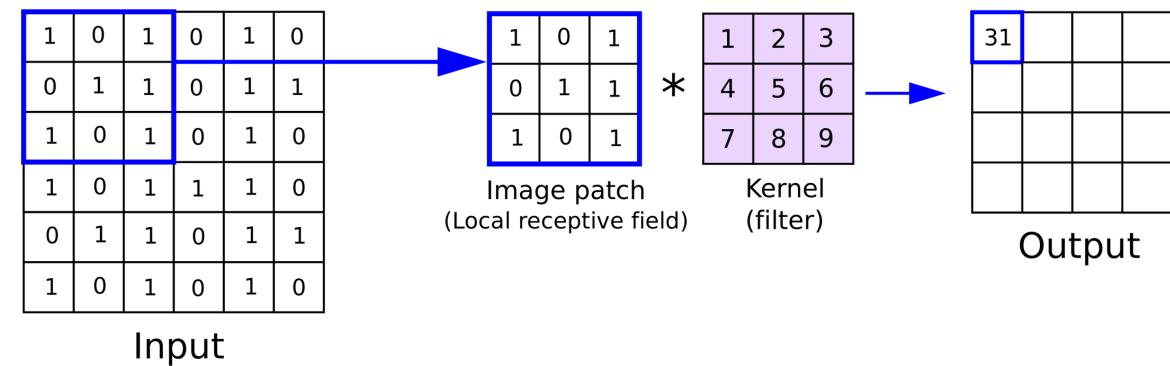
- One way to emulate 2D images is with Convolutional Neural Networks
- Traditionally used for pattern recognition
- And subsequent classification of images



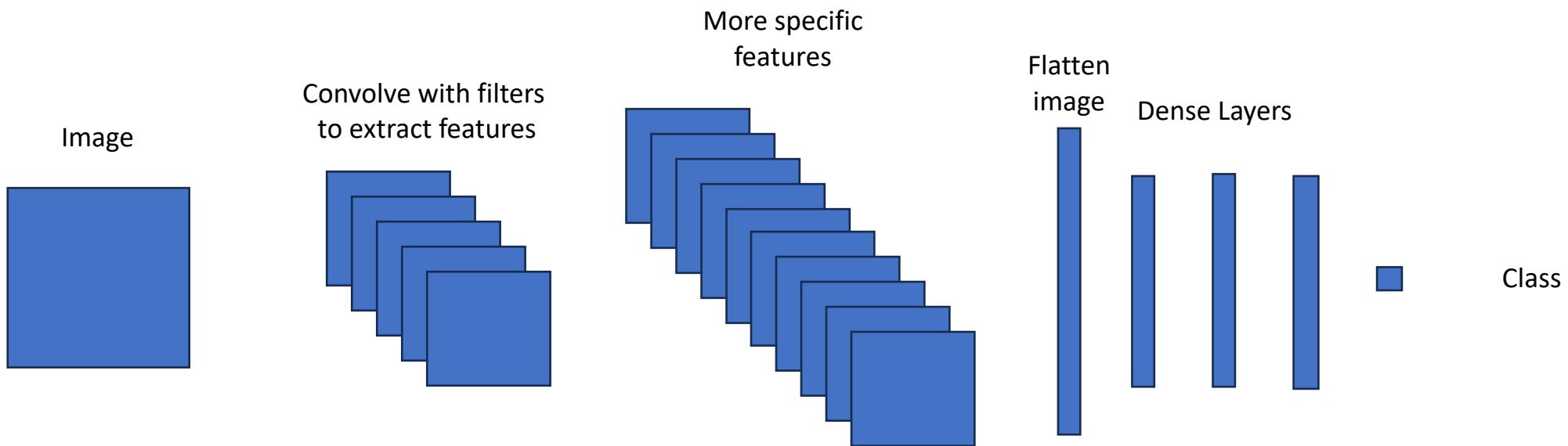


# Convolutional Layers

- Convolutional layers take an image and slide a filter across the image performing a dot product as they go
- Many filters are used to pick out key features gives you a stack of filtered images or a volume
- Filters can be 2D or 3D in nature to increase or decrease the volume of the feature space at each layer

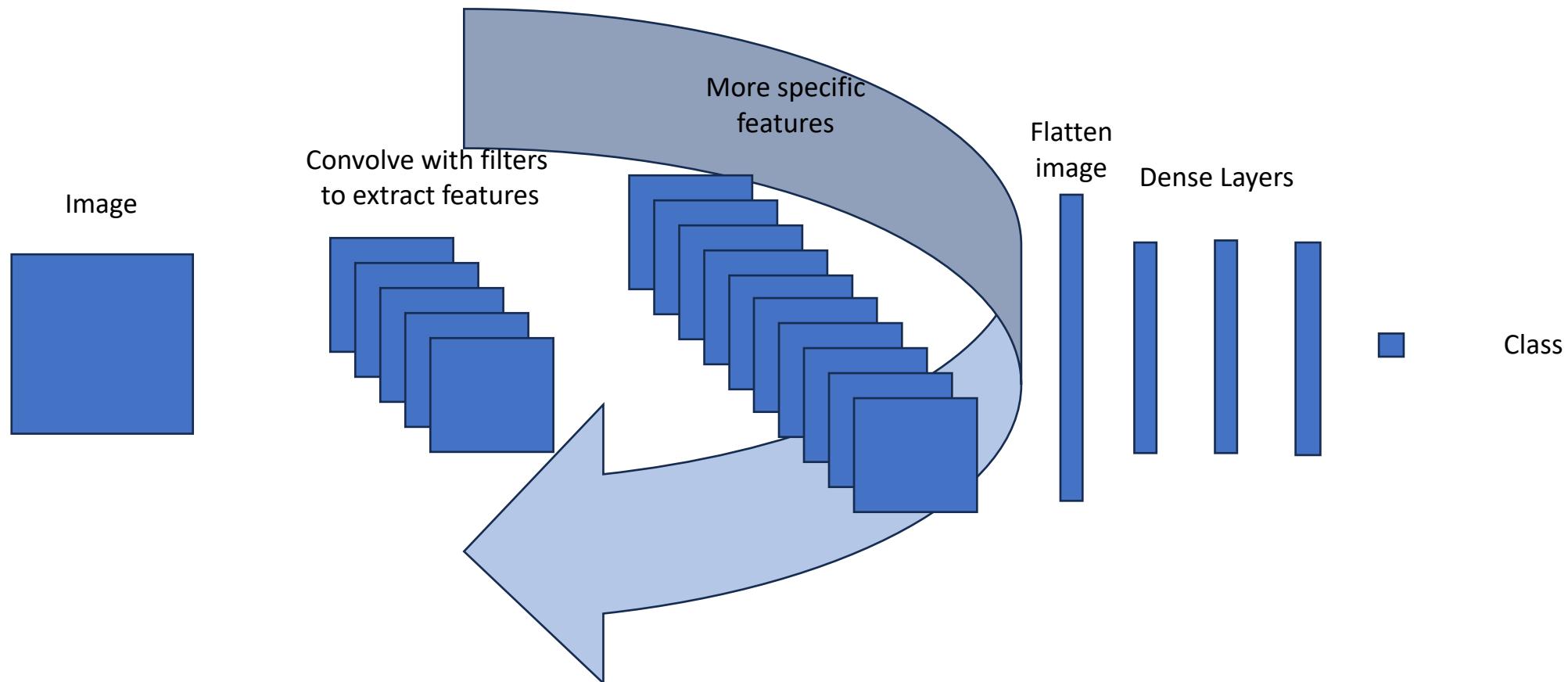


# Convolutional Neural Networks



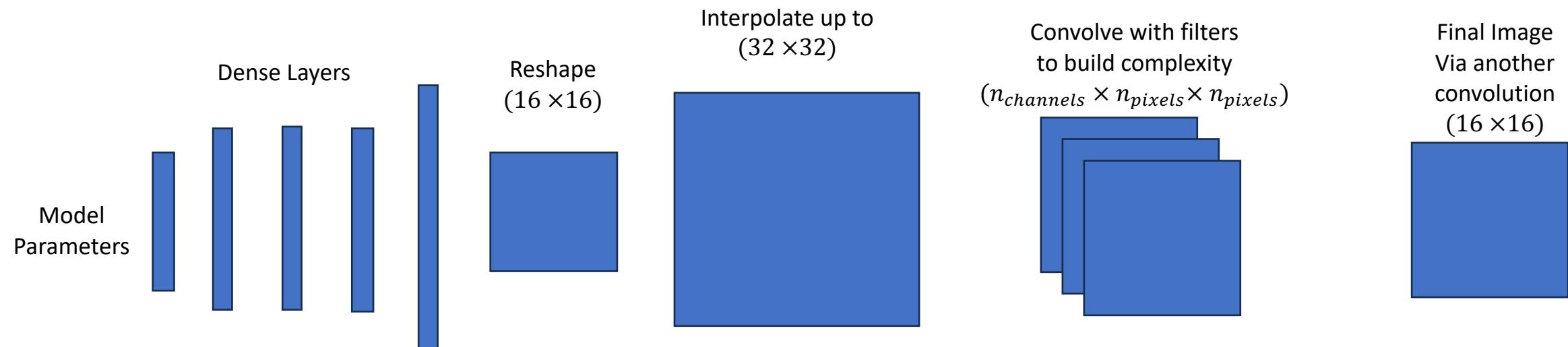


# As Emulators





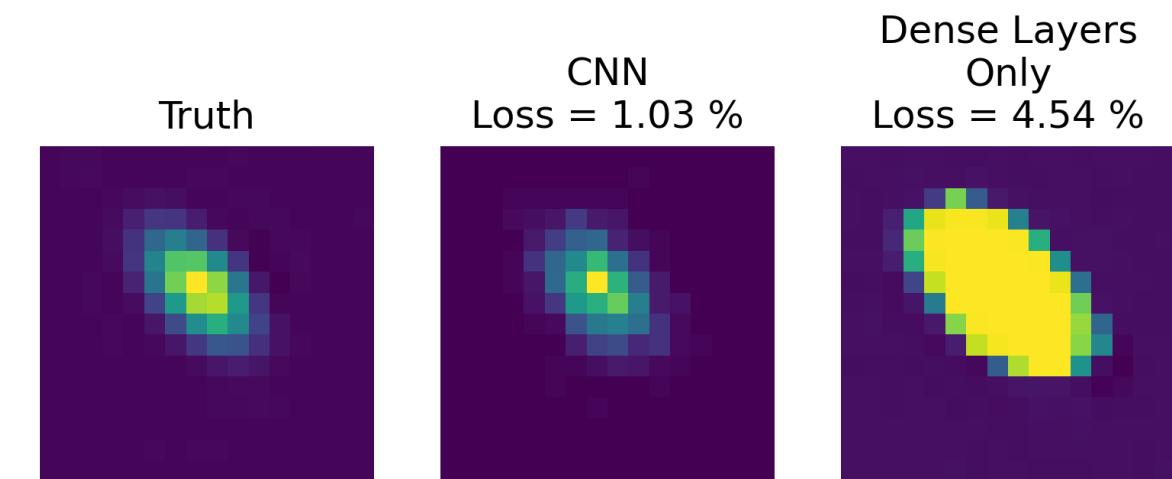
# As Emulators





# An Example

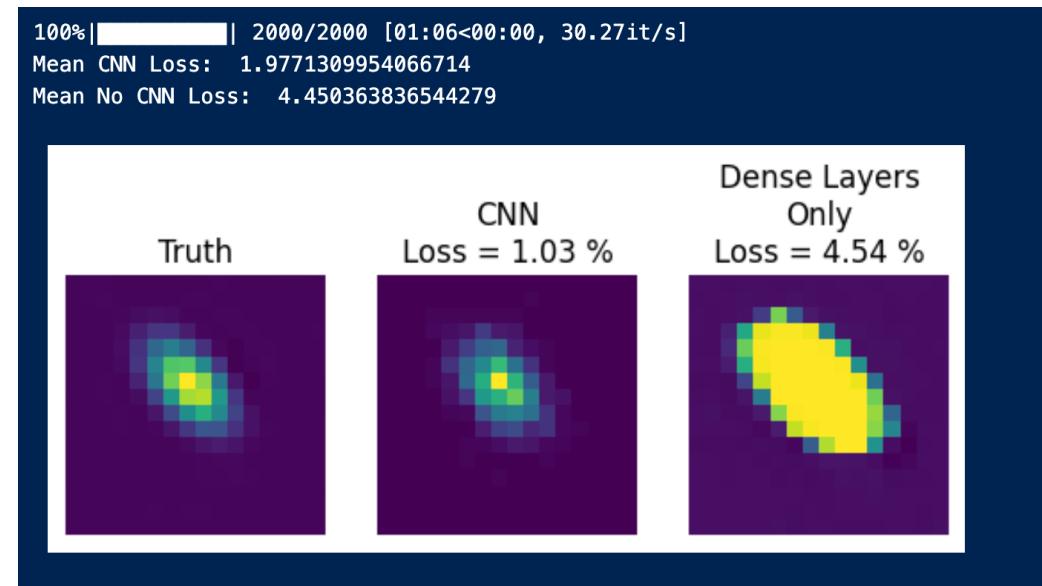
- Multivariate 2D Gaussian with random covariance and means
- Histogram in  $(16 \times 16)$  grid where each bin represents a pixel with some intensity
- Train a network with Convolutional layers and interpolation and a network with just dense layers for comparison





# Optimisation

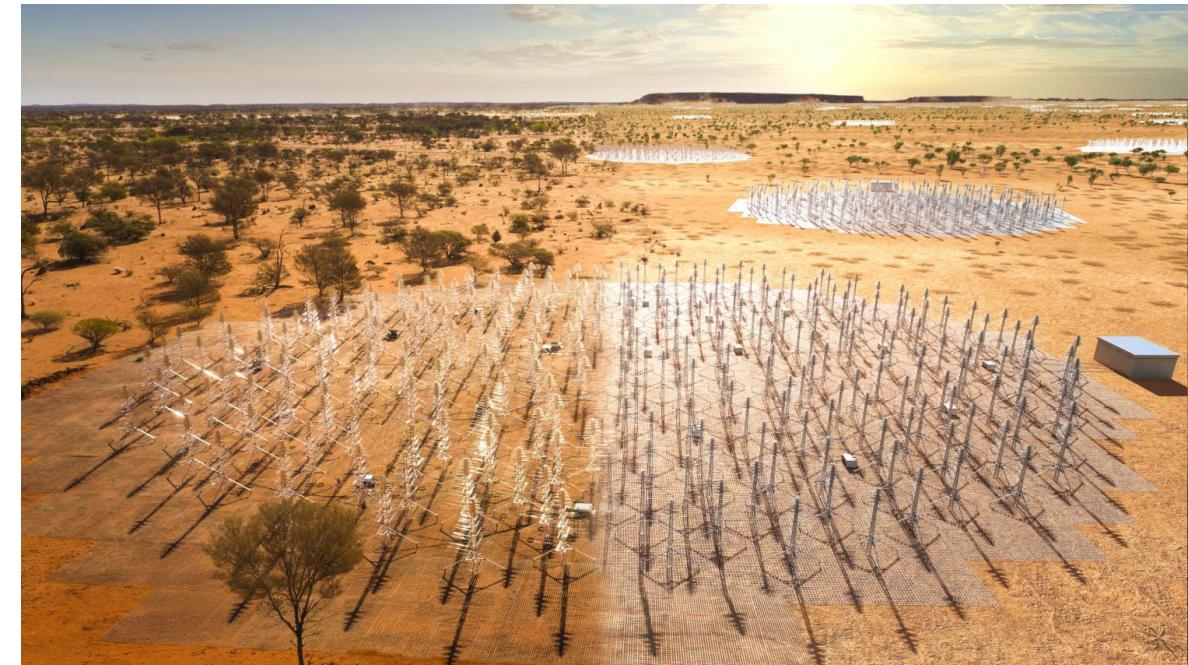
- The values of the filters are optimized in the same way as the weights in a fully connected network to minimize  $\epsilon_\phi$
- Define a pixel by pixel loss between the image and the prediction





# For the SKA 21-cm Observations

- The likelihood is going to be on the power spectrum
- Emulator takes us from astrophysics to the 2D power spectrum
- An example application using CNNs to directly emulate the 2D power spectrum can be found in 21cmEMU [2309.05697]





# Summary



# Emulators allow us to do inference

- Emulators are an efficient way to approximate complex semi-numerical simulations
- They take of order milliseconds to evaluate compared to hours per realization making inference possible
- Sensible choices about architectures, use of dimensionality reduction techniques, activation functions, loss functions etc can make a big difference to the run time and accuracy of emualtors



# Many different ways to do this

- Host of different tools that can be used to build emulators
  - Here we have looked at Dense Neural Networks and Convolutional Neural Networks
  - But we can also do this with Normalizing Flows and Gaussian Processes for example

Slides available at <https://github.com/htjb/Talks> (and Moodle!)  
Example codes on github too!

# Next Lecture: Simulation Based Inference

- Why not just emulate the whole likelihood?