NCI AUSTRALIA

AI for Astronomy

Emulating the model for fast Bayesian inference

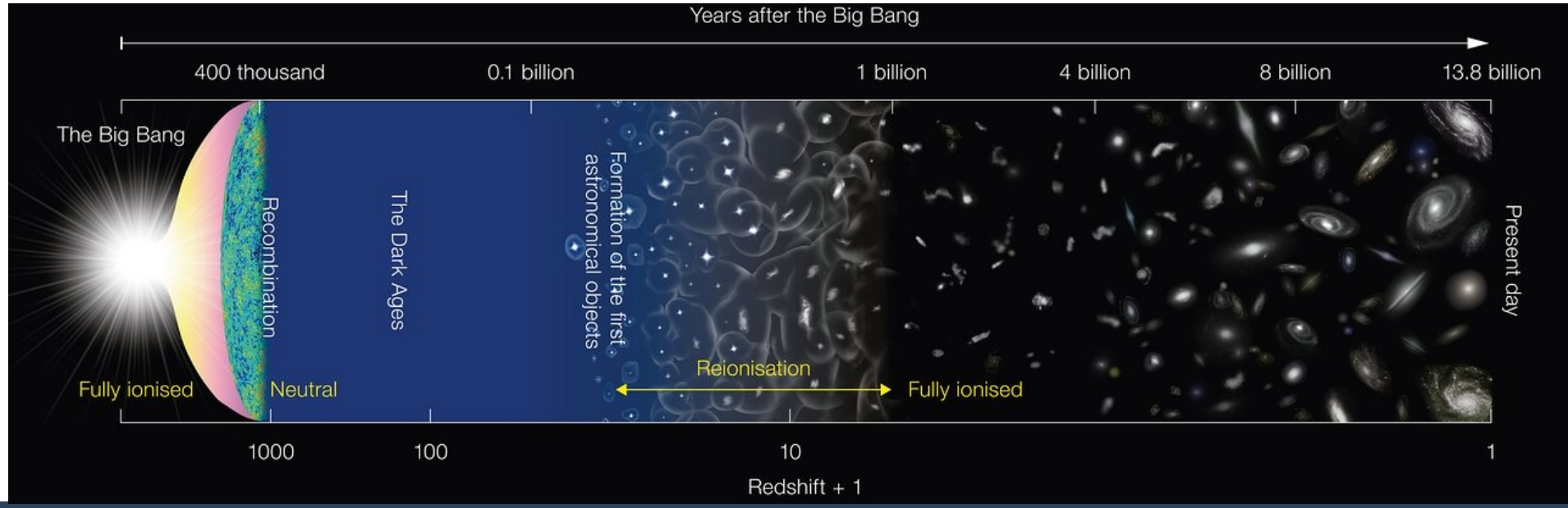
Yuxiang Qin (yuxiang.qin@unimelb.edu.au)

11/June/2023

Observing the Universe

The chronology of the universe describes the history of our Universe.
Astronomers are or will be observing most epochs shown in this diagram, including:

- CMB temperature and polarization anisotropies
- 21cm global signal and spatial variation
- Ionization of neutral hydrogen
- Galaxies
- and astronomical objects within galaxies (e.g. pulsars, FRBs, gravitational waves,…)

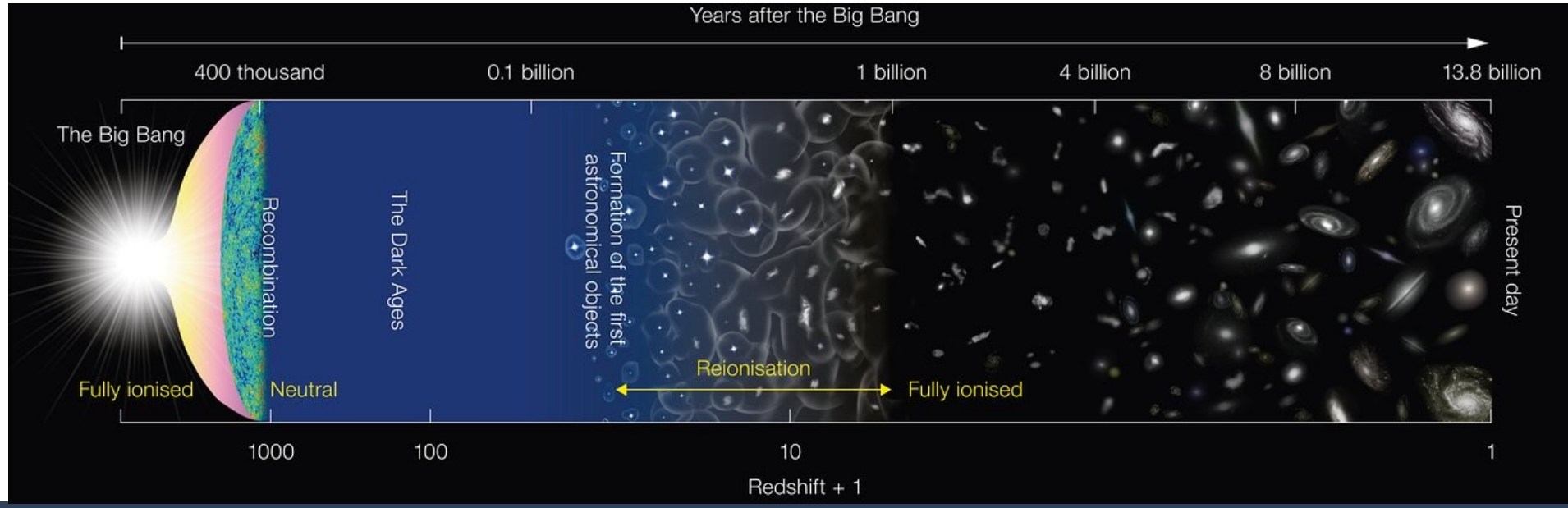


Modelling the Universe or part of it

The chronology of the universe describes the history of our Universe.
Astronomers are or will be observing most epochs shown in this diagram, including:

- CMB temperature and polarization anisotropies
- 21cm global signal and spatial variation
- Ionization of neutral hydrogen
- Galaxies
- and astronomical objects within galaxies (e.g. pulsars, FRBs, gravitational waves,…)

Theorists are also modelling these to understand the property of underlying drivers



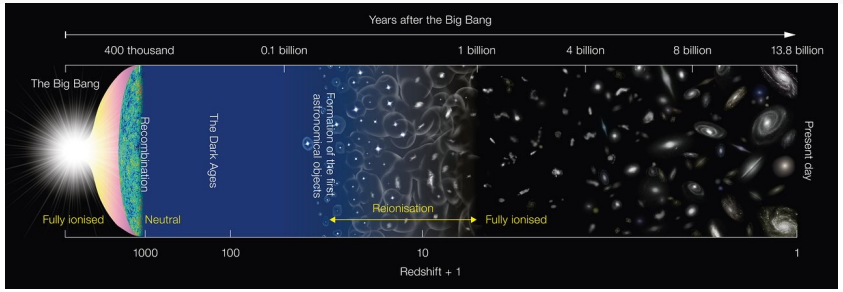
Summary statistics

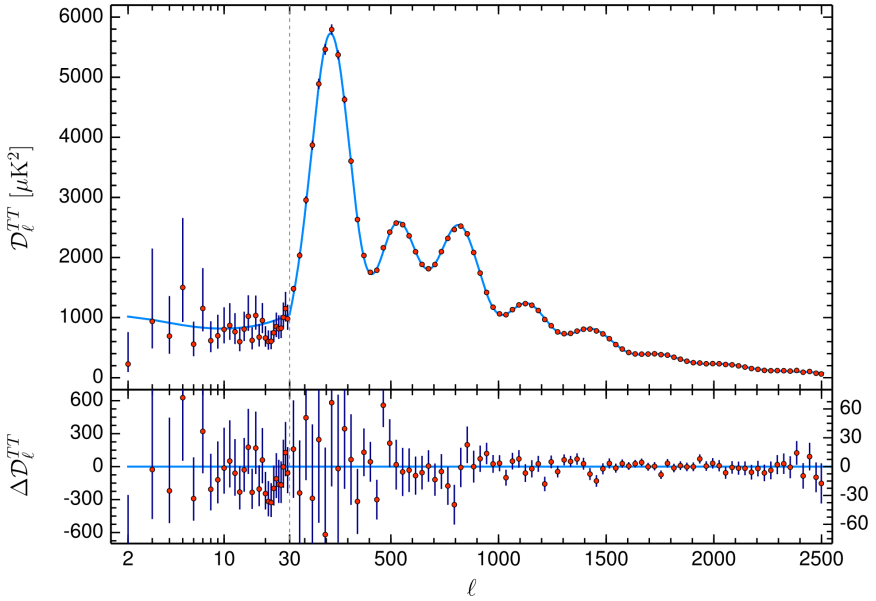
On large scale, matching/fitting the entire maps is often impractical due to the lack of initial condition. e.g. CMB maps, 21cm maps, cosmic webs of galaxies, galaxy morphology.

On small scale, observing the object becomes challenging due to resolution limit and magnitude depth of our instrument. e.g., imaging of a neutron star or blackhole, supernovae or their remnant.

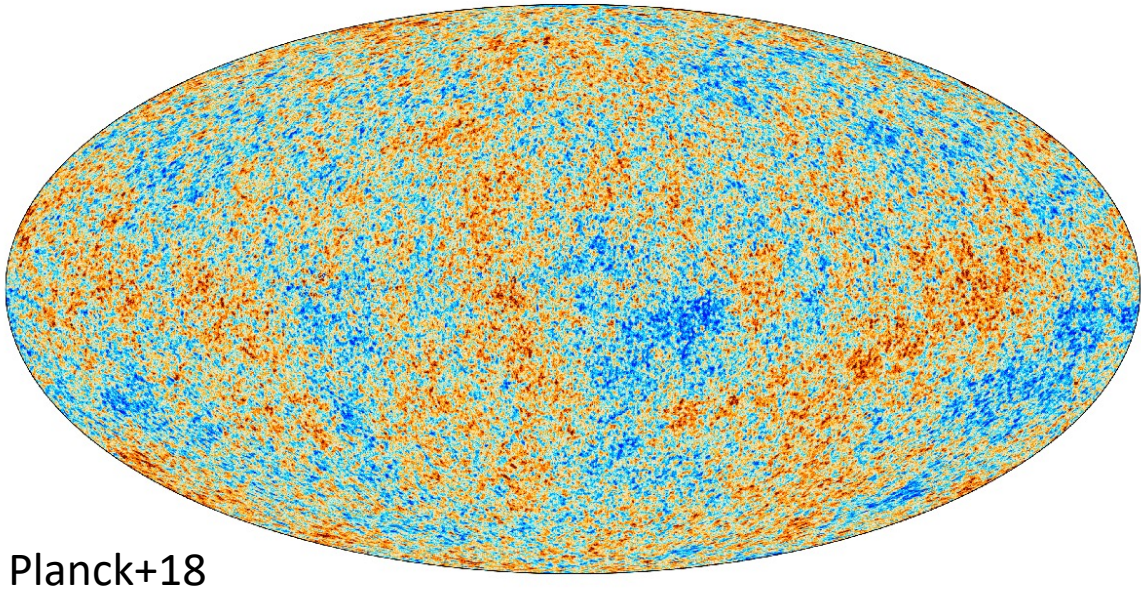
Therefore, in many cases, summary statistics are proposed to bridge observation and models. e.g. power spectra for intensity mappings, number density for a population of astronomical objects, light curve of a pulsar, strain evolution of a gravitational wave event…

CMB (power spectra)

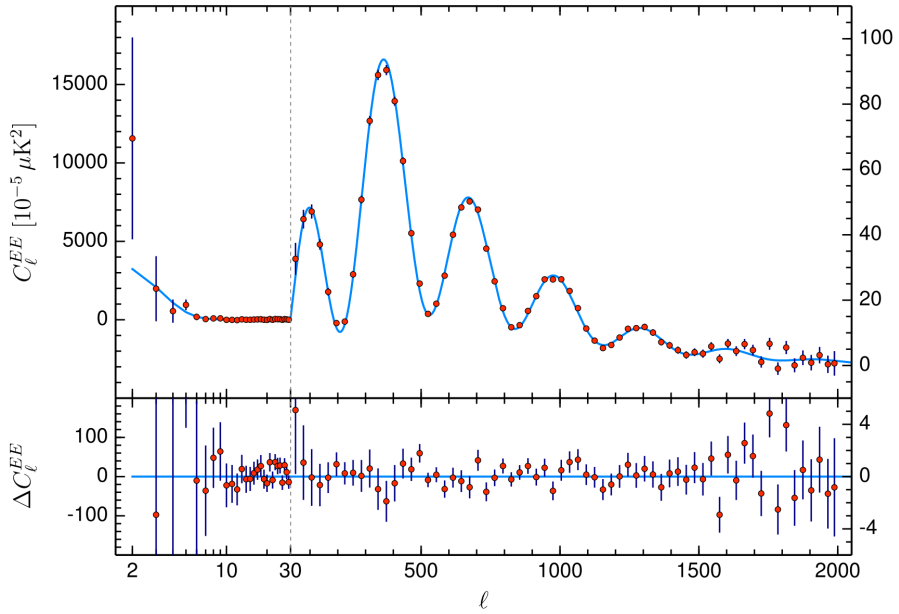




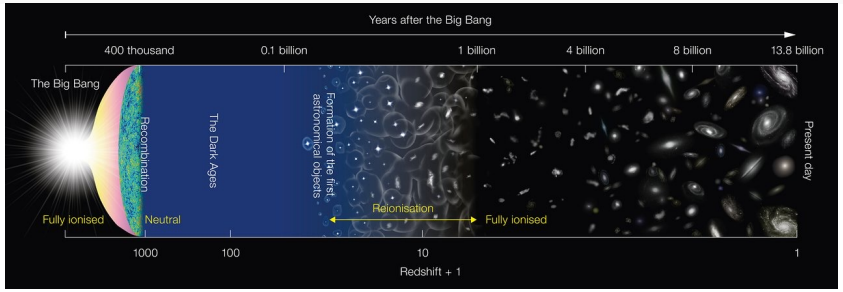
2D map->1D curve

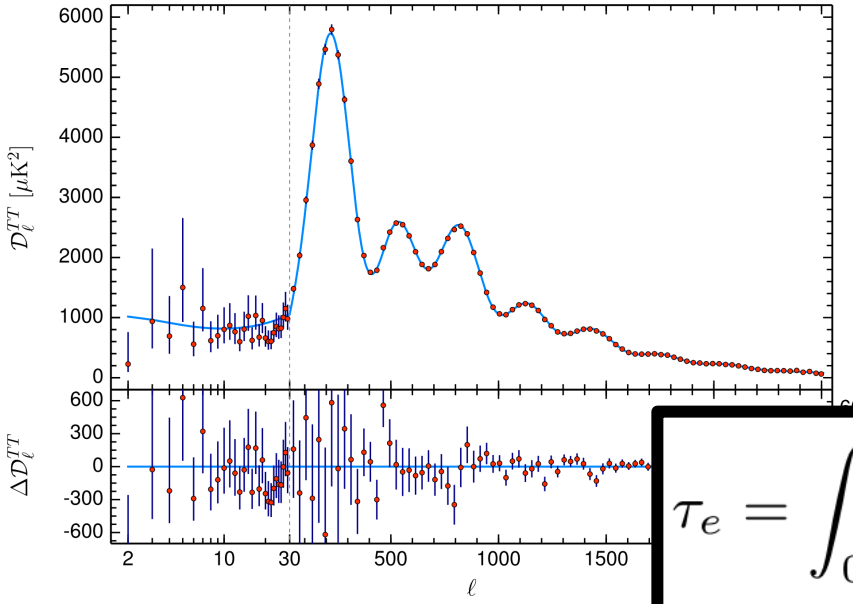


Planck+18



CMB (optical depth)

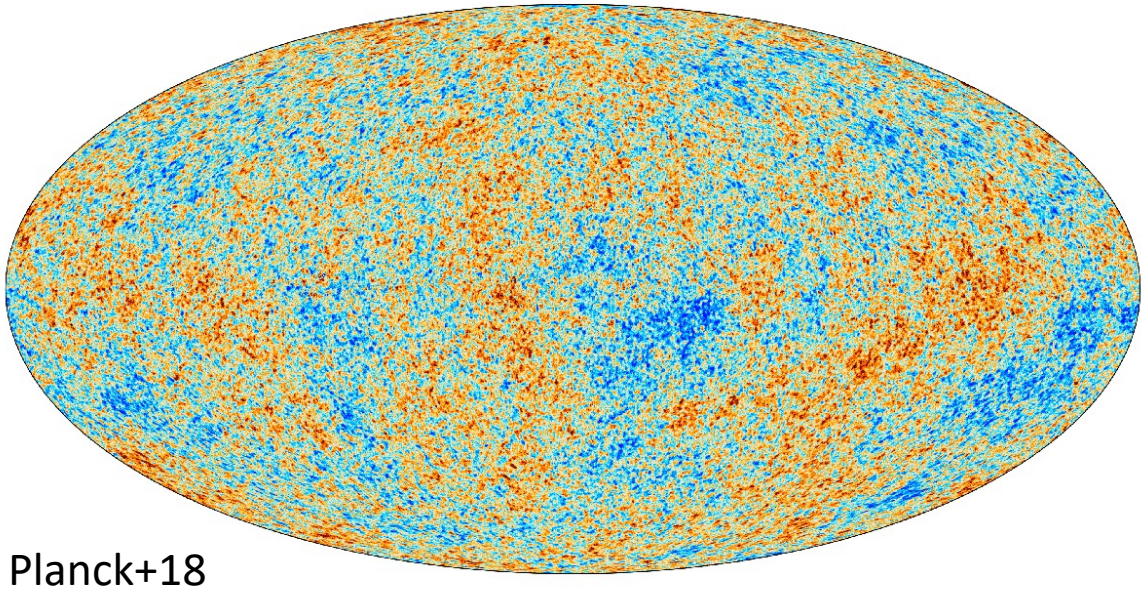




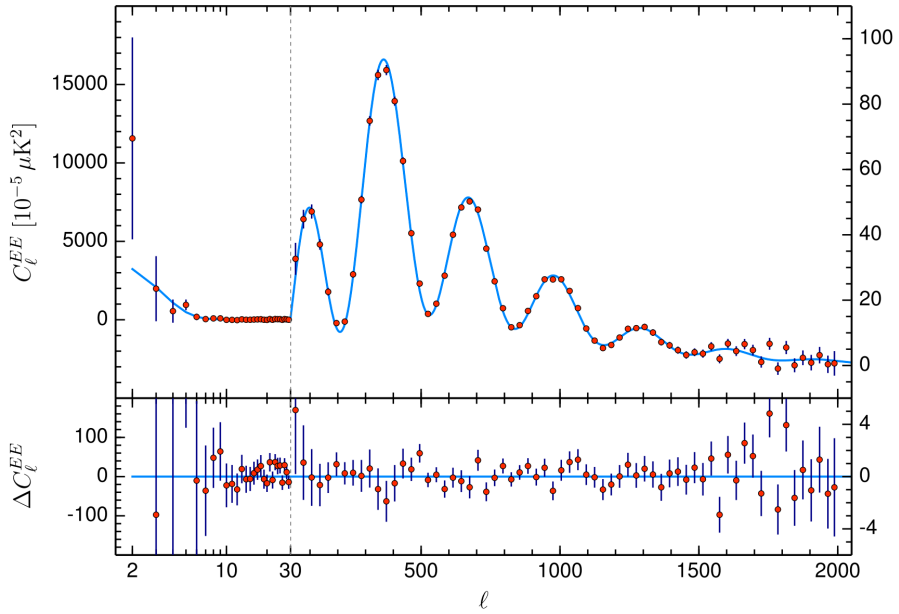
2D map->1D curve

->1 scalar

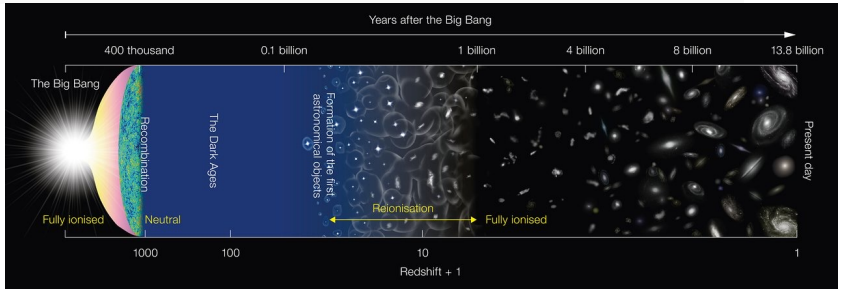
$$\tau_e = \int_0^{z_\mathrm{d}} cH^{-1}\,(1+z)^2\,n_e\sigma_\mathrm{T}$$

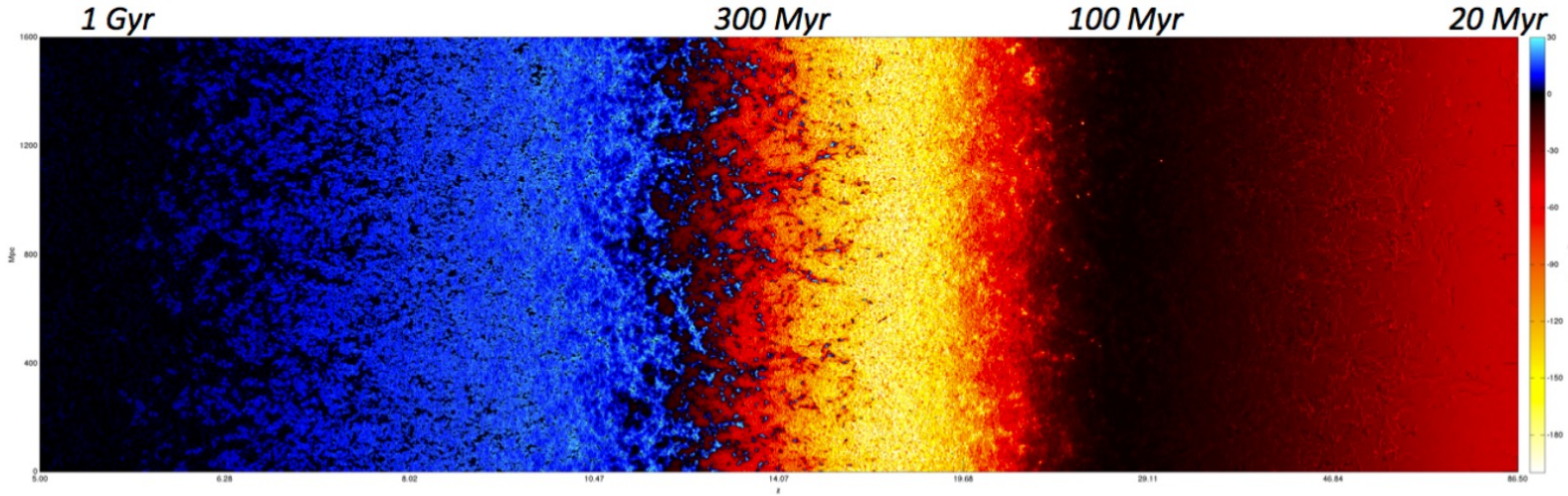


Planck+18

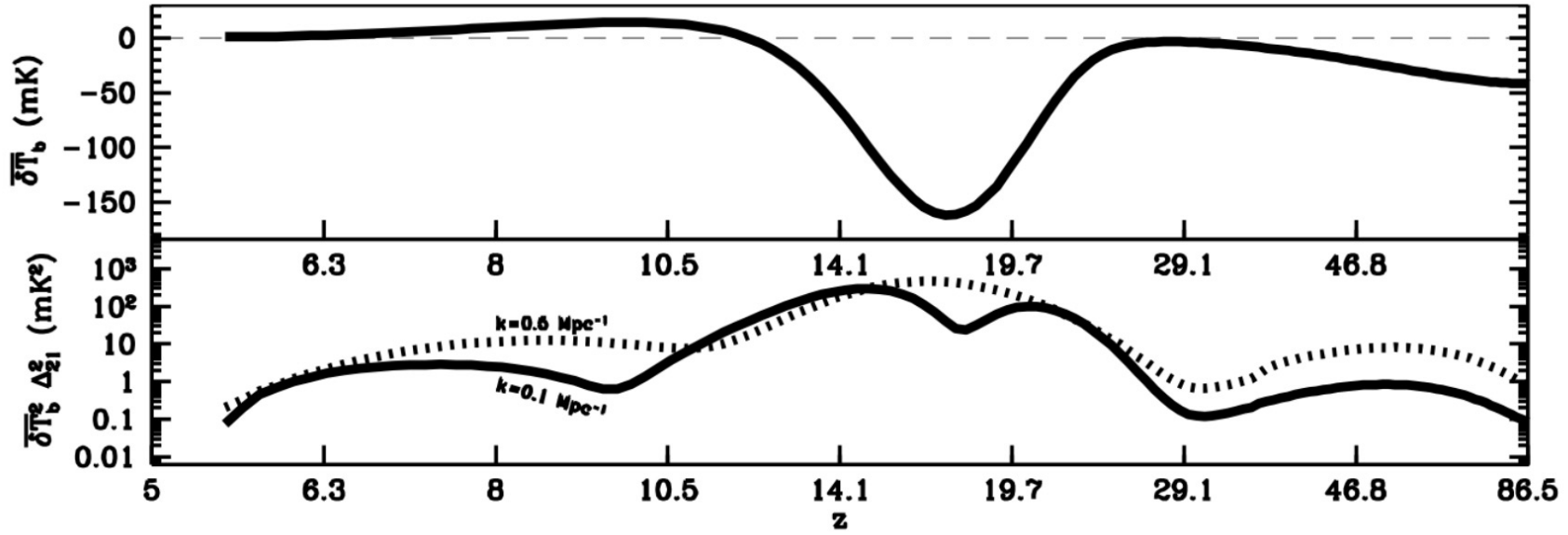


21cm signal





Reionization X-ray heating Lyα coupling Dark Ages



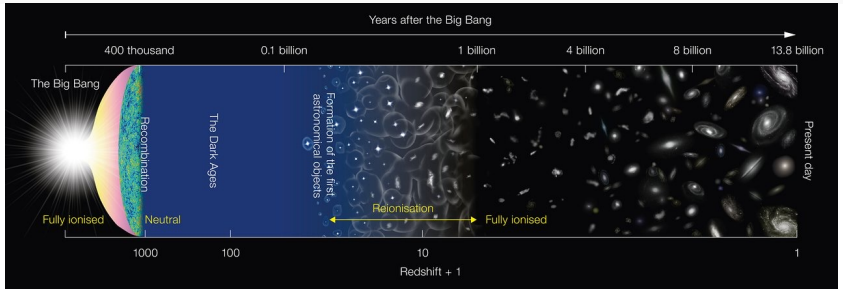
3D lighcone

->1D curve

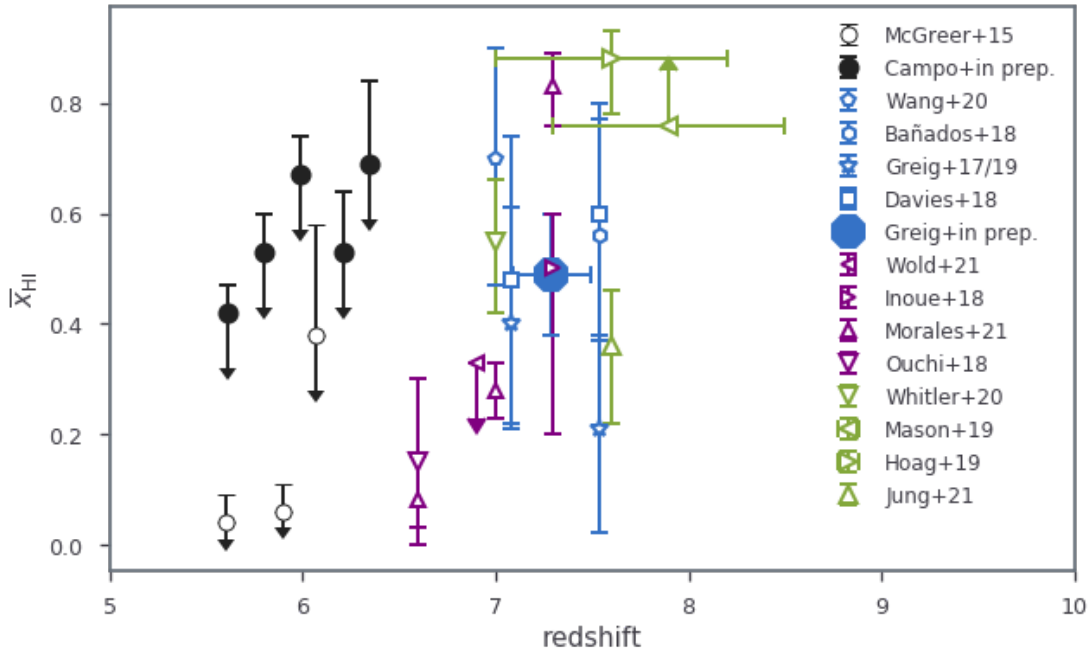
Or several 1D curves

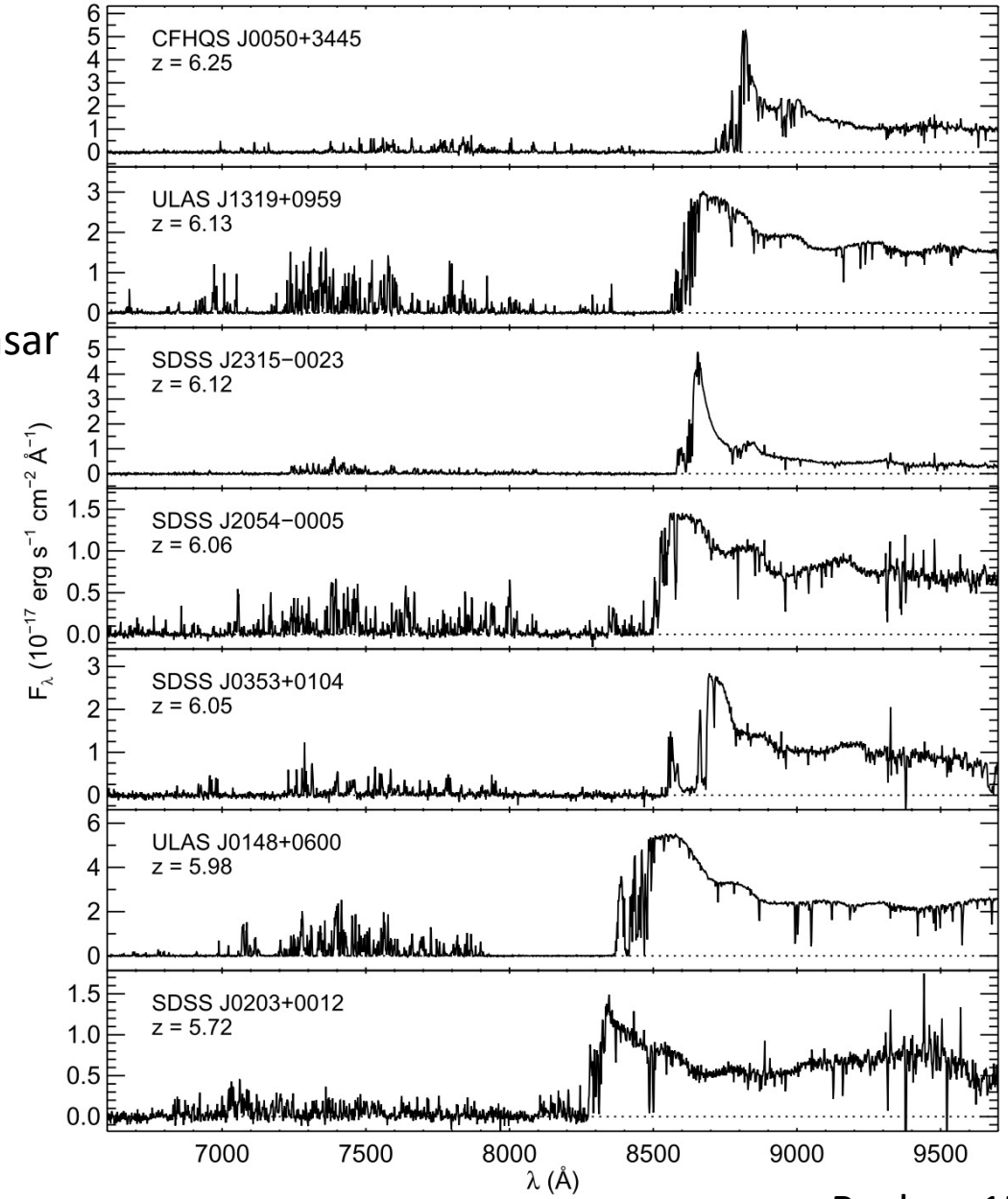
Mesinger+16

Neutral hydrogen fraction of the IGM



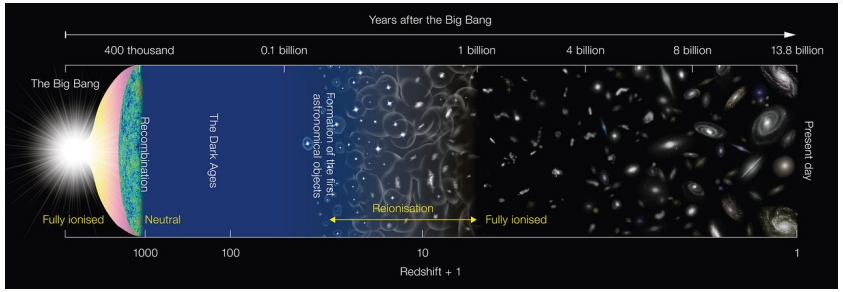
One example of using quasar spectra to constrain xHI.
1D curve -> a number

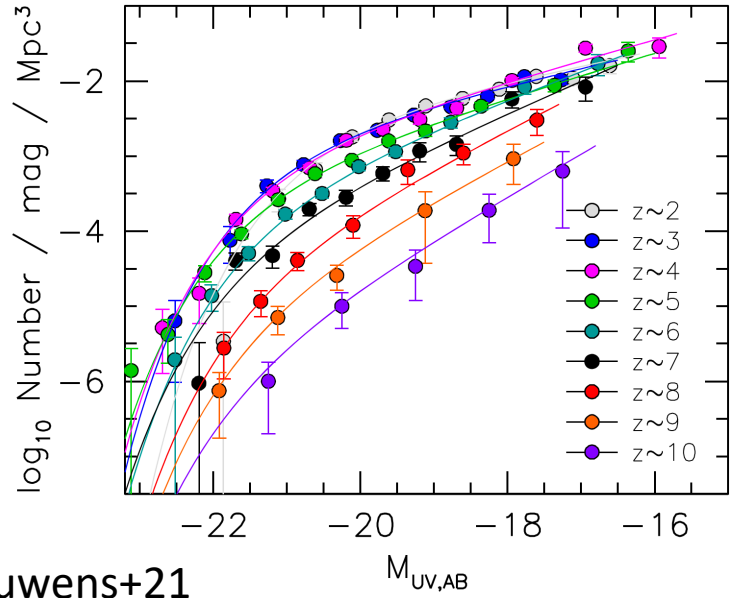




Becker+15

Galaxy population



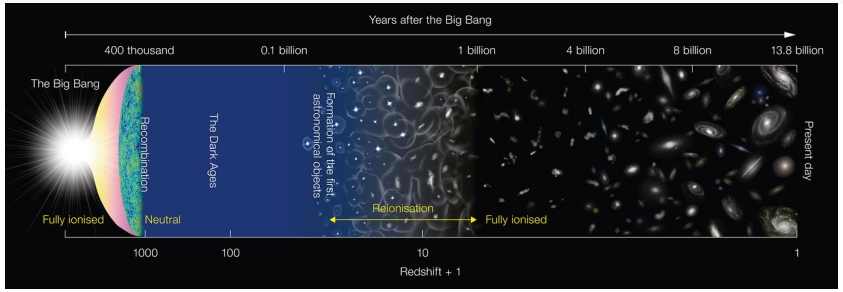


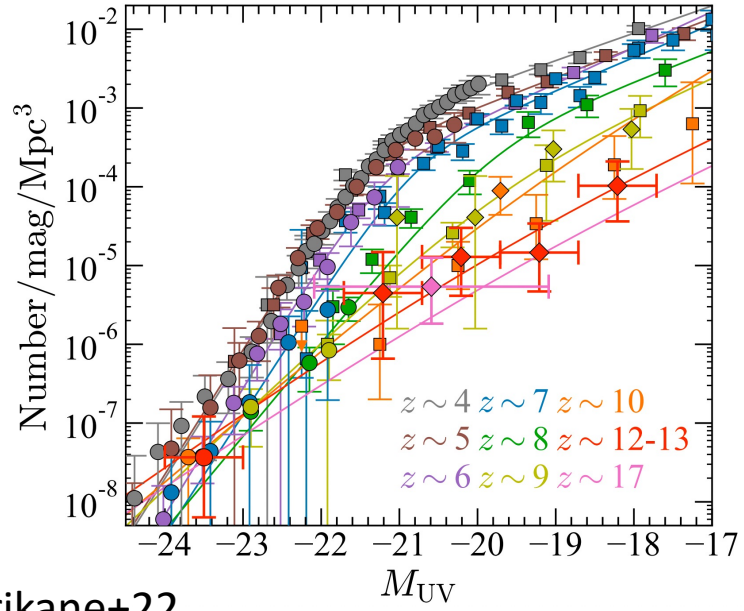
Bouwens+21

several 1D curves <- 3D lightcone

Hubble Ultra Deep Field

Galaxy population





Harikane+22

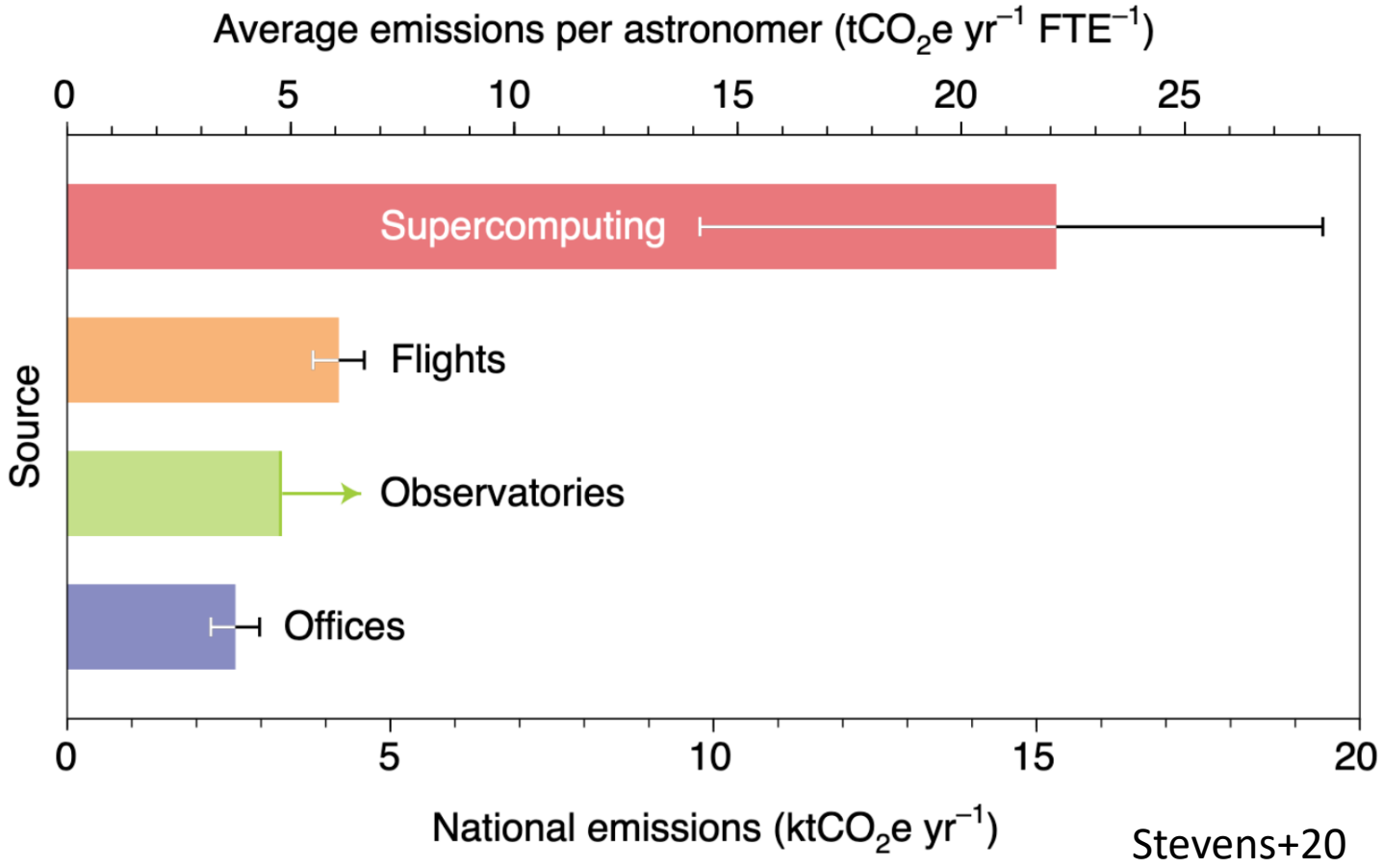
several 1D curves <- 3D lightcone

JWST SMACS 0723

Computational cost

Models often come with free parameters. Bayesian inference are preferred to determine parameter degeneracies or to select A model over others when compared to real data.

Bayesian inferences, however, are expensive as they require a sampling of model parameter space within a statistical framework.



Stevens+20

Emulator

To reduce the cost of forward modelling the observed summary statistics while sampling a high dimensional parameter space, emulating the model with simpler model can help.

- Subgrid physics (sometimes can still be expensive)
- Fitting functions (sometimes are difficult to find/fit)
- Neural networks
  - its complex architecture is able to fit a lot of summary statistics
  - It is still relatively simple as its essential part is a combination of a number of functions that have simple form
  - But, when model changes, network has to be retrained as well.
  - So it’s more useful for cases where the model is stable but observation gets updated frequently.

Neural network

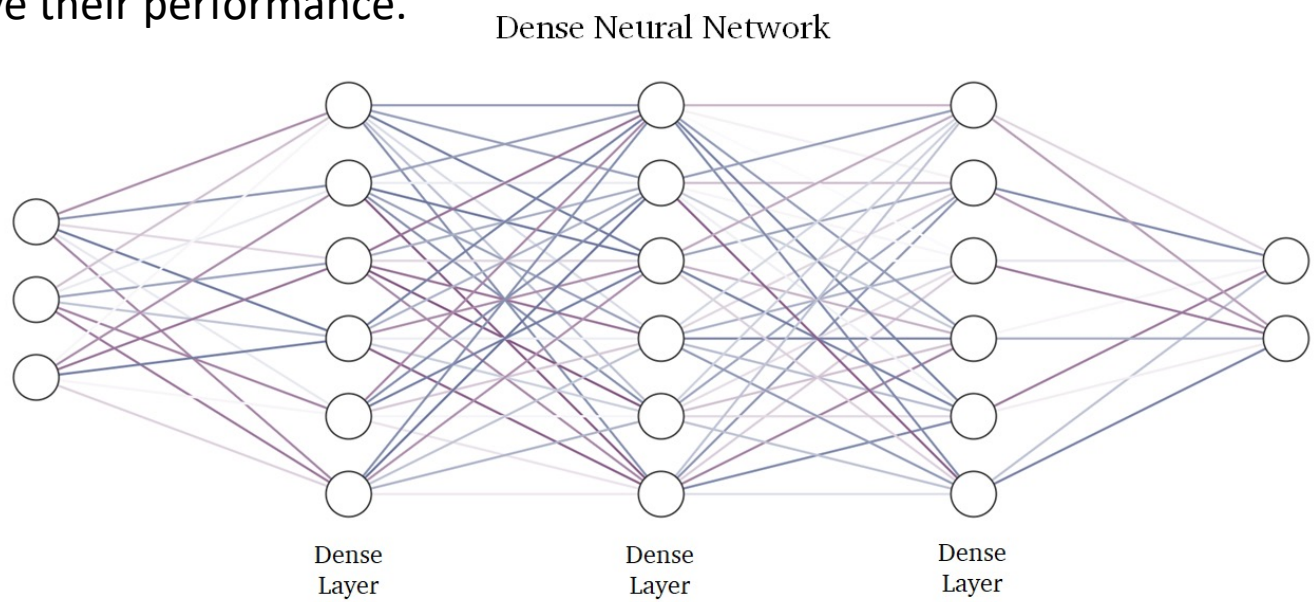
Outline:
- Neurons
- Activation function
- Types of neural networks
- Training neural networks

A method of computing, based on the interaction of multiple connected processing elements.

A powerful technique to solve many real world problems.

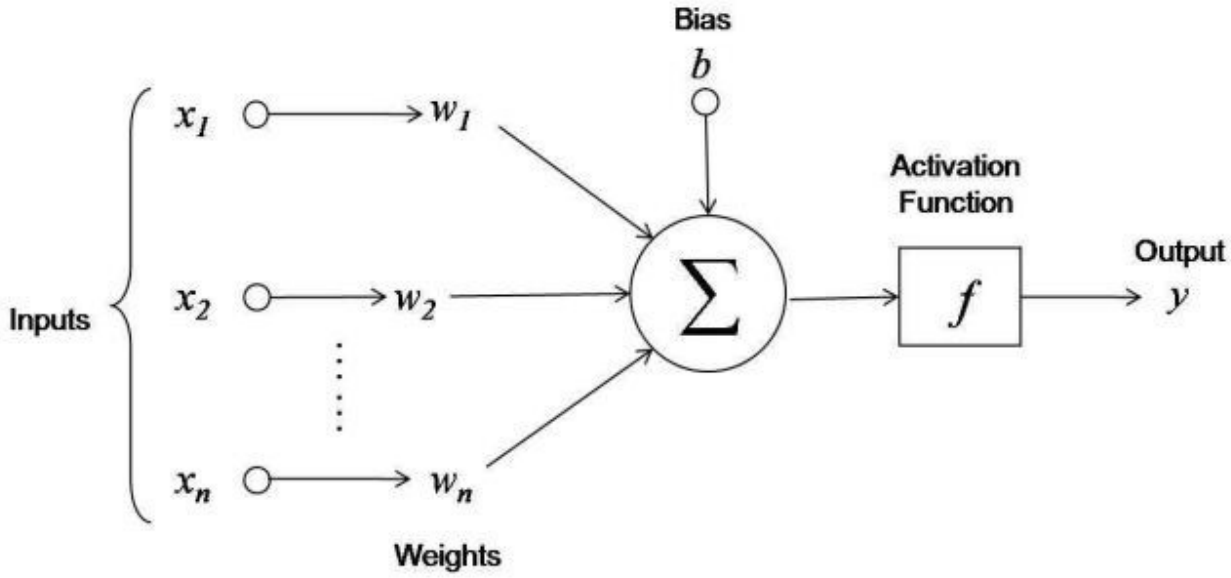
The ability to learn from experience in order to improve their performance.

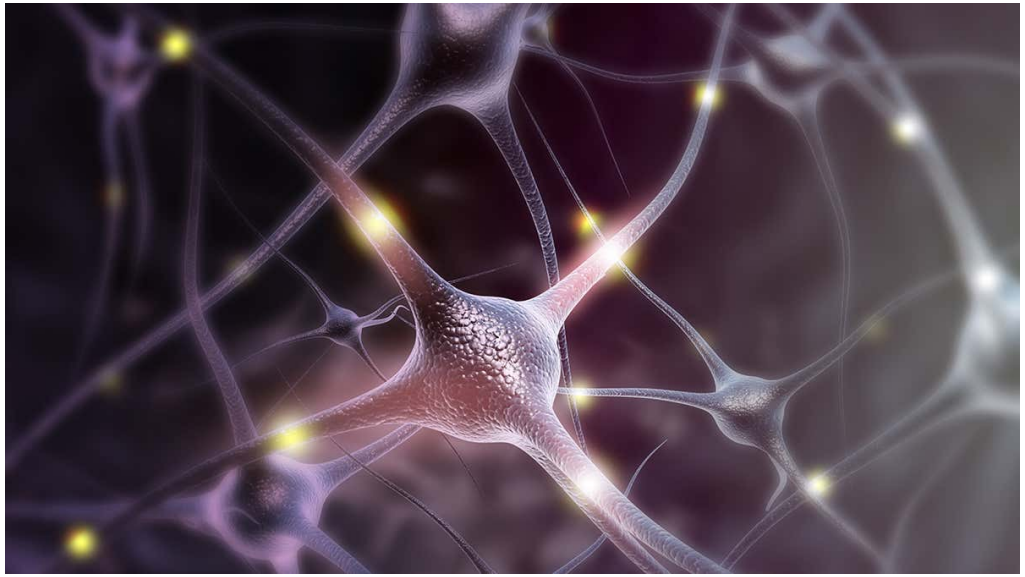
The ability to deal with incomplete information.



Neural network - Neurons

- Receives inputs (an N-D array)
- Multiples each input by its weight (coefficient)
- Applies activation function to the sum
- Output result



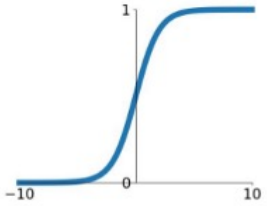


Neural network – Activation functions

- Controls when the unit is active or inactive.
- Adds non-linearity to the neural network (without activation function, it's just a linear regression model)

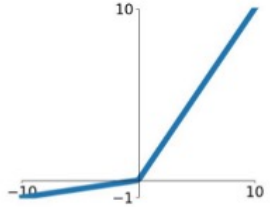
Sigmoid

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



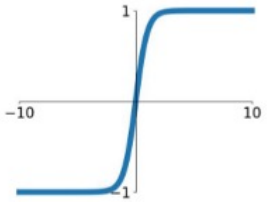
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

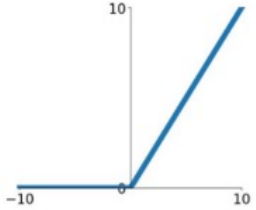


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

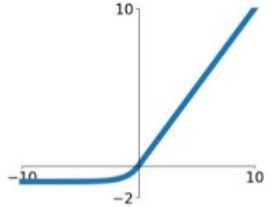
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

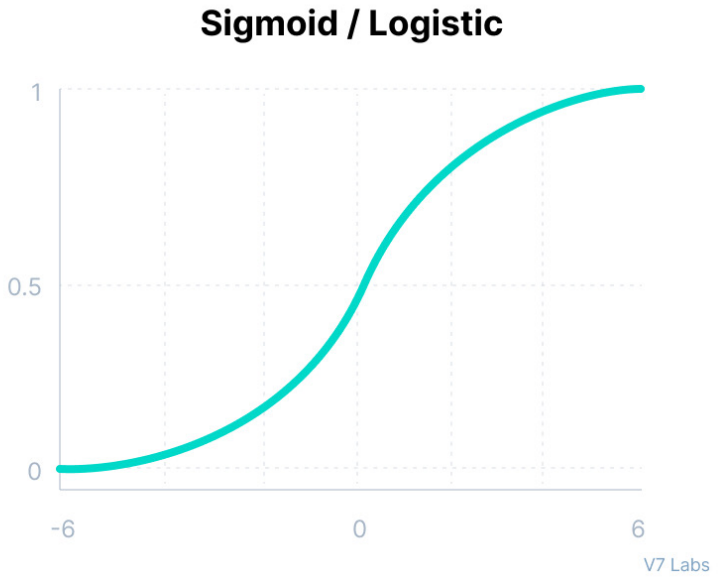


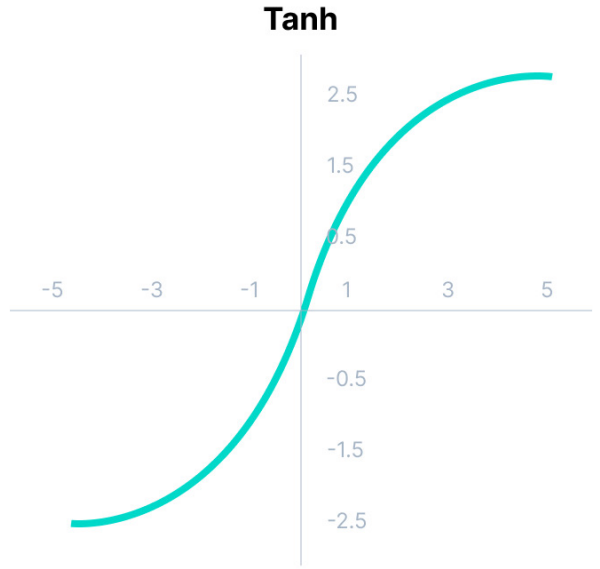
Neural network – Activation functions

Two issues to consider when choosing activation functions

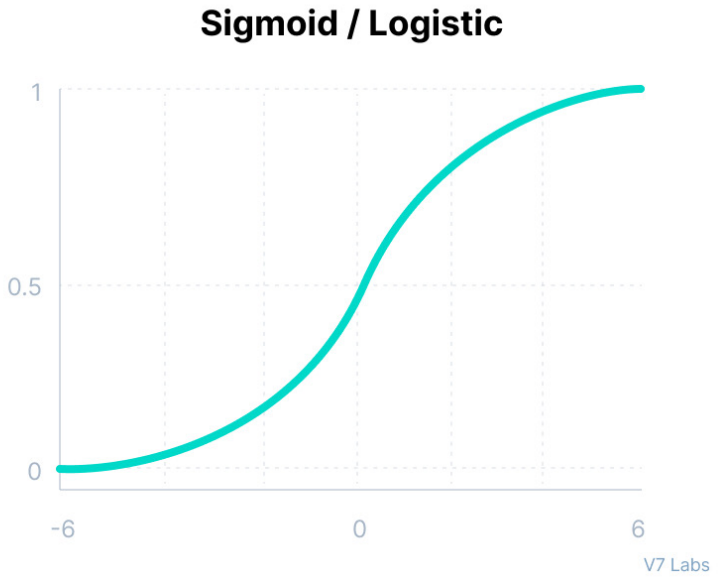
- Zero center or not

  The output of the tanh activation function is zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.





Neural network – Activation functions



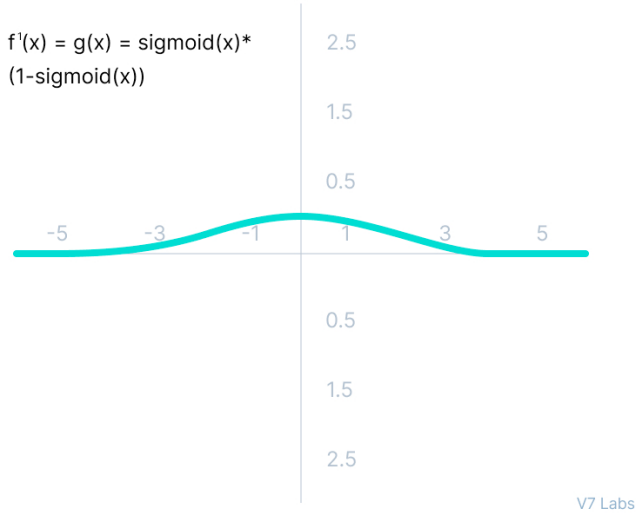
Two issues to consider when choosing activation functions

- Zero center or not

  The output of the tanh activation function is zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.

- Vanishing gradient

  As the gradient value approaches zero, the network ceases to learn.



ELU

Neural network – Activation functions

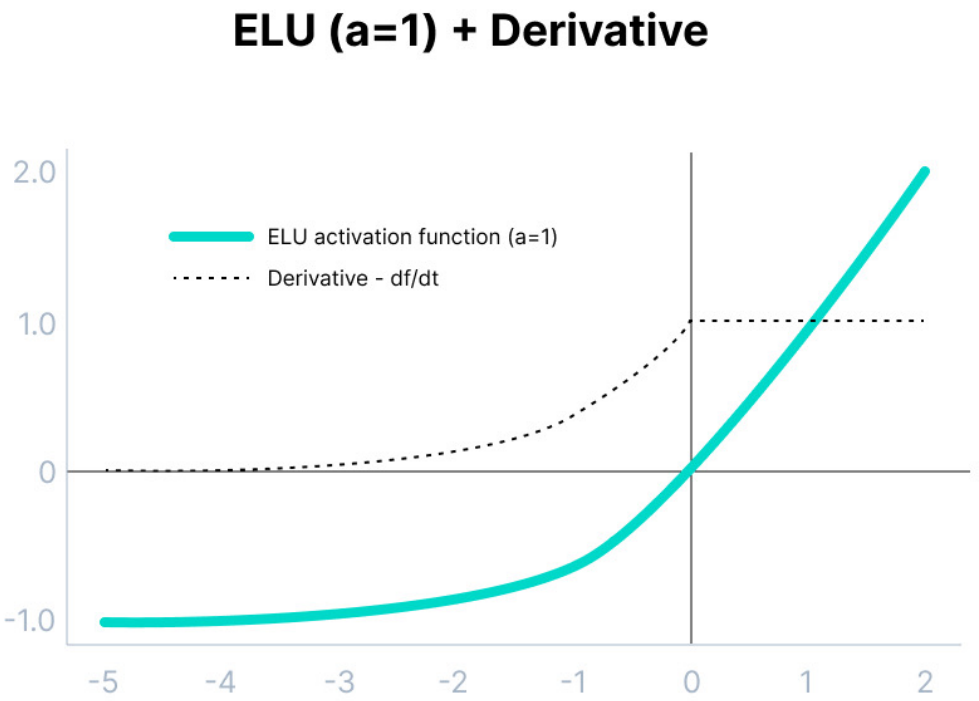
$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha\left(e^x - 1\right) & for\ x < 0 \end{cases}$$

Two issues to consider when choosing activation functions

- Zero center or not
  The output of the tanh activation function is zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.

- Vanishing gradient
  As the gradient value approaches zero, the network ceases to learn.

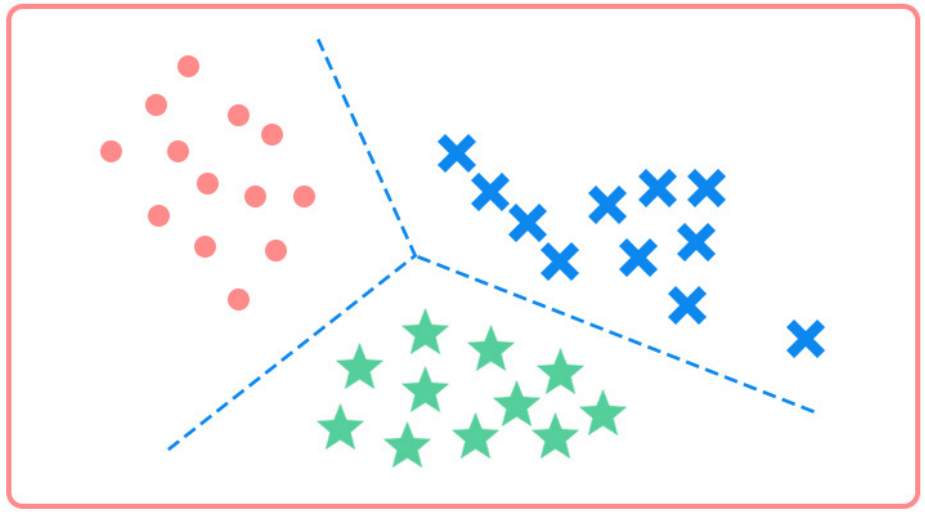
- Exploding gradient
  Error gradient can accumulate and result in very large updates to the weight, causing overflow and hence unstable network.

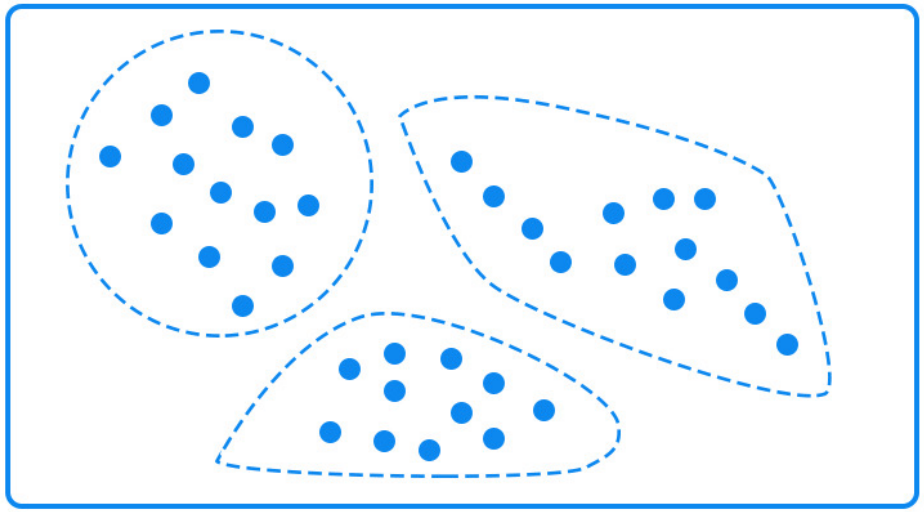


Neural network - Types

Supervised or not

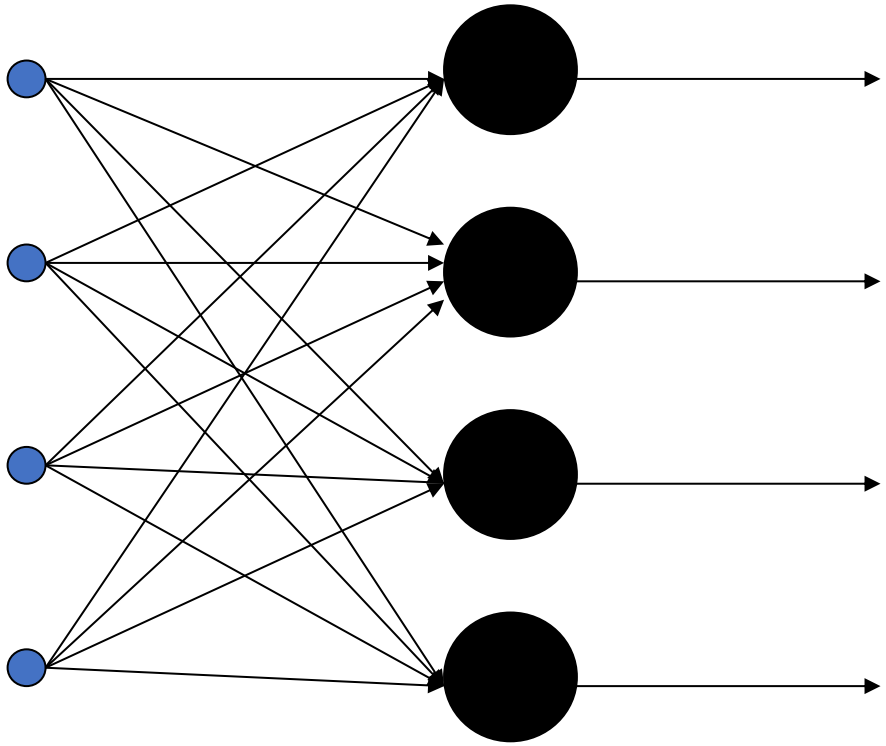
- Pre-categorized data or unlabeled data
- Interested in predictions or patterns
- Classification (divide the socks by color) or clustering (divide by similarity)
- Regression (divide the ties by length) or association (identify sequences)



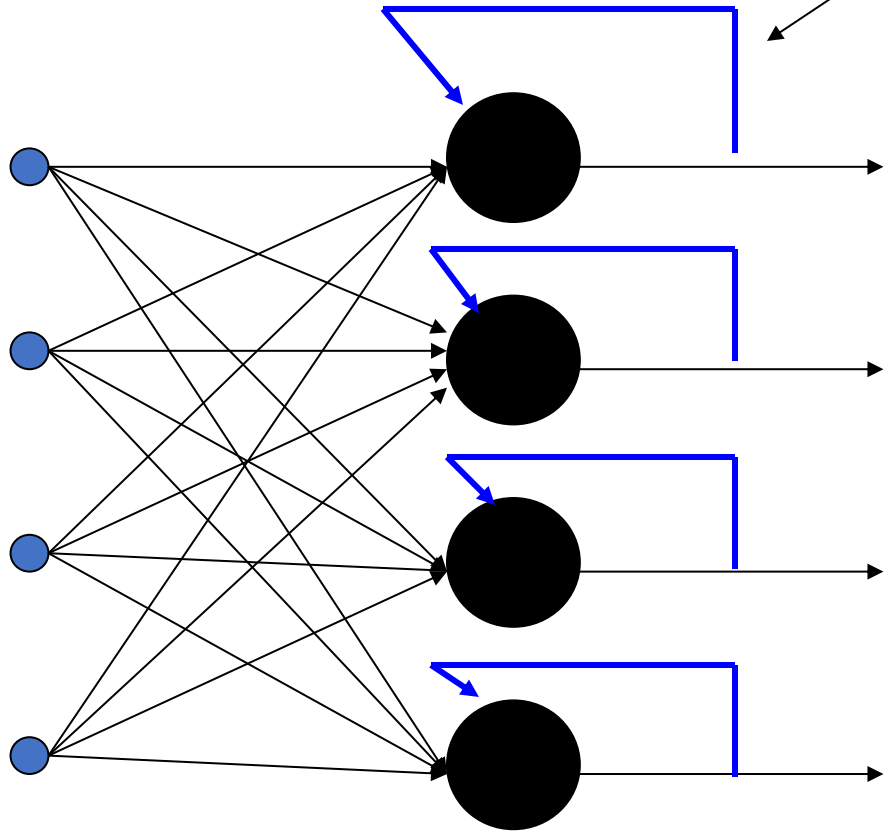


Neural network - Types

Recurrent or not

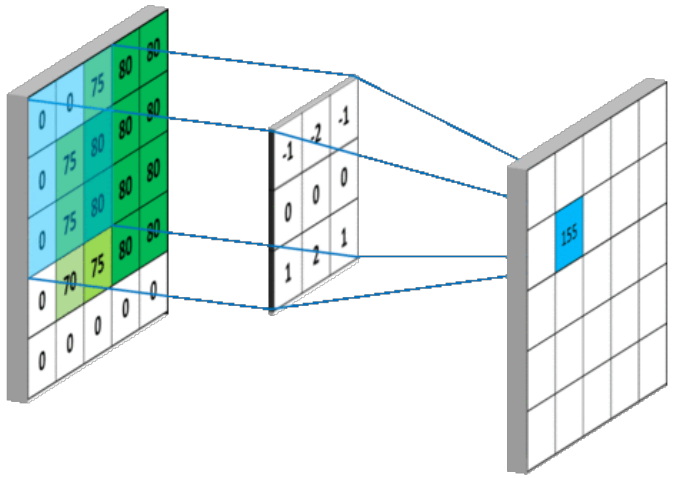


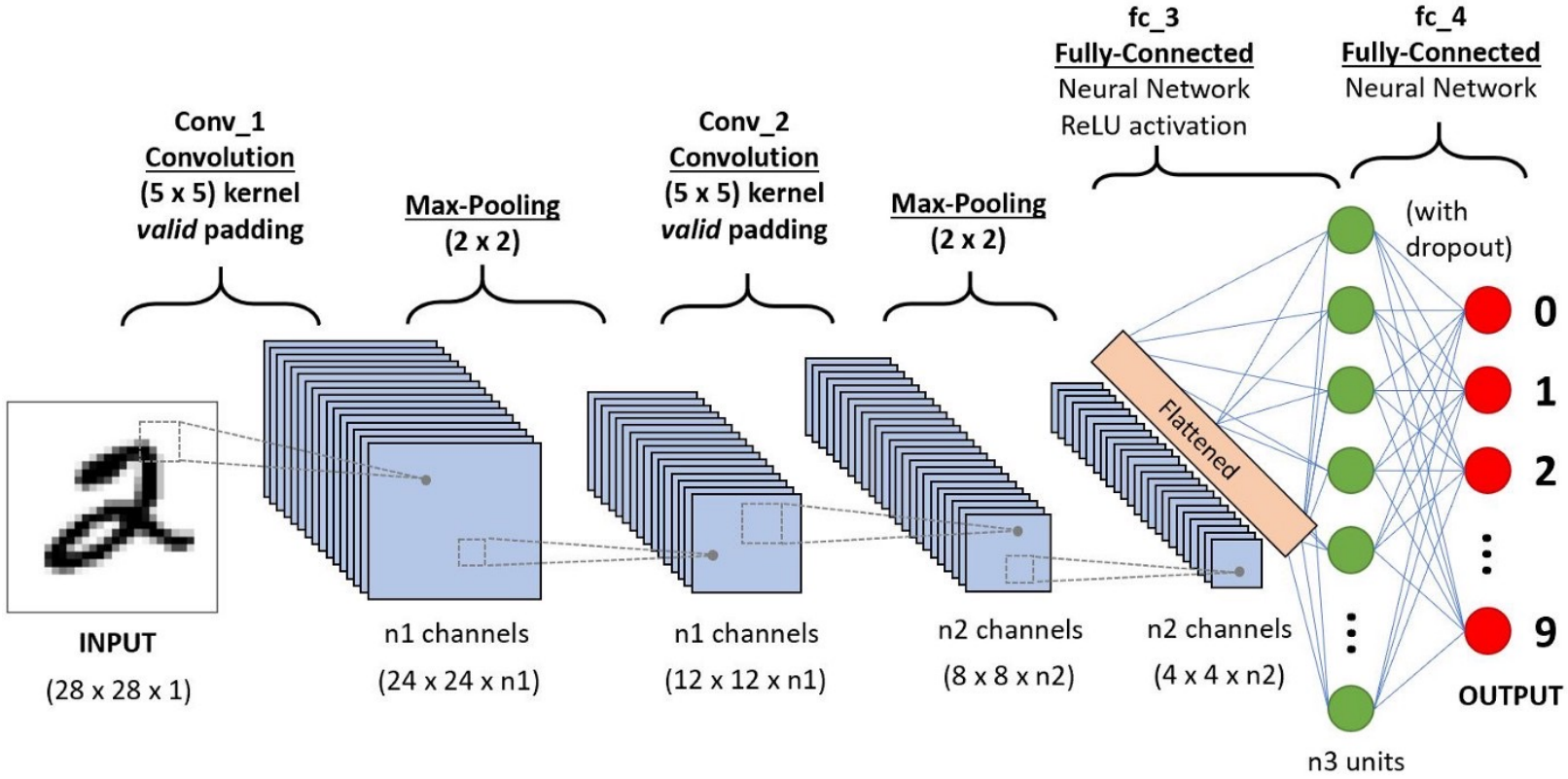
Feedback



Neural network - Types

1D or >1D



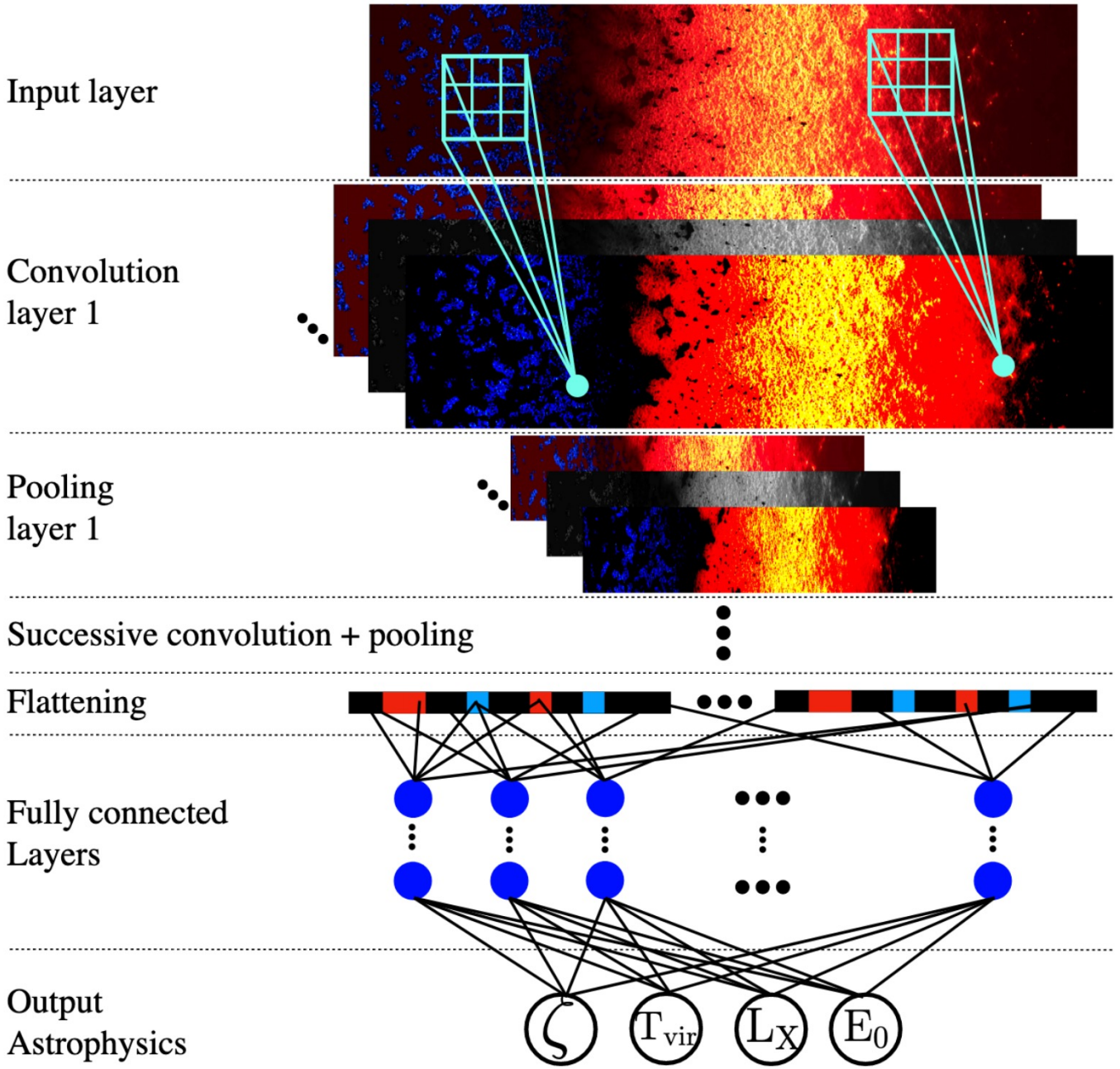


Neural network

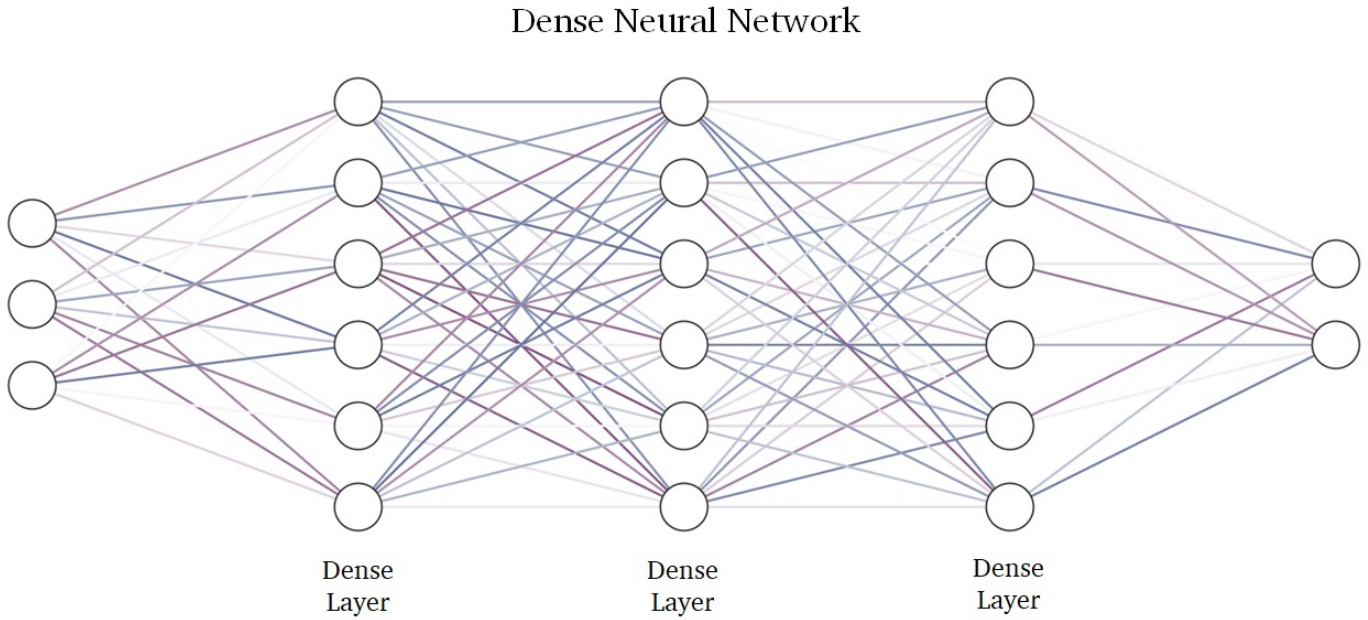
Inverse model since there are more information in the map that the input model parameters.

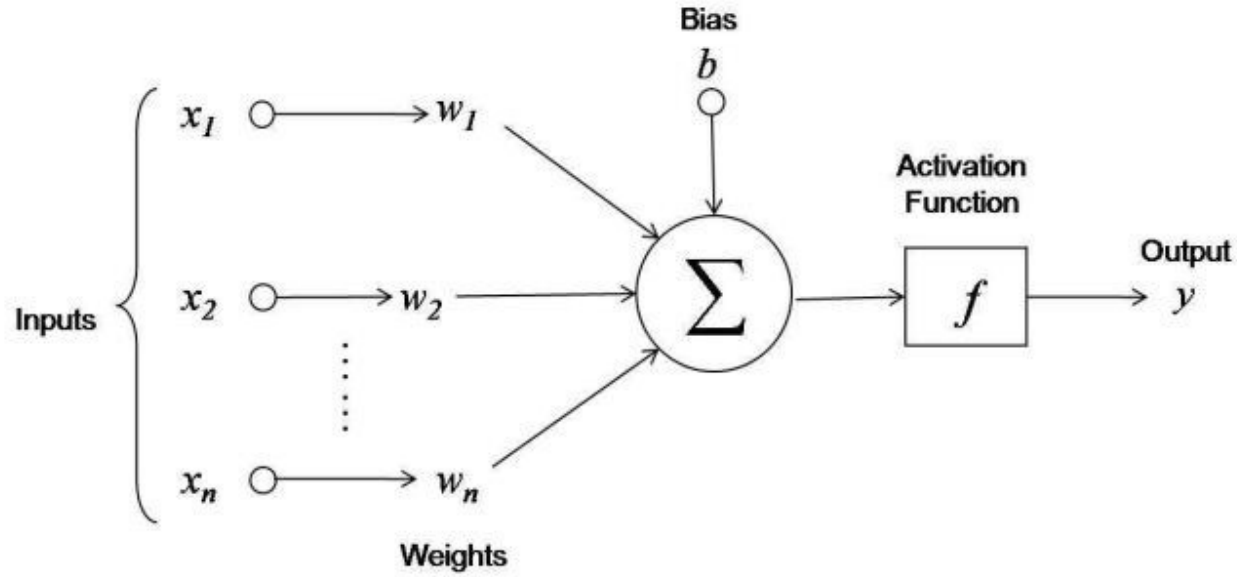
This can be used as an interpreter instead of an emulator.

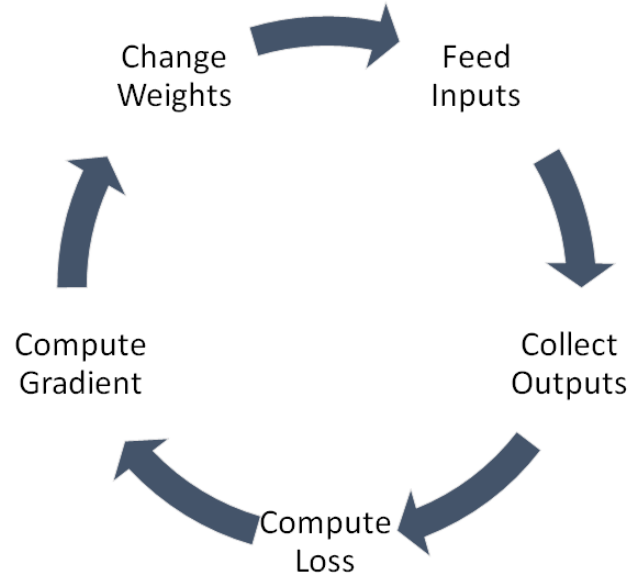
Forward model from N-dimensional parameter space to 3D lightones / 2D maps is more challenging.



Neural network - Training







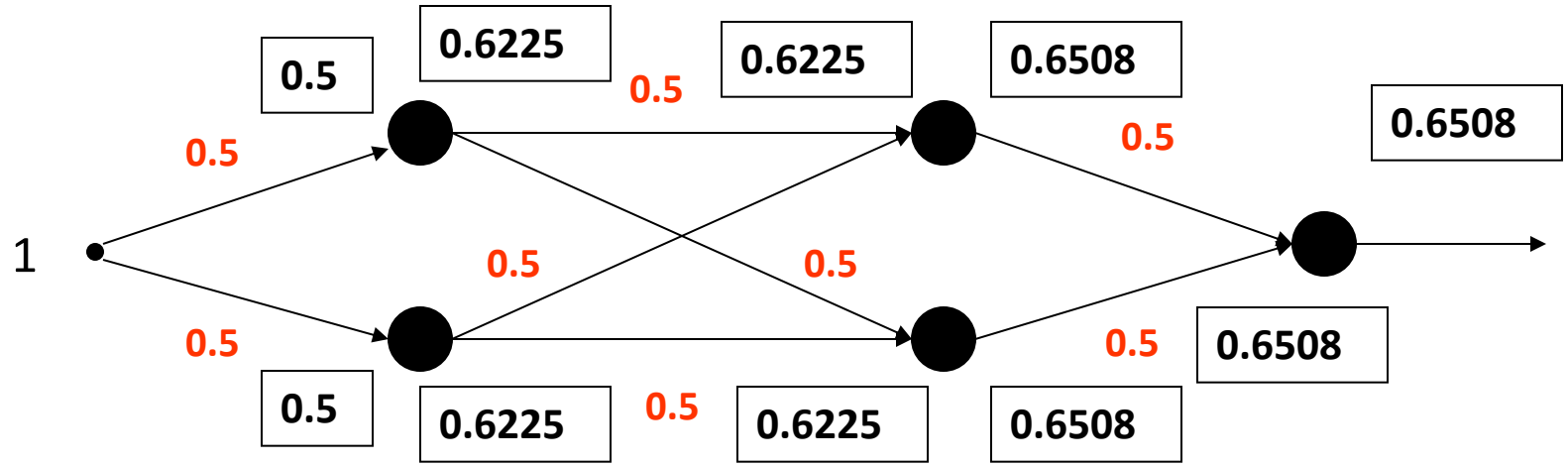
Neural network - Training

backpropagation of errors using gradient descent training
current weight of each neuron
updated weight of each neuron
gradient at each layer

Input & output

G1= (0.6225)(1-0.6225)(0.0397)(0.5)(2)=0.0093

G2= (0.6508)(1-0.6508)(0.3492)(0.5)=0.0397



G3=0.3492

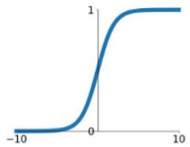
Gradient of the neuron= **G(i)** = Σ{ (output of the neuron) × slope of the activation function× **G(i+1)** × (weight of the neuron to the next neuron) }

Gradient of the output neuron = error

New Weight=Old Weight + {(learning rate)(gradient)(prior output)}

Sigmoid

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



$\frac{d\sigma}{dx} = 1 - \sigma$

Error=1-0.6508=0.3492

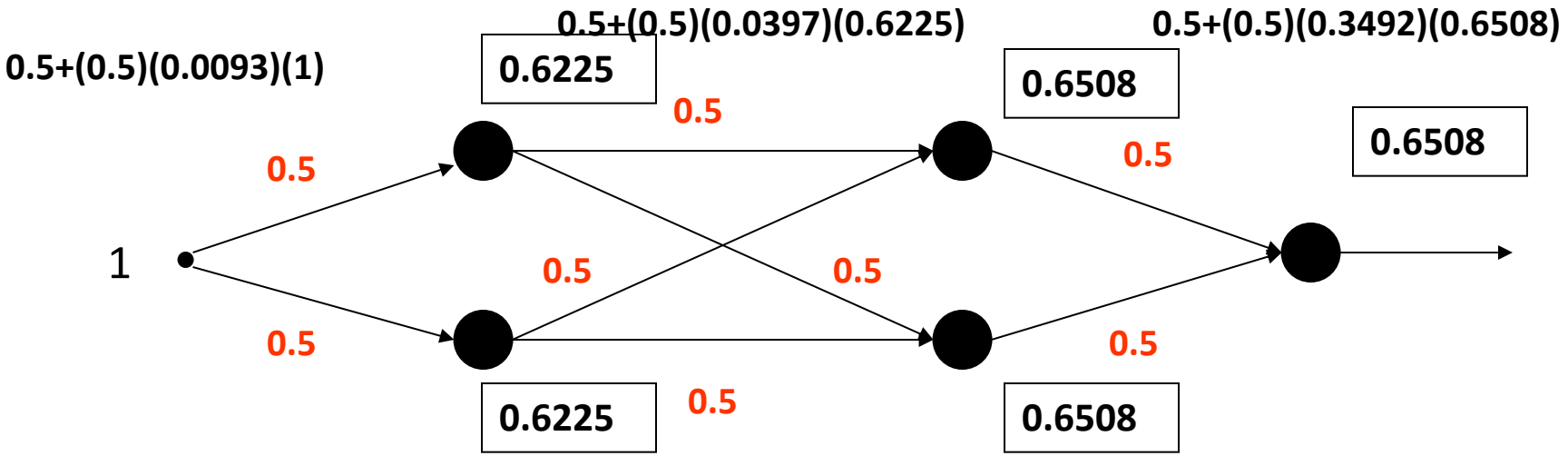
Neural network - Training

backpropagation of errors using gradient descent training
current weight of each neuron
updated weight of each neuron
gradient at each layer

Input & output

G1= (0.6225)(1-0.6225)(0.0397)(0.5)(2)=0.0093

G2= (0.6508)(1-0.6508)(0.3492)(0.5)=0.0397



G3=0.3492

Gradient of the neuron= **G(i)** = $\Sigma\{$ (output of the neuron) × slope of the activation function× **G(i+1)** × (weight of the neuron to the next neuron) $\}$

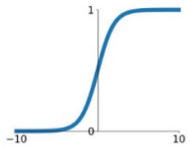
Gradient of the output neuron = error

New Weight=Old Weight + {(learning rate)(gradient)(prior output)}

learning rate is assumed to be 0.5

Sigmoid

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



$\frac{d\sigma}{dx} = 1 - \sigma$

Neural network - Training

backpropagation of errors using gradient descent training
current weight of each neuron
updated weight of each neuron
gradient at each layer

Input & output

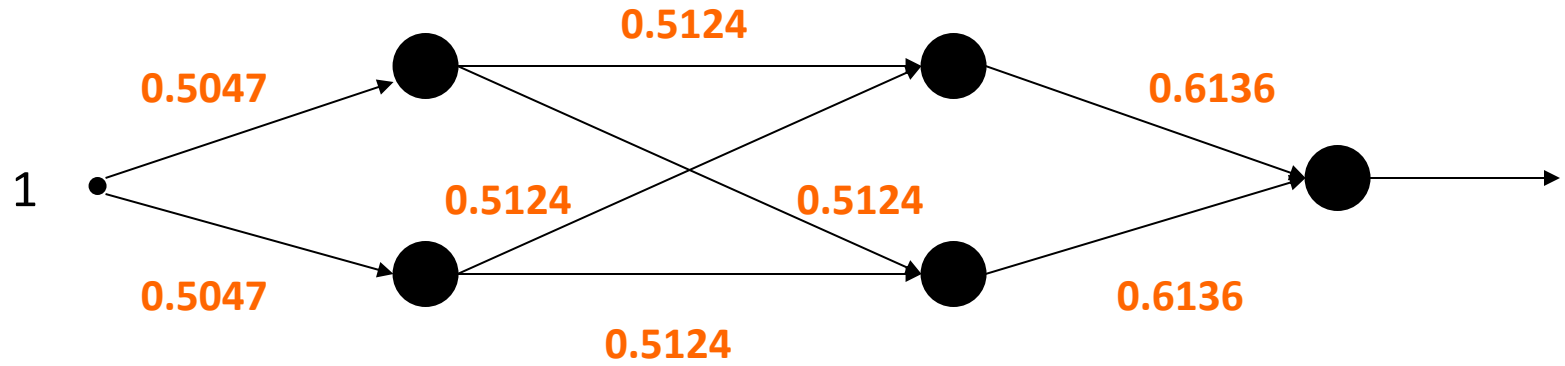
G1= (0.6225)(1-0.6225)(0.0397)(0.5)(2)=0.0093

G2= (0.6508)(1-0.6508)(0.3492)(0.5)=0.0397

0.5+(0.5)(0.0093)(1)

0.5+(0.5)(0.0397)(0.6225)

0.5+(0.5)(0.3492)(0.6508)



G3=0.3492

Gradient of the neuron= **G(i)** = $\Sigma\{$ (output of the neuron) × slope of the activation function× **G(i+1)** × (weight of the neuron to the next neuron) $\}$

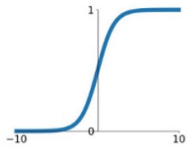
Gradient of the output neuron = error

New Weight=Old Weight + {(learning rate)(gradient)(prior output)}

learning rate is assumed to be 0.5

Sigmoid

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



$\frac{d\sigma}{dx} = 1 - \sigma$

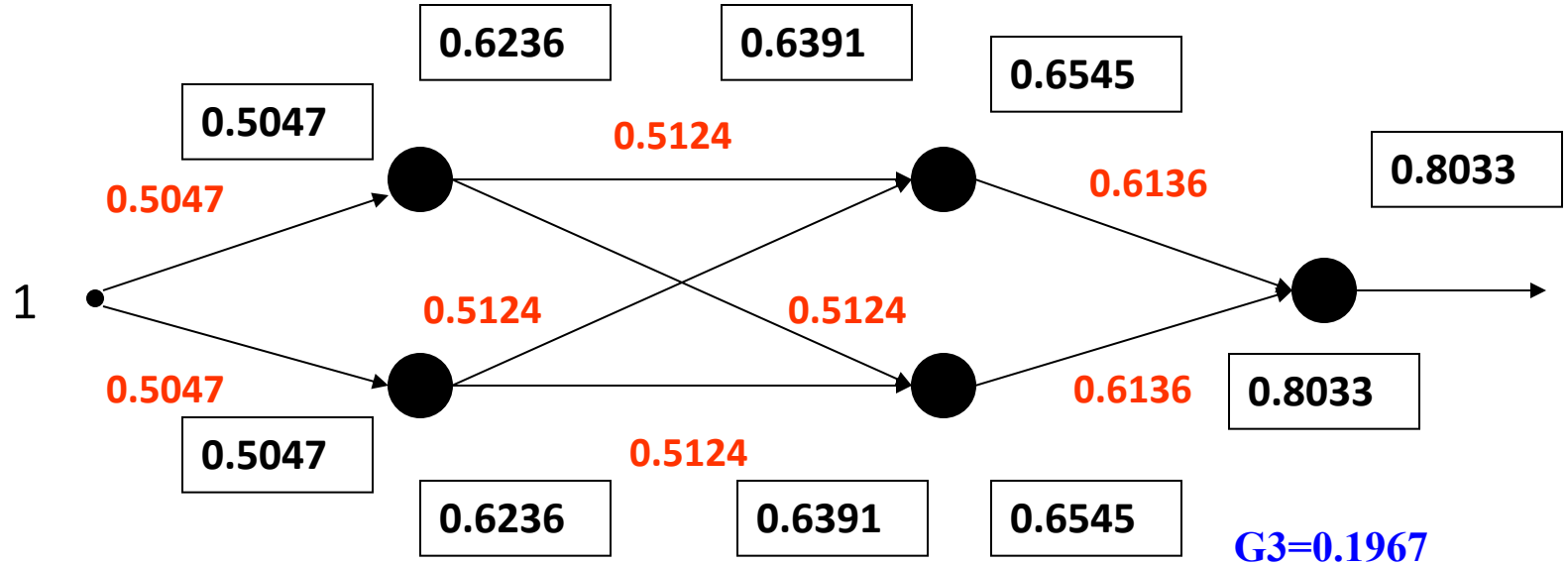
Neural network - Training

backpropagation of errors using gradient descent training
current weight of each neuron
updated weight of each neuron
gradient at each layer

Input & output

G1= (0.6236)(1-0.6236)(0.5124)(0.0273)(2)=0.0066

G2= (0.6545)(1-0.6545)(0.1967)(0.6136)=0.0273



G3=0.1967

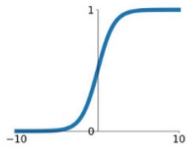
Gradient of the neuron= **G(i)** = Σ{ (output of the neuron) × slope of the activation function× **G(i+1)** × (weight of the neuron to the next neuron) }

Gradient of the output neuron = error

New Weight=Old Weight + {(learning rate)(gradient)(prior output)}

Sigmoid

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



$\frac{d\sigma}{dx} = 1 - \sigma$

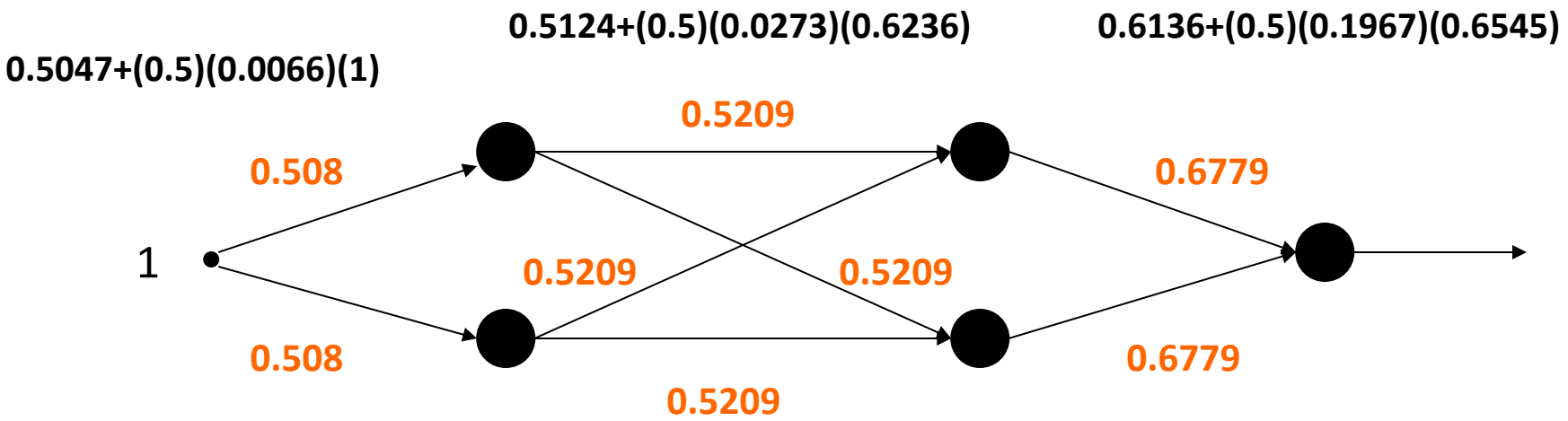
Error=1-0.8033=0.1967

learning rate is assumed to be 0.5

Neural network - Training

backpropagation of errors using gradient descent training
current weight of each neuron
updated weight of each neuron
gradient at each layer

Input & output



Gradient of the neuron= **G(i)** = $\Sigma\{$ (output of the neuron) × slope of the activation function× **G(i+1)** × (weight of the neuron to the next neuron) $\}$

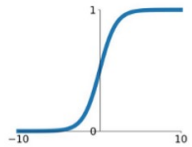
Gradient of the output neuron = error

New Weight=Old Weight + {(learning rate)(gradient)(prior output)}

learning rate is assumed to be 0.5

Sigmoid

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

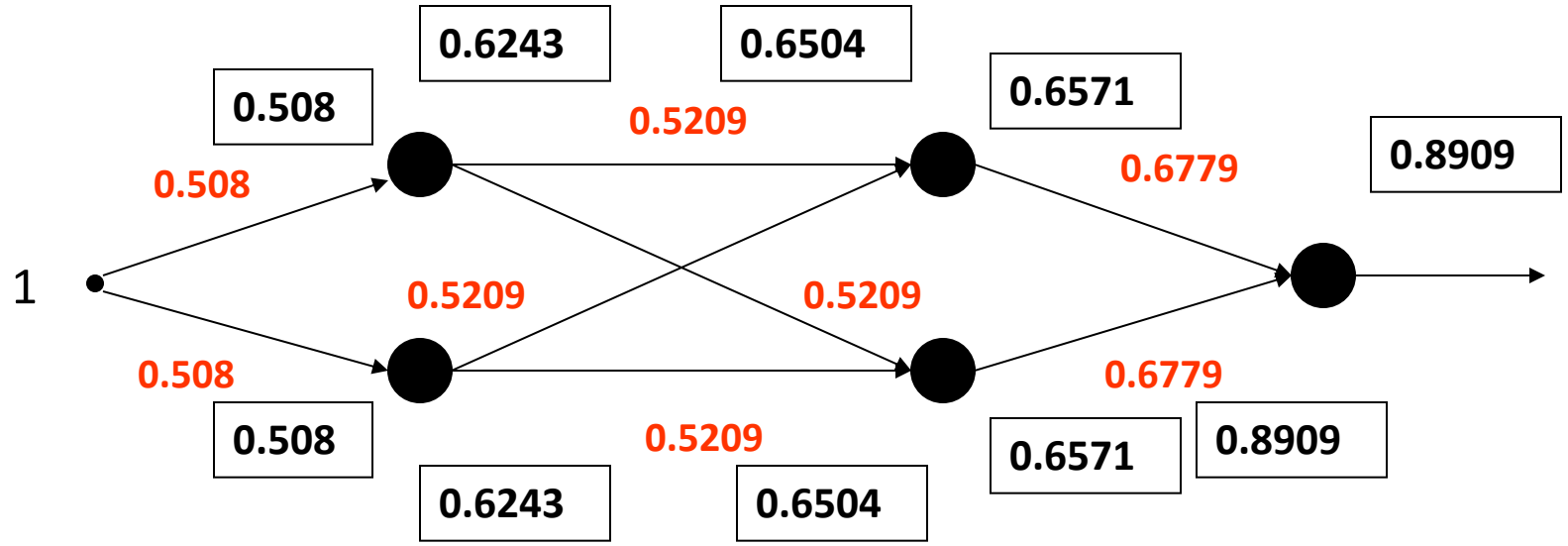


$\frac{d\sigma}{dx} = 1 - \sigma$

Neural network - Training

backpropagation of errors using gradient descent training
current weight of each neuron
updated weight of each neuron
gradient at each layer

Input & output



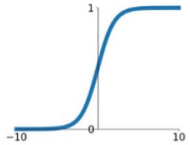
Gradient of the neuron= **G(i)** = $\Sigma\{$ (output of the neuron) × slope of the activation function× **G(i+1)** × (weight of the neuron to the next neuron) $\}$

Gradient of the output neuron = error

New Weight=Old Weight + {(learning rate)(gradient)(prior output)}

Sigmoid

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



$\frac{d\sigma}{dx} = 1 - \sigma$

Error=1-0.8909=0.1091

learning rate is assumed to be 0.5

Neural network - Training

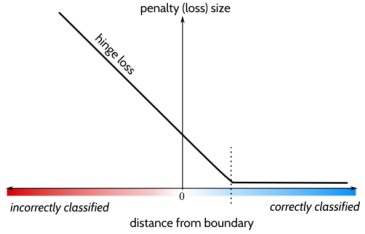
	Weights			Output	Expected (	Error
	w1	w2	w3			
Initial conditions	0.5	0.5	0.5	0.6508	1	0.3492
Pass 1 Update	0.5047	0.5124	0.6136	0.8033	1	0.1967
Pass 2 Update	0.508	0.5209	0.6779	0.8909	1	0.1091

W1: Weights from the input to the input layer
W2: Weights from the input layer to the hidden layer
W3: Weights from the hidden layer to the output layer

Neural network – Loss Function

	Weights			Output	Expected	Error
	w1	w2	w3			
Initial conditions	0.5	0.5	0.5	0.6508	1	0.3492
Pass 1 Update	0.5047	0.5124	0.6136	0.8033	1	0.1967
Pass 2 Update	0.508	0.5209	0.6779	0.8909	1	0.1091

- The process continues until the required loss has been reached.
- Loss function is used to quantify how good or bad the model is performing.
- Training + Validation + Test set
- Regression Loss Function
  - Mean Squared Error Loss (MSE): $L(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
  - Mean Squared Logarithmic Error Loss (MSLE): $L(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}[\log(y_i + 1) - \log(\hat{y}_i + 1)] = \frac{1}{N}\sum_{i=1}^{N}\left(\log\frac{(y_i+1)}{\hat{y}_i+1}\right)$
  - Mean Absolute Error Loss (MAE): $L(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$
- Binary Classification Loss Function
  - $$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$
  - Hinge Loss

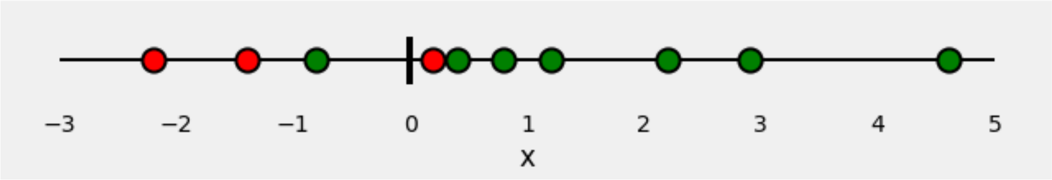


- Multi-Class Classification Loss Function

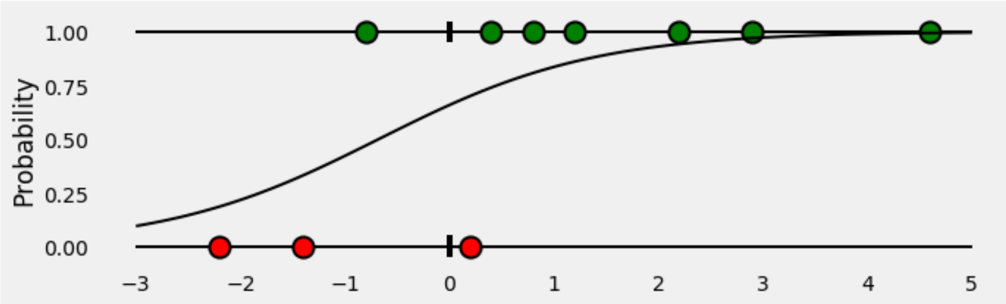
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

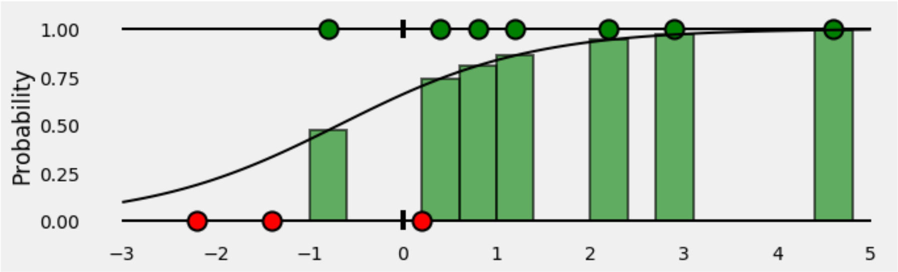
Neural network – Binary Cross Entropy Loss

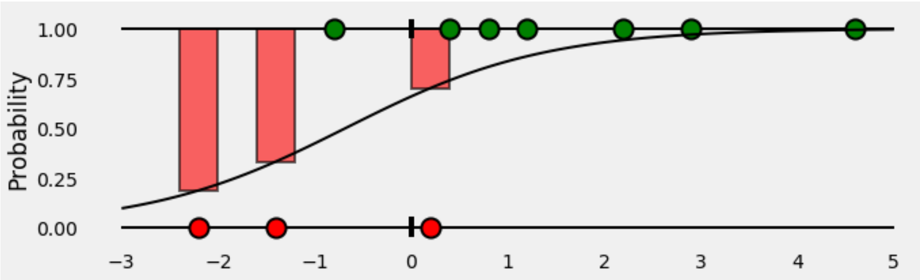
Task: classify the red and green points

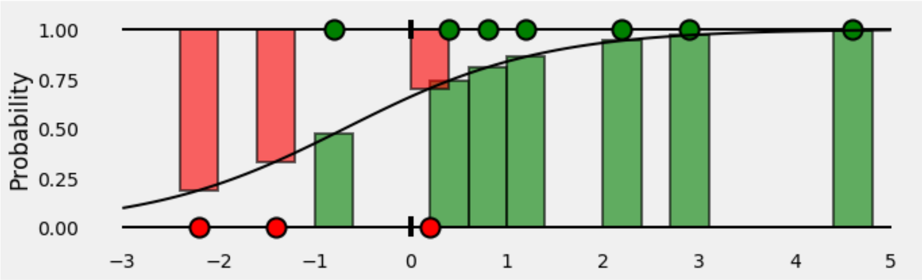


ML language -> what is the probability of the point being green?
i.e., P(green) = 1 and P(red)=0

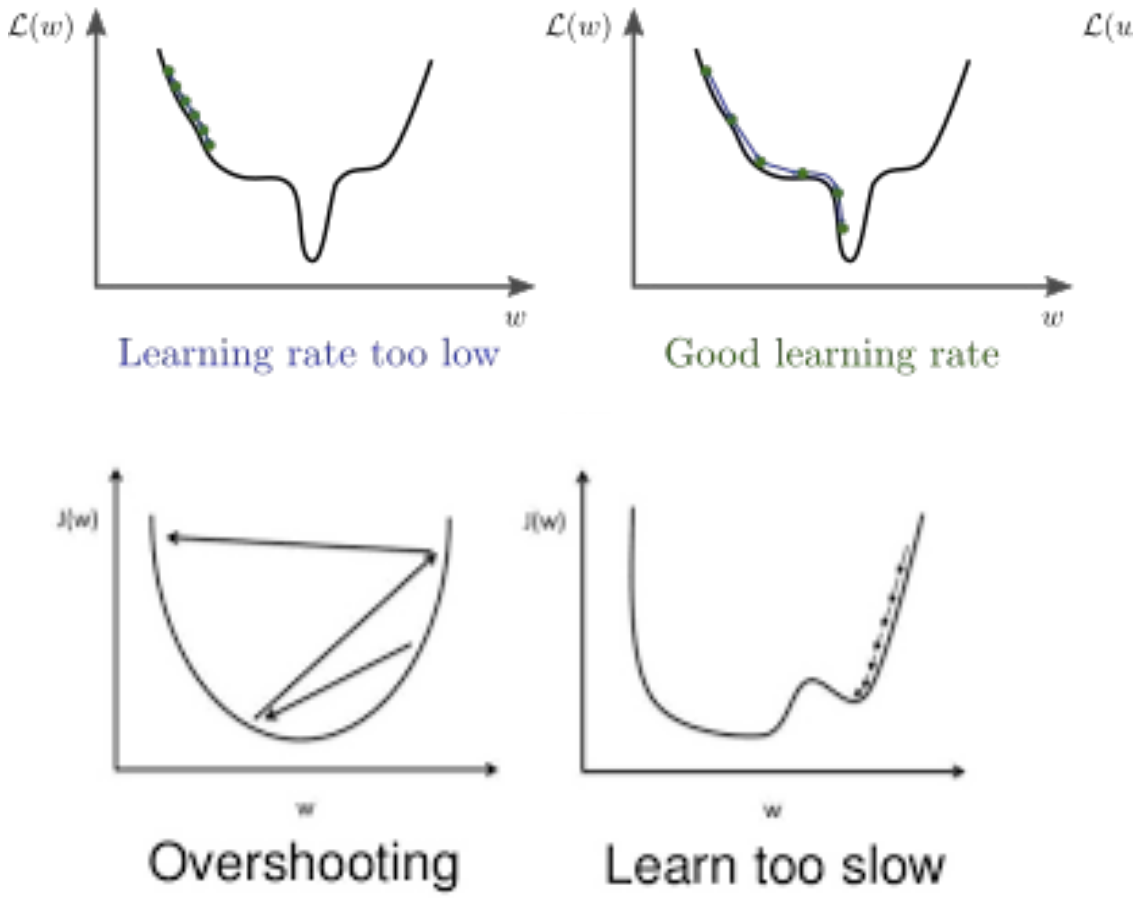




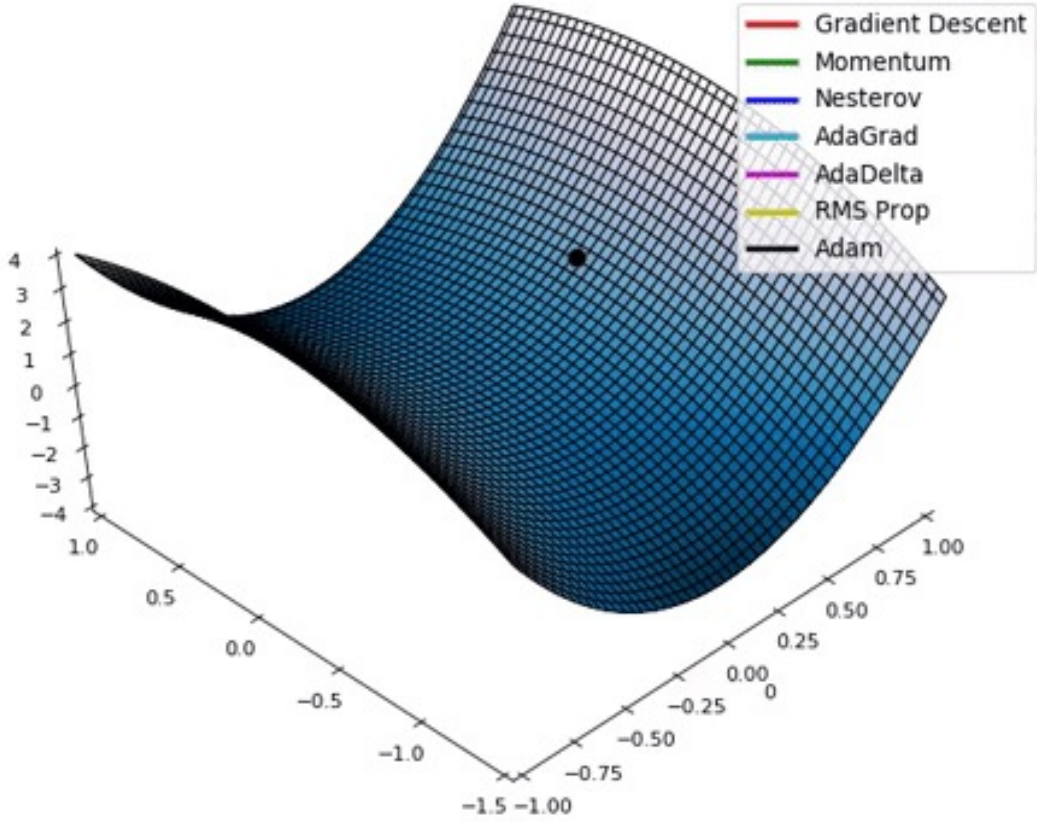




Neural network – learning rate and optimizer



- Adam (https://arxiv.org/abs/1412.6980)
  - Adaptive moment estimation
  - Optimizers modify the weights and learning rate



Neural network example – prepare dataset

```
import tensorflow.compat.v1 as tf
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam

# load training data (X_train, Y_train)
# load validation data (X_valid, Y_valid)
# covert them into tensorflow format
x_train = tf.data.Dataset.from_tensor_slices(X_train); x_val = tf.data.Dataset.from_tensor_slices(X_valid)
y_train = tf.data.Dataset.from_tensor_slices(Y_train); y_val = tf.data.Dataset.from_tensor_slices(Y_valid)

# This is the size of a single training batch (training is not done on the entire test set at once, but on batches of data)
batch_size = 64

# package the data
training_data = tf.data.Dataset.zip((x_train, y_train)).shuffle(X_train.shape[0]).batch(batch_size)
validation_data = tf.data.Dataset.zip((x_val, y_val)).shuffle(X_valid.shape[0]).batch(batch_size)

# fancy features
callbacks = [tf.keras.callbacks.ReduceLROnPlateau(patience = 20), # If loss does not improve for 20 epochs, reduce learning rate
    tf.keras.callbacks.EarlyStopping(patience = 50)] # If loss does not improve for 50 epochs, stop the learning
```

Neural network example – define network structure

```
# Input size = number of params
input_layer = layers.Input(shape = X_train.shape[-1])

# Hidden fully-connected layers
output = layers.Dense(64)(input_layer)
# Batch normalization = normalize the training weights at every batch.
output = layers.BatchNormalization()(output)
output = layers.ReLU()(output)
output = layers.Dense(128)(output)
output = layers.BatchNormalization()(output)
output = layers.ReLU()(output)
output = layers.Dense(64)(output)
output = layers.BatchNormalization()(output)
output = layers.ReLU()(output)
# Last layer output shape
output = layers.Dense(Y_train.shape[-1])(output)

model = tf.keras.Model(inputs = [input_layer], outputs = [output])
model.summary()
model.compile(loss = 'mse', optimizer = Adam(learning_rate = 1e-3))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 9)]               0
_________________________________________________________________
dense (Dense)                (None, 64)                640
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256
_________________________________________________________________
re_lu (ReLU)                 (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
re_lu_1 (ReLU)               (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
re_lu_2 (ReLU)               (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 54)                3510
=================================================================
Total params: 21,750
Trainable params: 21,238
Non-trainable params: 512
_________________________________________________________________
```

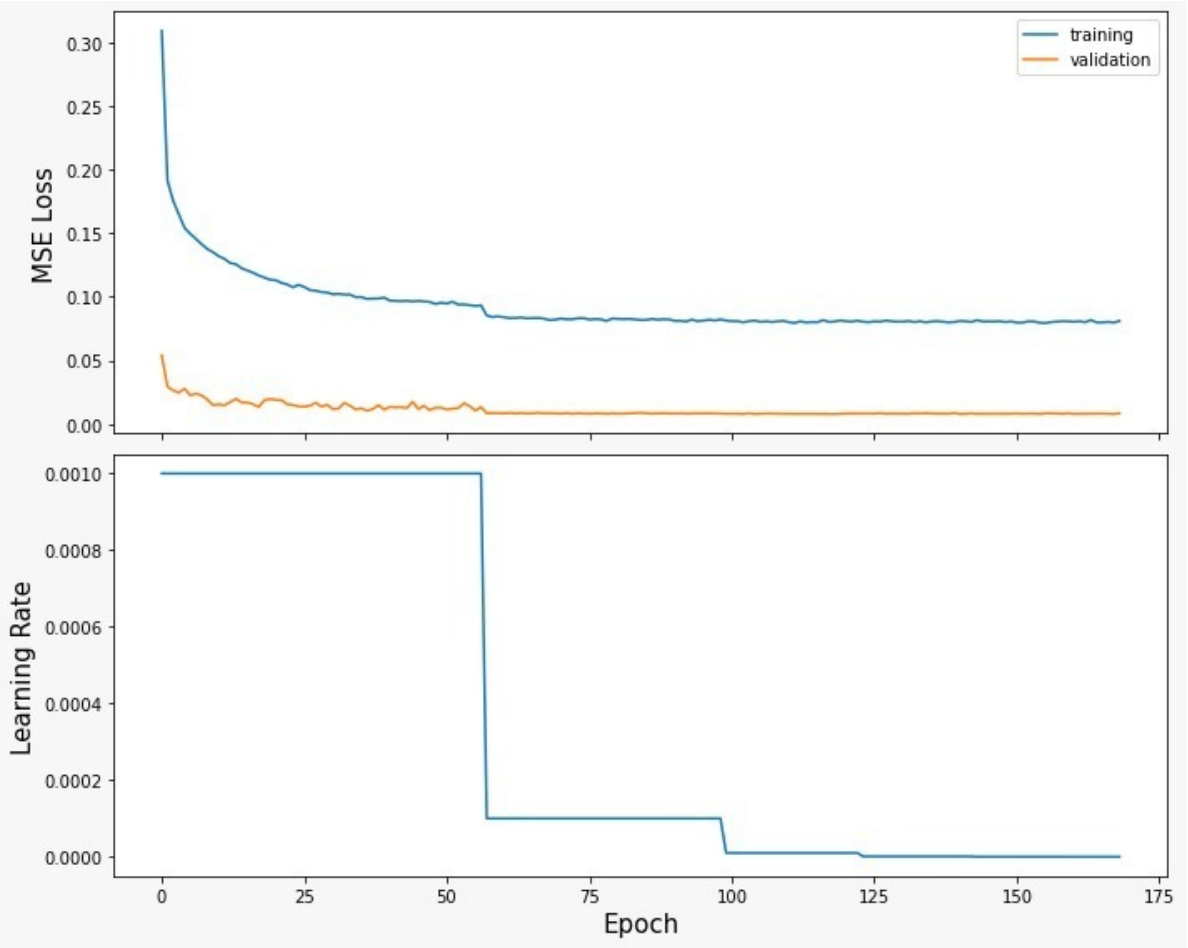
Neural network example – train network

```
history = model.fit(training_data,
                    epochs = 300,
                    batch_size = batch_size,
                    validation_data = validation_data,
                    callbacks = callbacks,
                    shuffle = True # shuffle the data in each batch
                    )
```

get histories of loss for training set (history.history['loss']) and validation set (history.history['val_loss']) as well as the learning rate (history.history['lr'])

make prediction by calling model.predict(X_test)

save the network by calling model.save('emulator')

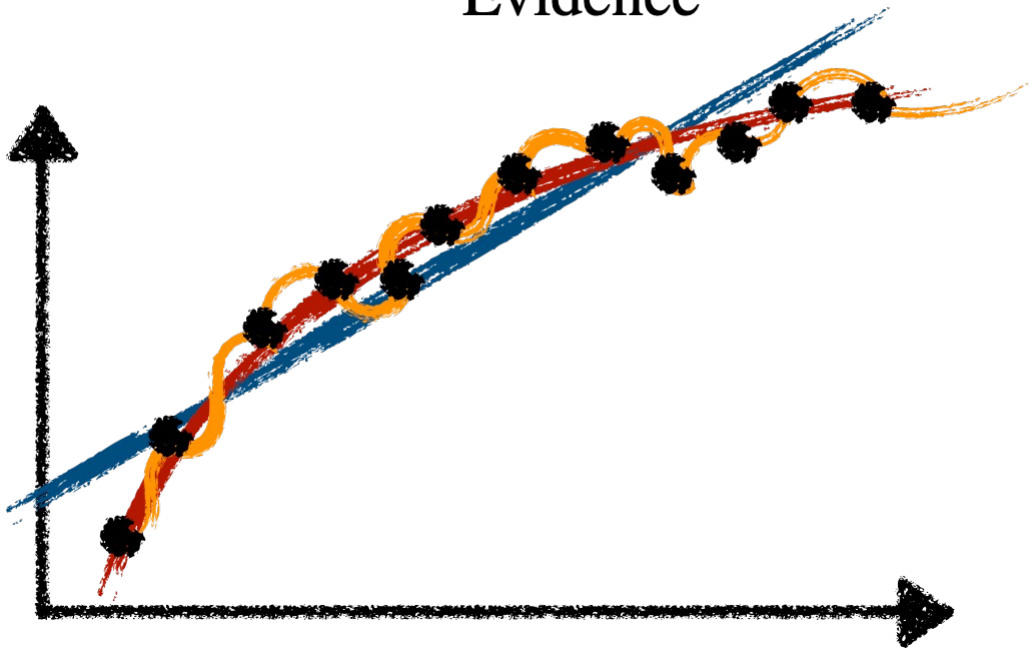


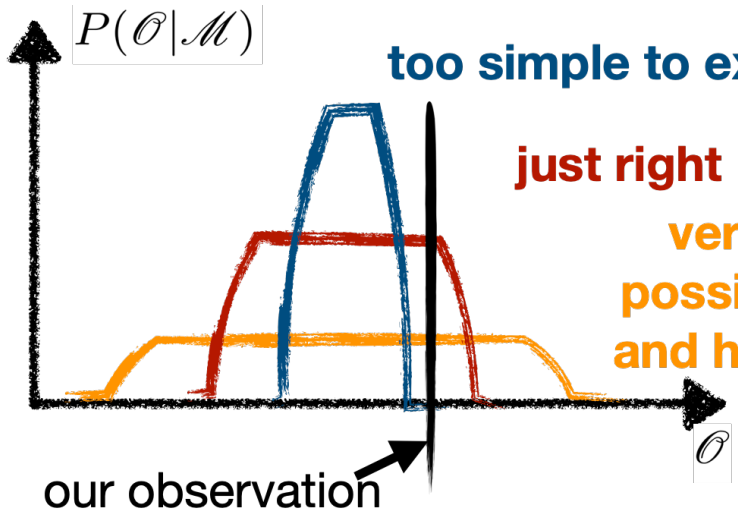
Bayesian inference

$$P(\theta|\mathscr{O},\mathscr{M}) = P(\theta|\mathscr{M})\frac{P(\mathscr{O}|\theta,\mathscr{M})}{P(\mathscr{O}|\mathscr{M})}$$

$$\text{Posterior} = \text{Prior}\frac{\text{Likelihood}}{\text{Evidence}}$$

$$P(\mathscr{O}|\mathscr{M}) = \oint_\theta \mathrm{d}\theta P(\mathscr{O}|\theta,\mathscr{M})P(\theta|\mathscr{M})$$





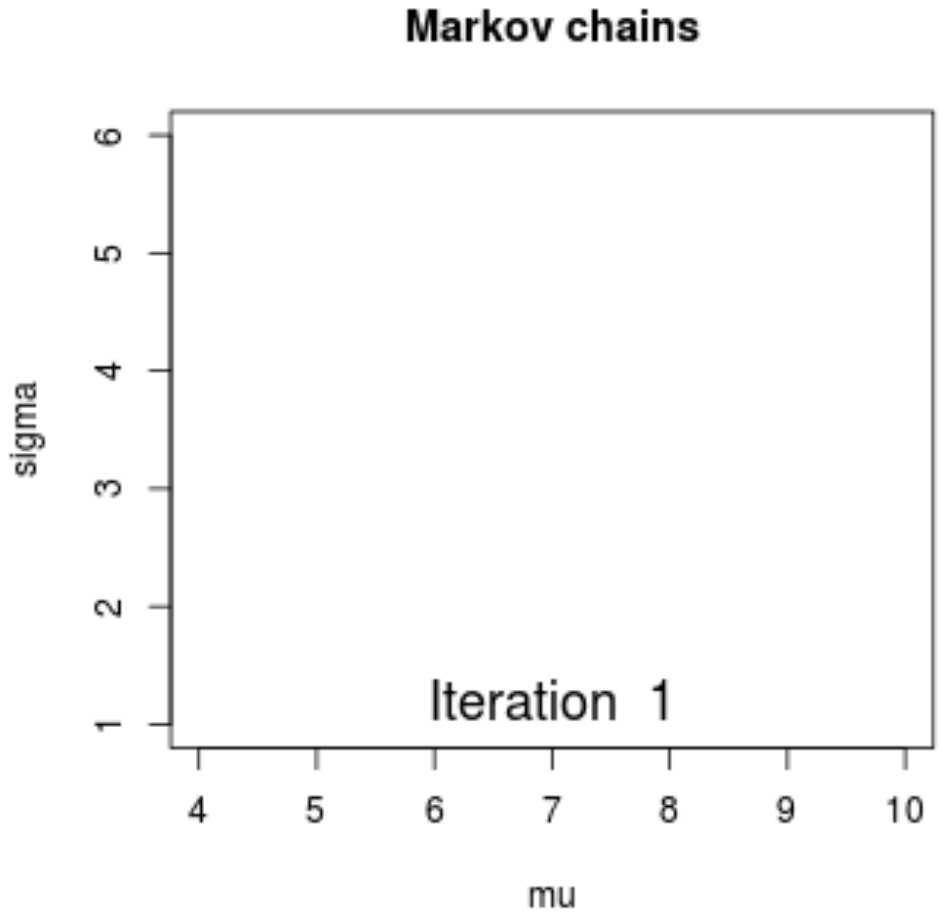
too simple to explain the observation

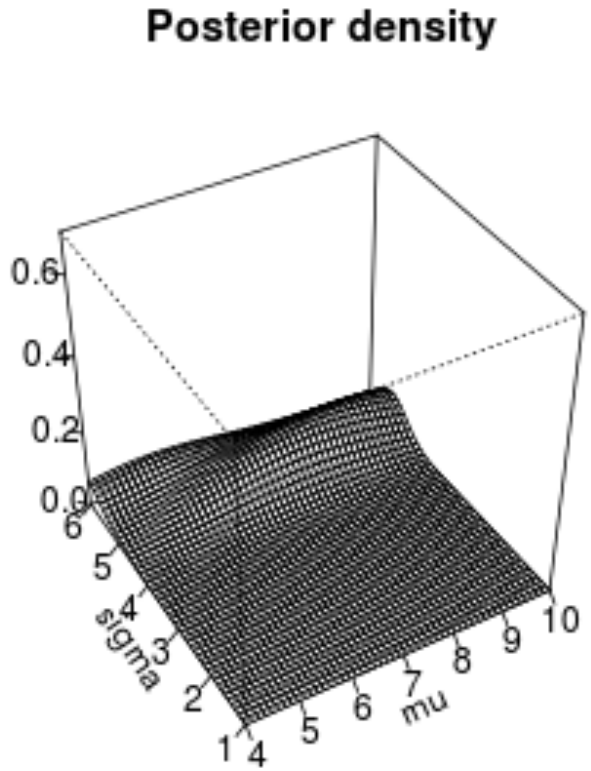
just right

very complex so that different possible dataset can be predicted and however, with a low probablity

Markov chain Monte Carlo (MCMC)

We will use EMCEE in this demonstration





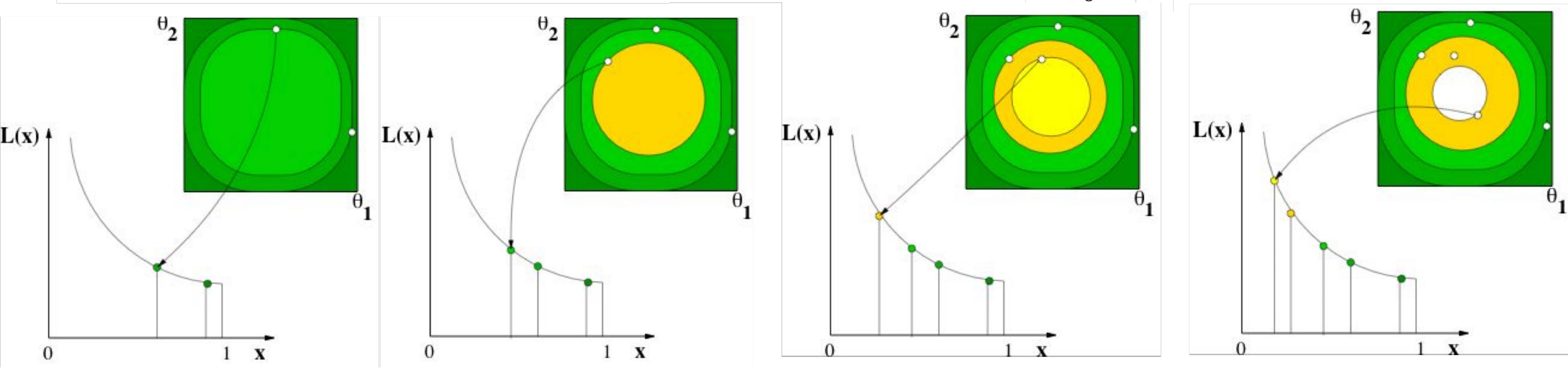
Nested samplers

We will use MultiNest in this demonstration

$$x(\lambda) = \int_{P(\mathscr{O}|\theta,\mathscr{M})>\lambda} \mathrm{d}\theta\, P(\theta|\mathscr{M})$$

Essentially, nested samplers do the sampling in the 1D prior space rather than the N-D parameter space

$$P(\mathscr{O}|\mathscr{M}) = \oint_\theta \mathrm{d}\theta P(\mathscr{O}|\theta,\mathscr{M})P(\theta|\mathscr{M}) = \int_0^1 \mathrm{d}P(\theta|\mathscr{M})P[\mathscr{O}|P(\theta|\mathscr{M})] = \int_0^1 \mathrm{d}x\, P(x)$$

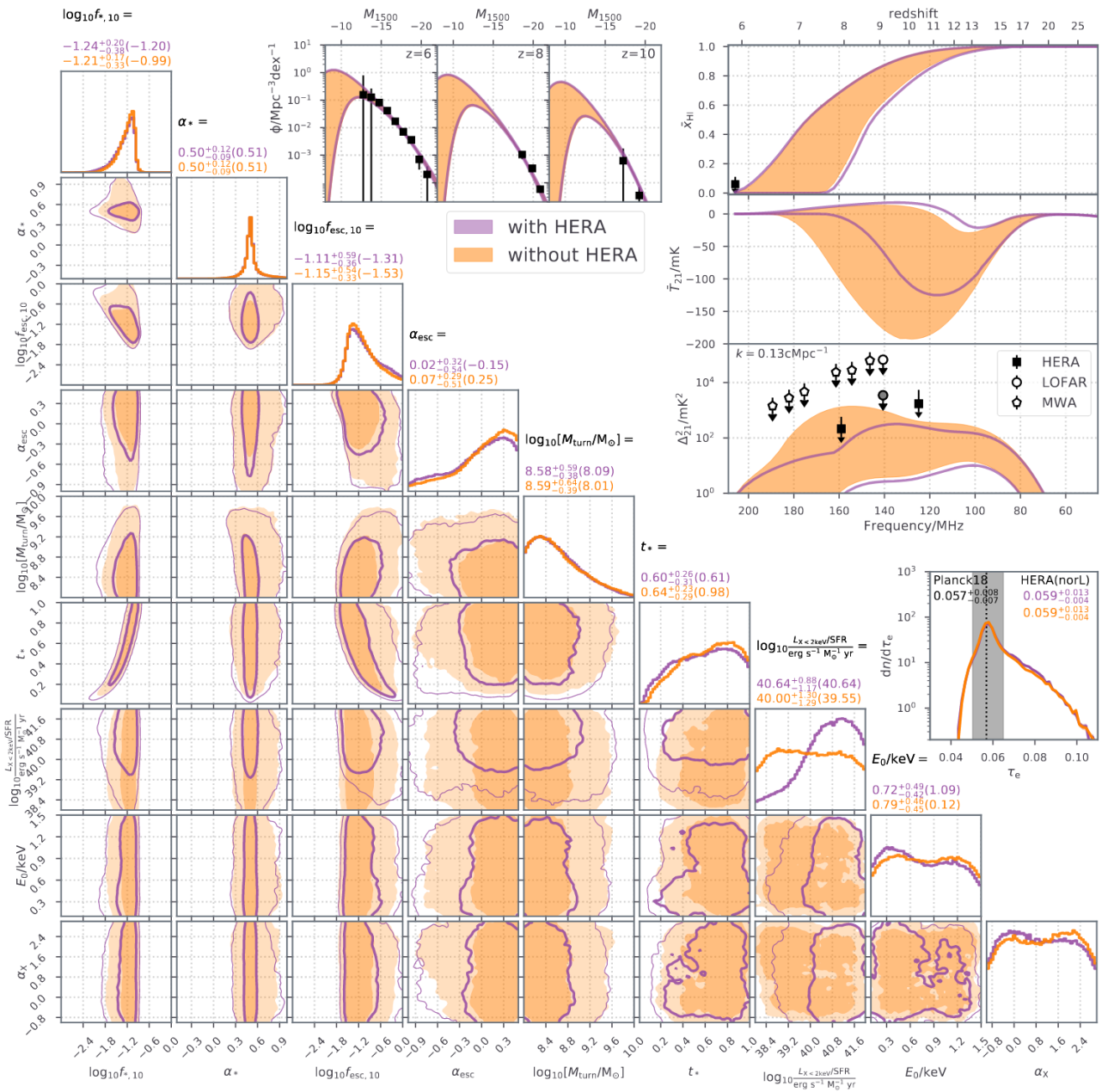


Task

Inspect and clean the data

Train dense network to emulate the model

Perform Bayesian inference using the emulator instead of the original model



HERA Collaboration+21

Package setup

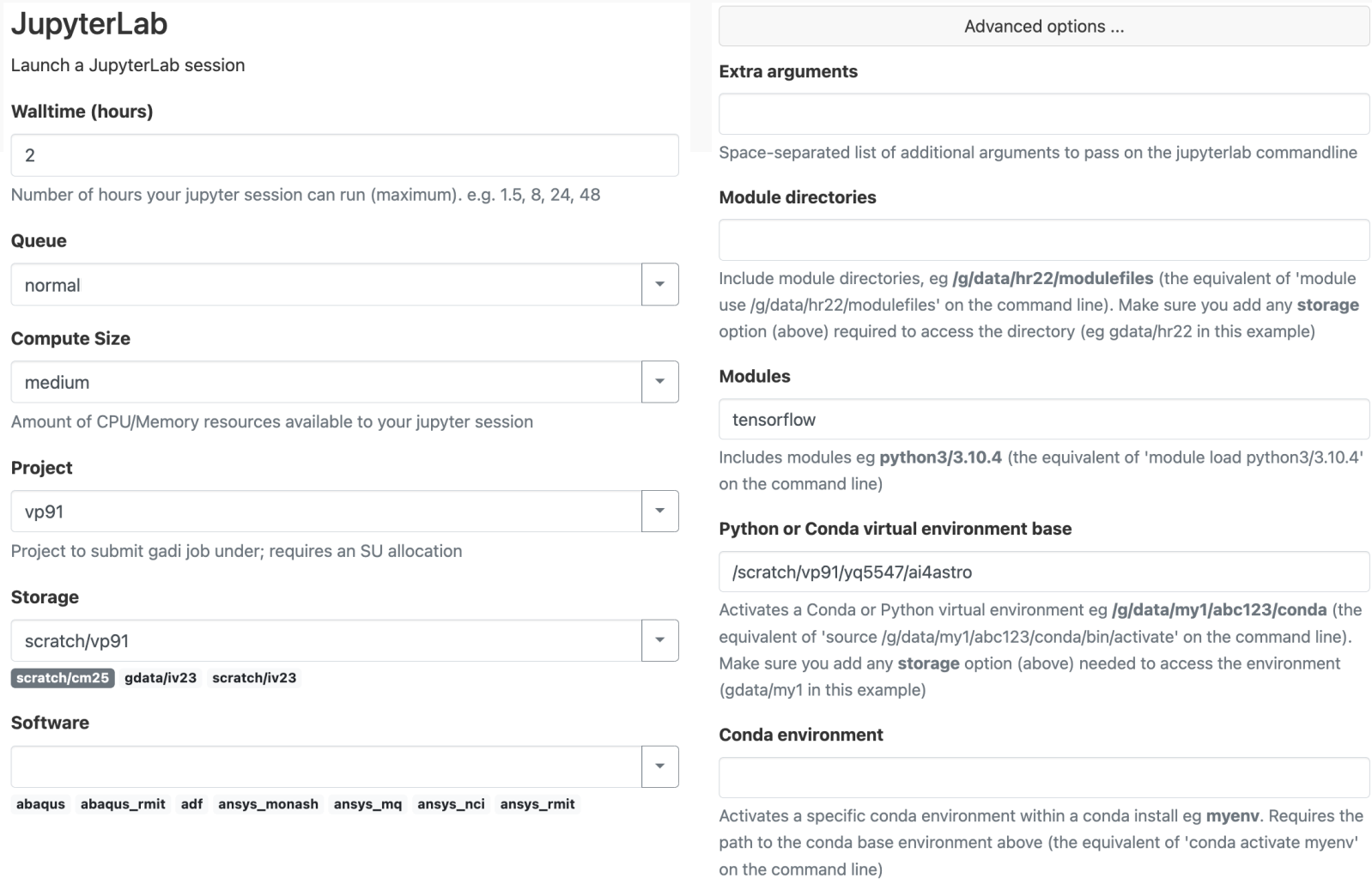
```
# create and go to your working directory
mkdir p scratch/vp91/$USER
cd /scratch/vp91/$USER

# needed modules
module load tensorflow #(intel-mkl/2021.4.0 python3/3.10.4 cuda/11.6.1 cudnn/8.2.2-cuda11.4 nccl/2.11.4
openmpi/4.1.2)

# create python environment
python3 -m venv ./ai4astro
pip install corner emcee ultranest jupyter jupyterlab scipy mpi4py
source ai4astro/bin/activate

# cp needed material
rsync  -avz /scratch/vp91/yq5547/notebooks ./
```

are@NCI

https://are.nci.org.au/

walltime: 2-3hours
queue: normal
(https://opus.nci.org.au/display/Help/Queue+Limits)
compute size: medium
project: vp91
storage: scratch/vp91

Advanced options:
Modules: tensorflow
Python or conda virtual environment base:
/scratch/vp91/*yq5567/ai4astro*



NCI AUSTRALIA

NCI Contacts
General enquiries: +61 2 6125 9800
Media enquiries: +61 2 6125 4389

Support: help@nci.org.au

Training request/inquiry: training.nci@anu.edu.au

Stay connected with us

All our training courses are at bit.ly/NCItrainingcalendar.
Follow us on LinkedIn at National Computational Infrastructure or bit.ly/NCILinkedIn.
Follow us on Twitter at NCINews or bit.ly/NCItwitter.

License CC BY SA