Machine Learning for Emulation and Uncertainty Quantification in a Land Surface Model

Katie Dagon

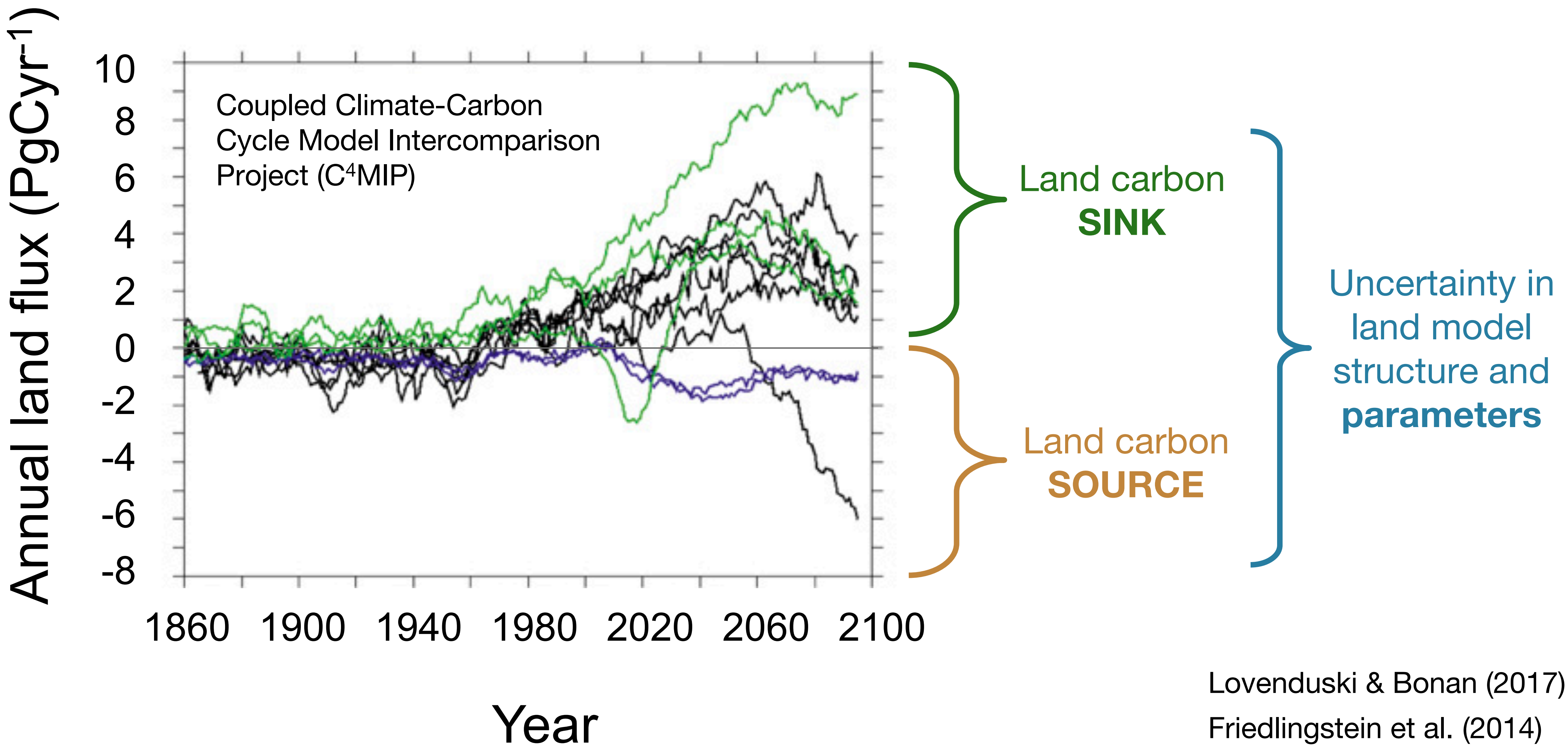
Climate and Global Dynamics Lab

National Center for Atmospheric Research

AI4ESS Summer School - June 24, 2020

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

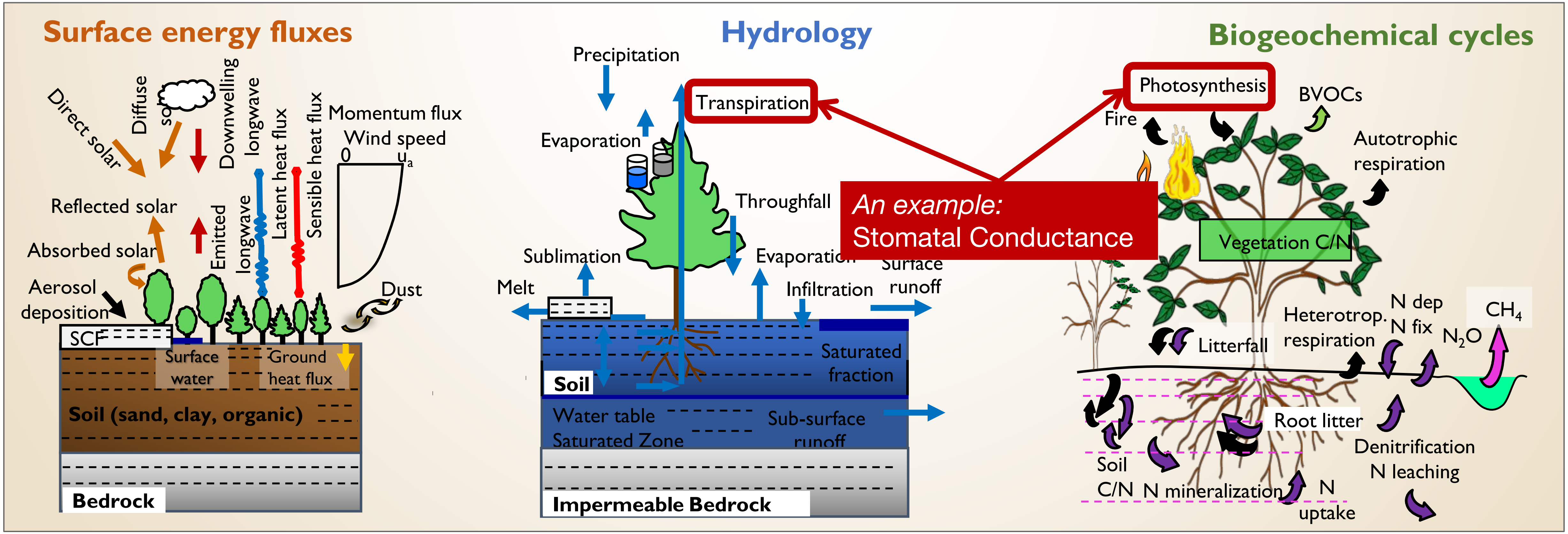
Land carbon cycle predictions are uncertain, but have significant consequences



Lovenduski & Bonan (2017)

Friedlingstein et al. (2014)

Uncertainty in Land Model Parameters



Schematic of NCAR's Community Land Model (CLM), version 5

Lawrence et al. (2019)

Example of Parameter Uncertainty: Stomatal Conductance

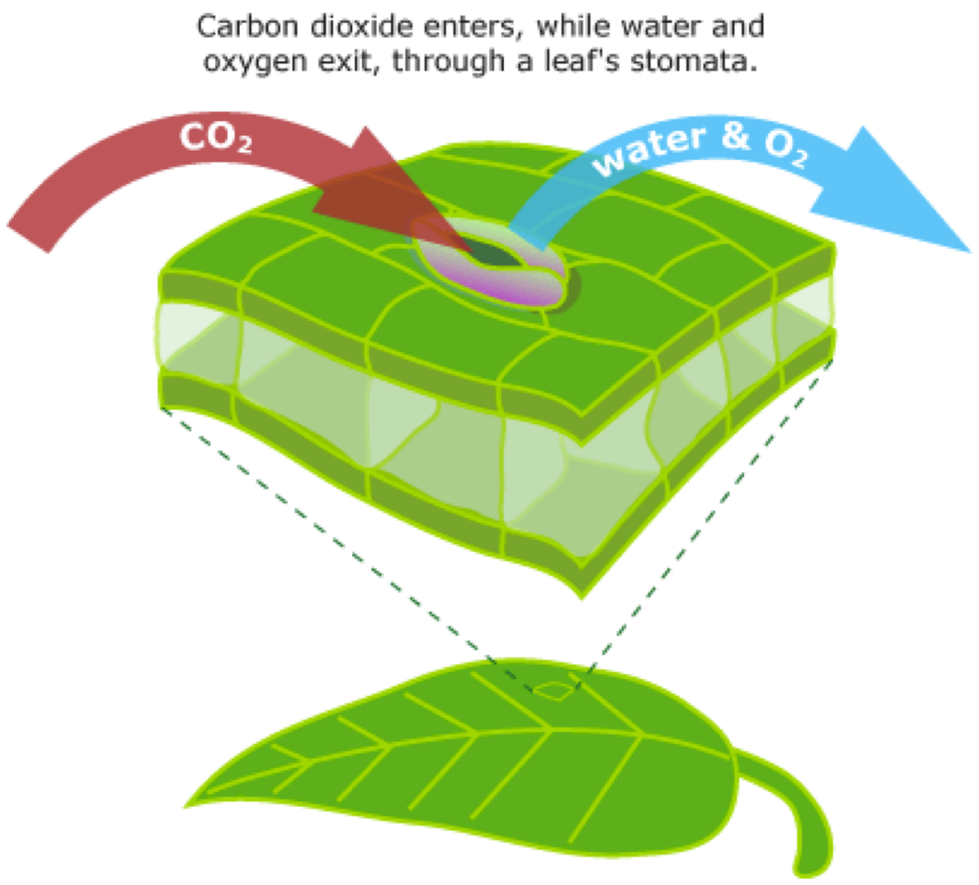
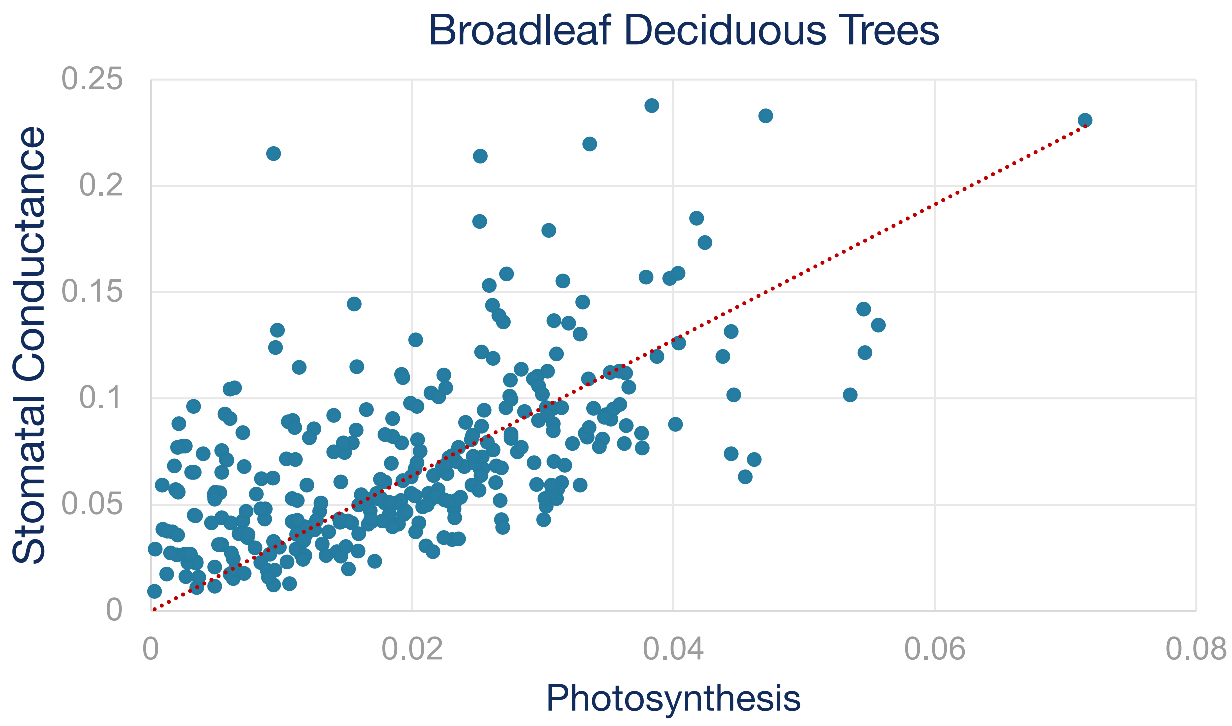


Image: evolution.berkeley.edu



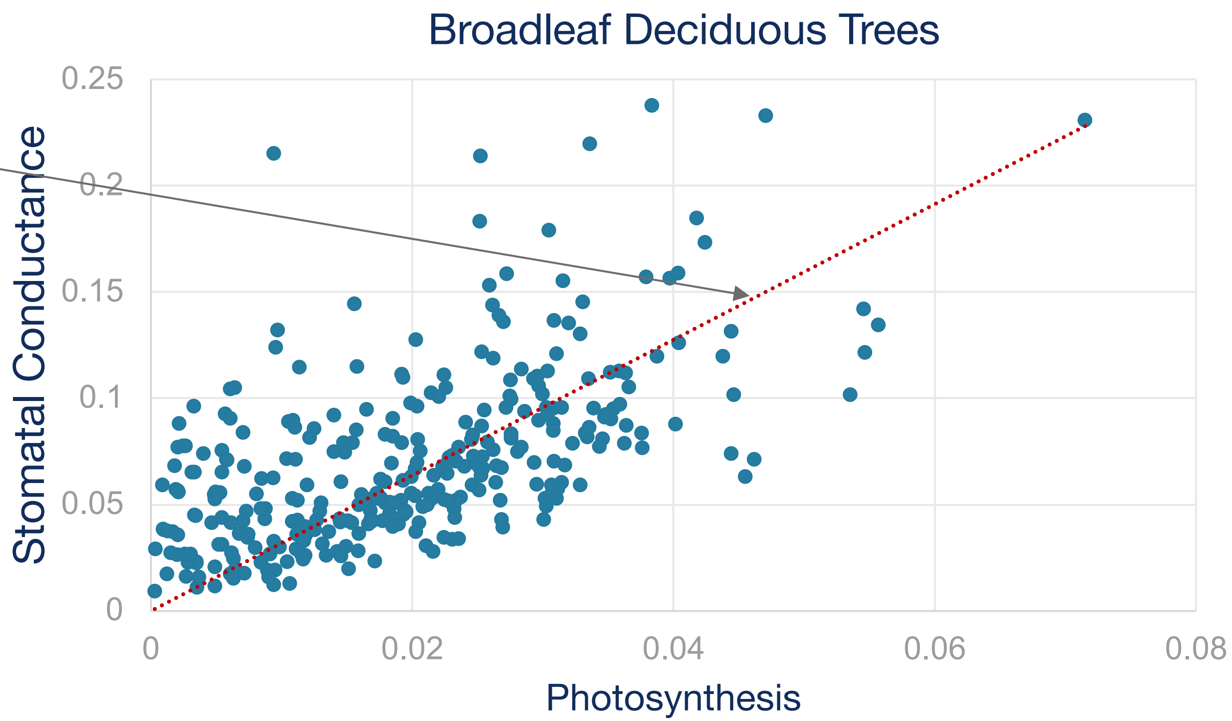
Data from Lin et al. (2015)

Example of Parameter Uncertainty: Stomatal Conductance

The slope of this line is an important model parameter, but its value is **uncertain**.

$$g_s = g_o + 1.6\left(1 + \frac{g_1}{\sqrt{D}}\right)\frac{A_n}{c_s/P_{atm}}$$

g_1 = slope parameter (μmol H_2O/μmol CO_2)

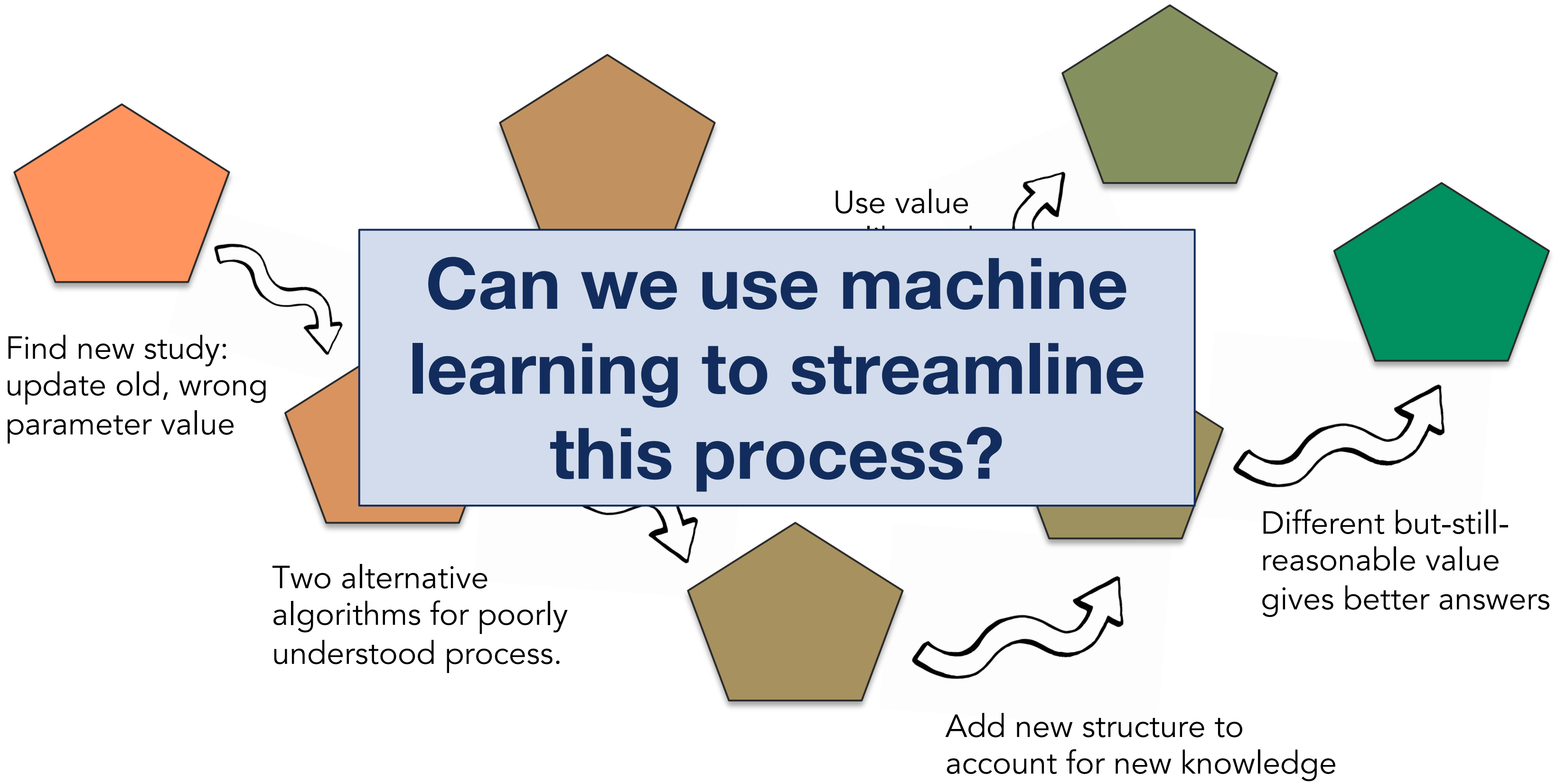


Medlyn et al. (2011)

Data from Lin et al. (2015)

Can we use machine learning to quantify parameter uncertainty?

Hand-tuning parameter values takes a long time (many model runs, trial and error).

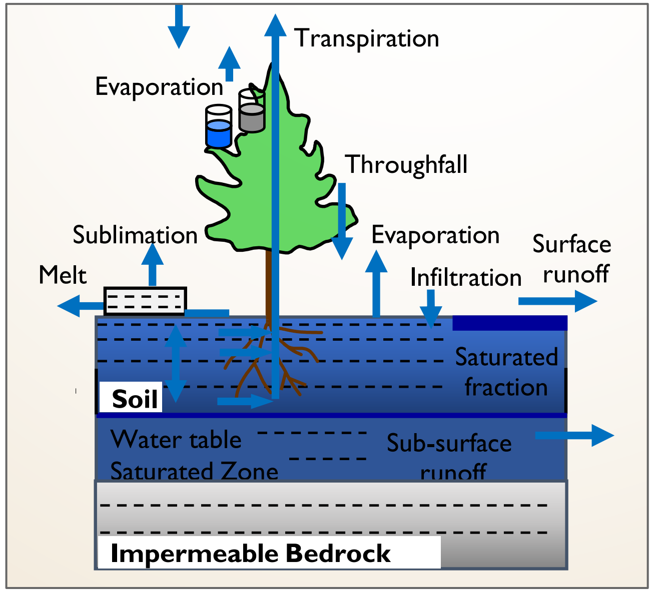


Can we use machine learning to streamline this process?

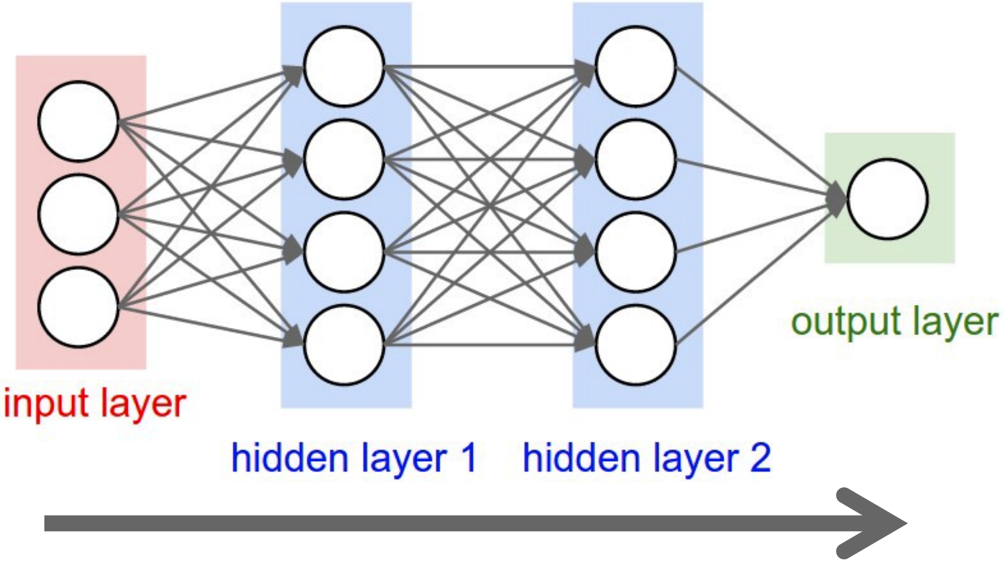
Figure from Rosie Fisher

Neural Networks as Land Model Emulators

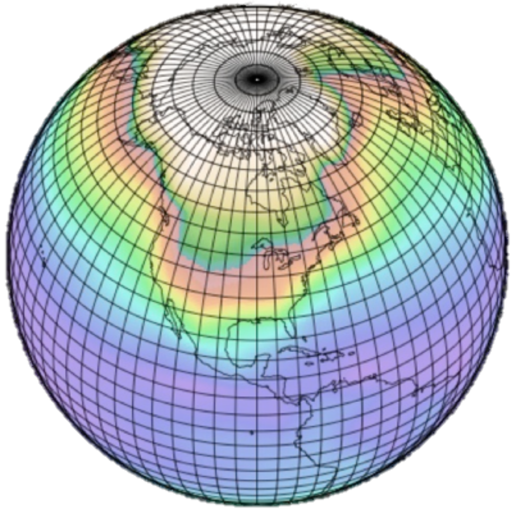
Input: parameter values



Neural network emulator



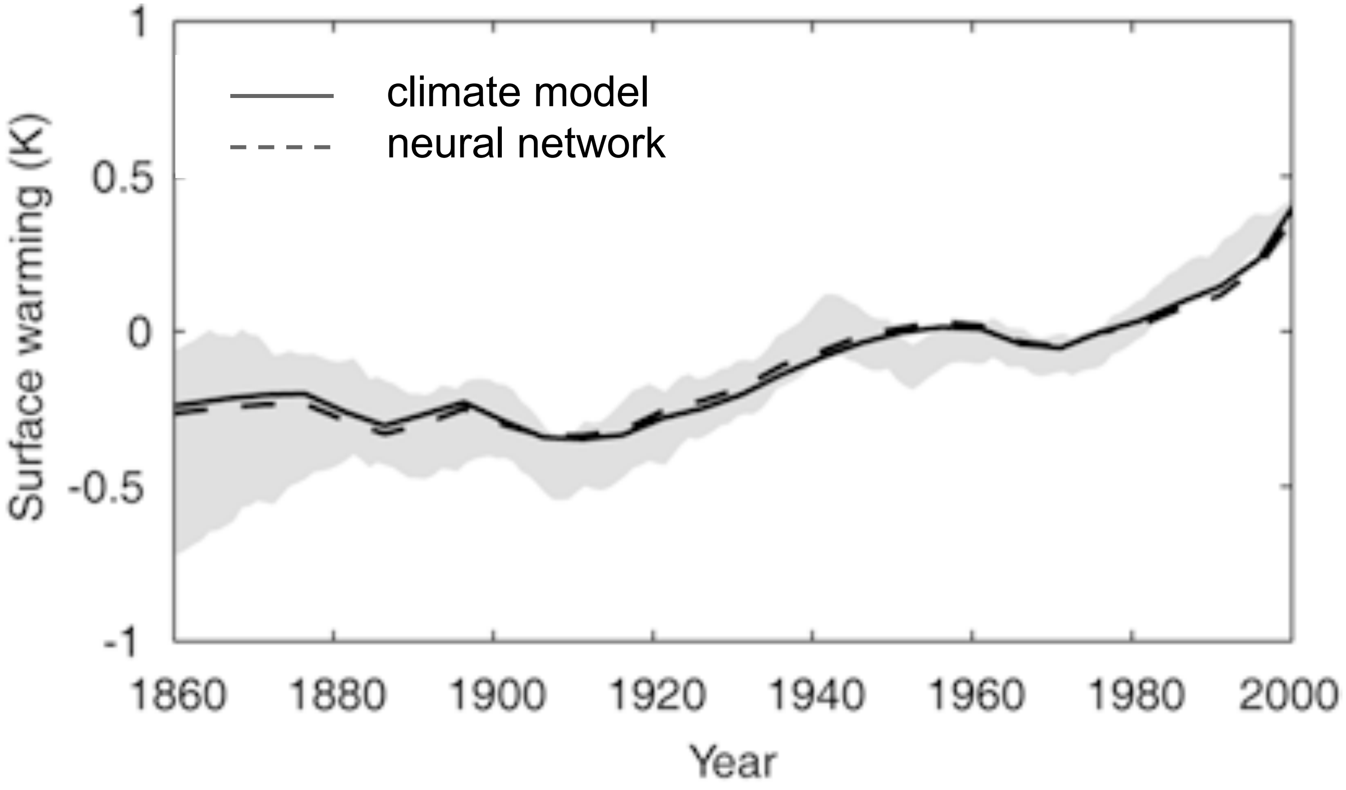
Output: land model predictions



Network image: http://cs231n.github.io/neural-networks-1/

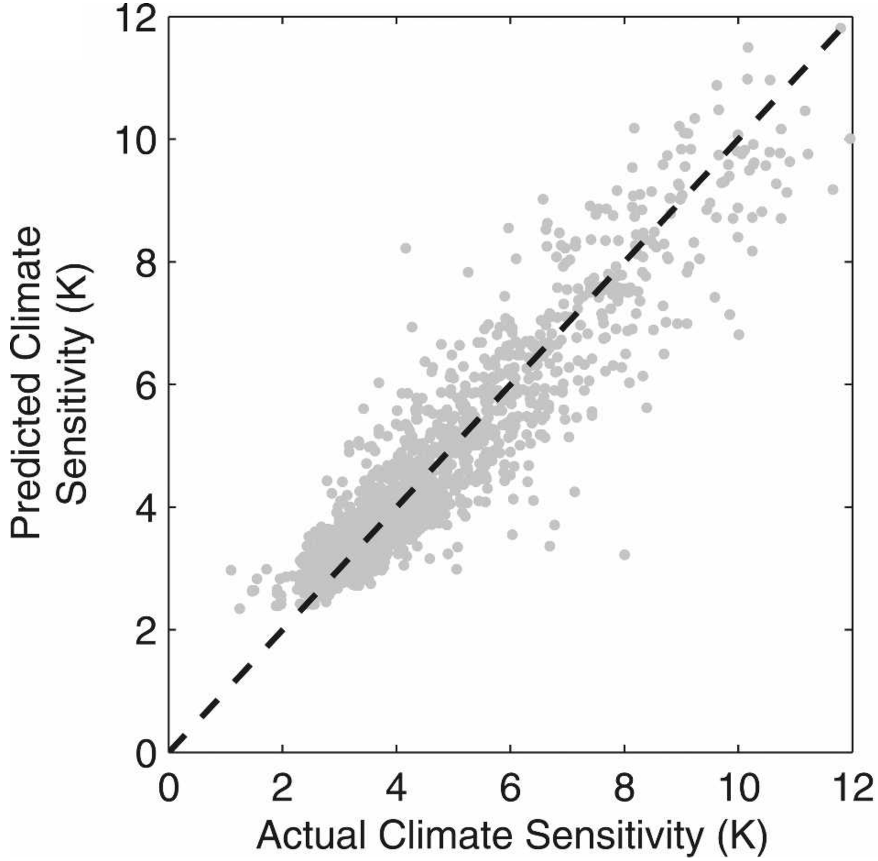
Emulation of Climate Models Using Neural Networks

Emulating changes in global average surface temperature



Knutti et al. (2003)

Emulating climate sensitivity



Sanderson et al. (2008)

Machine Learning Roadmap

1. *Train:* Build and train a series of **neural networks (NNs)** to predict land model output, given parameter values as input.

2. *Emulate:* Use trained NNs as **land model emulators** to make predictions with increased computational efficiency.

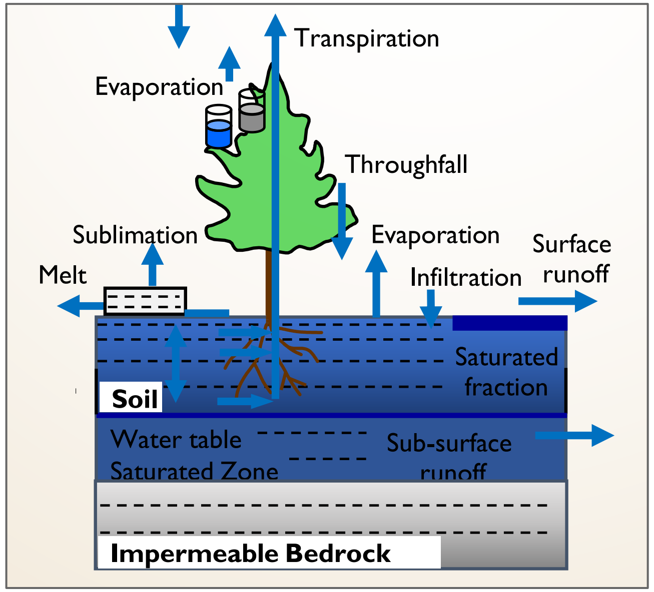
3. *Calibrate:* **Minimize error in predictions** relative to observations; generate optimal parameter values and distributions.

4. *Test:* Use optimal parameter values to **investigate changes in model predictive skill**.

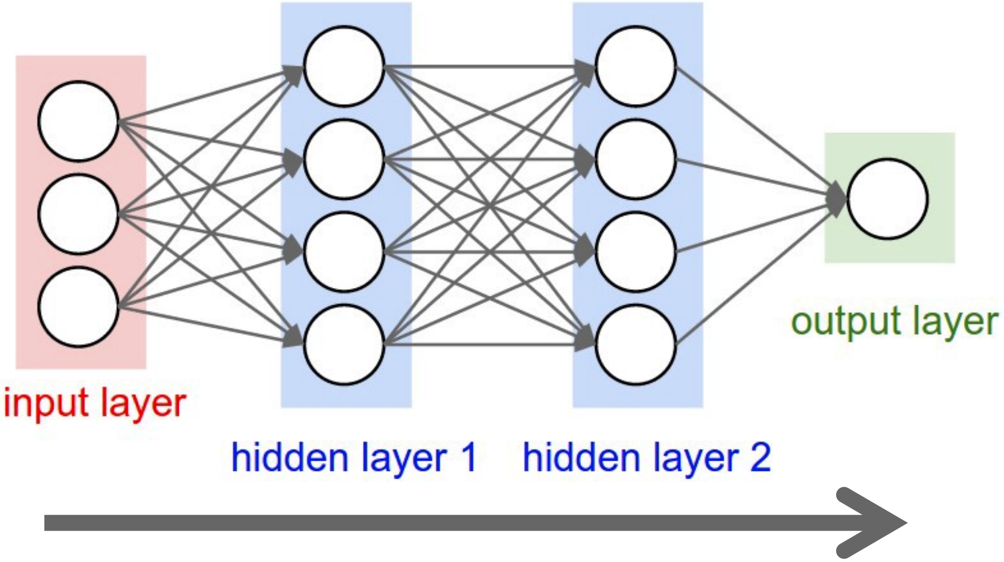
Neural Networks as Land Model Emulators

Step 1: Train

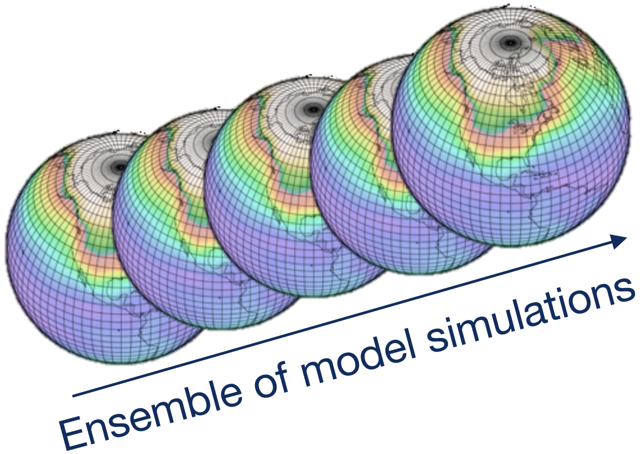
Input: parameter values



Neural network emulator



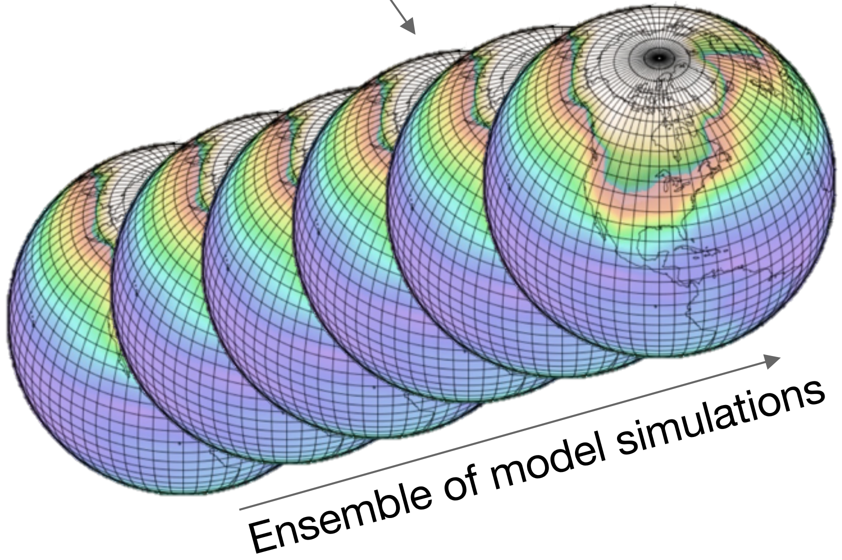
Output: land model perturbed parameter ensemble



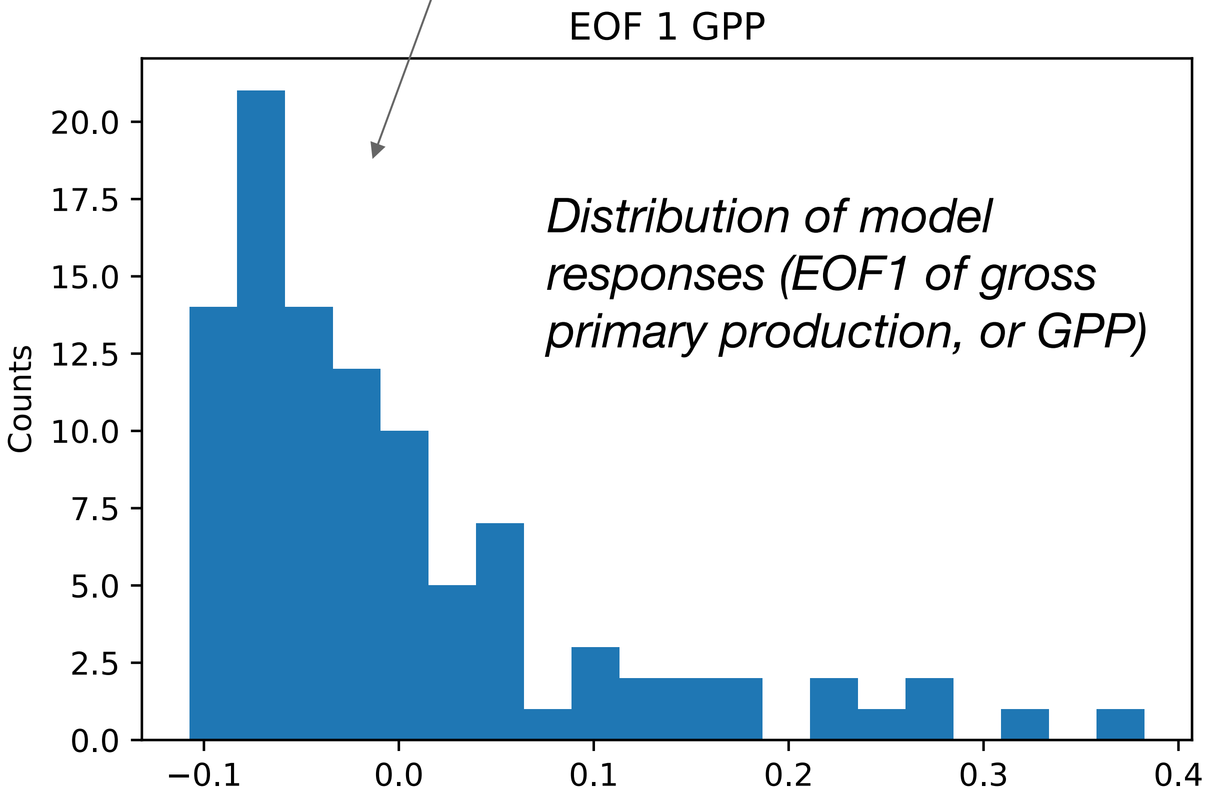
A machine learning algorithm is trained to predict land model output, given parameter values as input.

Generating the Training Data

Land model* perturbed physics ensemble (PPE) using 100 parameter combinations generated with Latin Hypercube sampling



Predict modes of variability of carbon and water fluxes



*Offline land-only simulations forced by atmospheric reanalysis data

Adventures in Pre-Processing!

Obs ~/CLM5hydro_LHC/obs

① -obs_file.ncl
fluxnet for 2000-2004
machines OR Glade (2 options)
MR_$VAR_0.9x1.25_2000-2004.nc
MR_$VAR 0.9x1.25.nc (all yrs for sd)

② regrid_obs.ncl
NCL only ?)
R_4x5_2000-2004.nc
R_4x5.nc
cl
from CLM PPE output ensemble mean
ensemble mean
well
$VAR_4x5_anom_forSVD.nc
$VAR_4x5_anom_forSVD_allyrs.nc

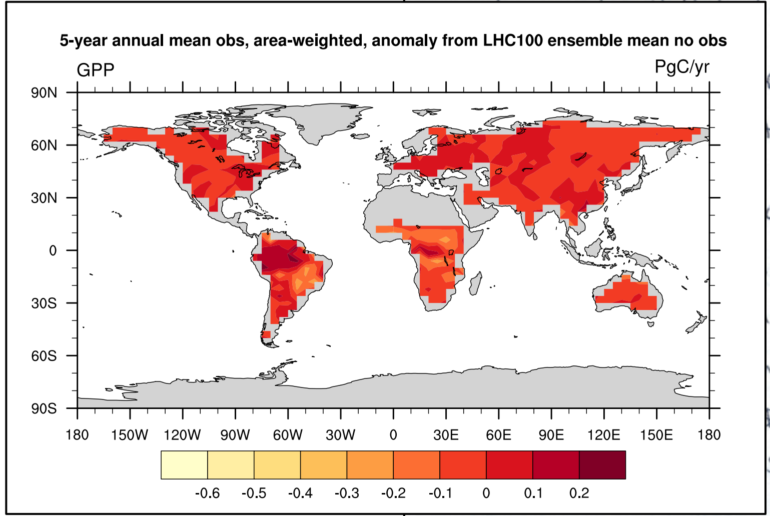
CLM PPE output ~/CLM5hydro_LHC/outputdata

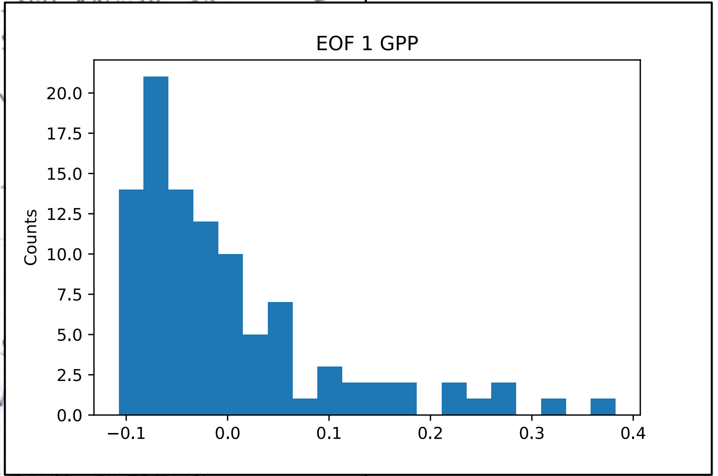
① process_outputdata_SVD.ncl
then save out datamask
requires datamask from regridded obs fi
area-weighted anomalies from ensemble m
outputdata_$VAR_forSVD_1

② SVD.py [~/CLM5hydro_LHC]
outputdata_$VAR_SVD_$Nmodes.npy
Obs_$VAR
modeldefault_

③ SVD_obs.py
obs_$VAR

④ process_model_SVD.ncl
default params and te
for projection into S
CLM_default_
test_paramset_$VAR_forSVD.nc





Adventures in Pre-Processing!

```python
# Read in input array
inputdata = np.load(file="lhc_100.npy", allow_pickle=True)

# Read in output array
outputdata = np.load("outputdata/outputdata_GPP_SVD_3modes.npy")

# Multi-dimension
nmodes = outputdata.shape[1]

# Separate training/test/val data: 60/20/20 split
x_train = inputdata[0:60]
x_test = inputdata[60:80]
x_val = inputdata[80:]
y_train = outputdata[0:60]
y_test = outputdata[60:80]
y_val = outputdata[80:]

# Max # of nodes
maxnode = 15

# Min # of nodes
minnode = 5

# Loop over # of nodes
metrics=[]
eps = []

# First layer
for i in range(minnode,maxnode+1):
    # Second layer
    for j in range(minnode,maxnode+1):

        print("Node configuration:")
        print(i,j)

        # Random seed for reproducibility
        np.random.seed(9)

        # Create 2-layer simple model
        model = Sequential()
        model.add(Dense(i, input_dim=inputdata.shape[1], activation='relu',
            kernel_regularizer=l2(.001)))
        model.add(Dense(j, activation='tanh', kernel_regularizer=l2(.001)))
        model.add(Dense(nmodes))
```

Keras

TensorFlow

NumPy

SciPy

jupyter

MACHINE LEARNING MASTERY

When to use a Sequential model

A `Sequential` model is appropriate for **a plain stack of layers** where each layer has **exactly one input tensor and one output tensor**.

Schematically, the following `Sequential` model:

```python
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)
```

https://keras.io/guides/sequential_model/

In this example, we will evaluate learning rates on a logarithmic scale from 1E-0 (1.0) to 1E-7 and create line plots for each learning rate by calling the *fit_model()* function.

```python
# create learning curves for different learning rates
learning_rates = [1E-0, 1E-1, 1E-2, 1E-3, 1E-4, 1E-5, 1E-6, 1E-7]
for i in range(len(learning_rates)):
    # determine the plot number
    plot_no = 420 + (i+1)
    pyplot.subplot(plot_no)
    # fit model and plot learning curves for a learning rate
    fit_model(trainX, trainy, testX, testy, learning_rates[i])
# show learning curves
pyplot.show()
```

https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

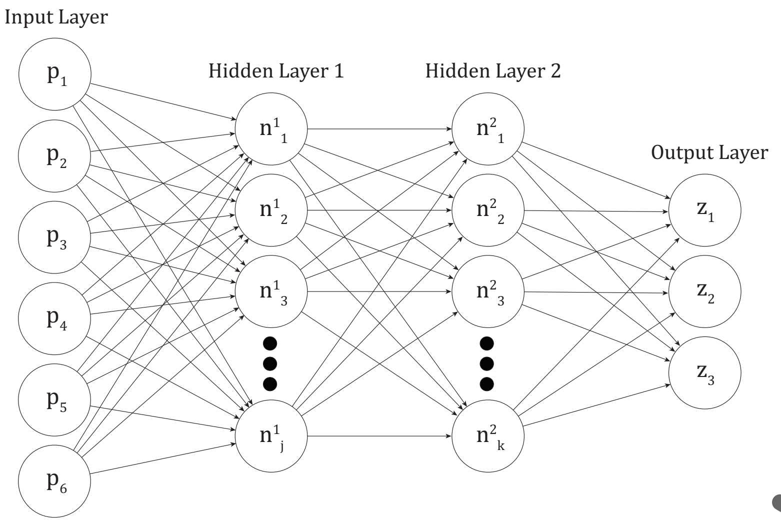
Neural Networks as Land Model Emulators

Step 1: Train

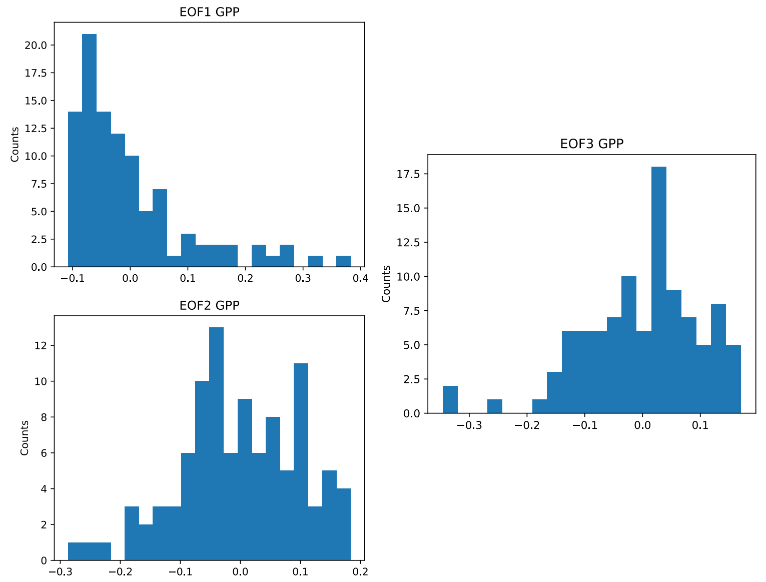
Input: parameter values

	P1	P2	P3	P4	P5	P6
S1	x1,1	x1,2	x1,3	x1,4	x1,5	x1,6
S2	x2,1	x2,2	x2,3	x2,4	x2,5	x2,6
S3	x3,1	x3,2	x3,3	x3,4	x3,5	x3,6
…	…	…	…	…	…	…
S100	x100,1	x100,2	x100,3	x100,4	x100,5	x100,6

2-layer feed-forward artificial neural network (ANN)



Output: land model perturbed parameter ensemble



Train to predict spatial variability (first 3 EOFs) of gross primary production (GPP).

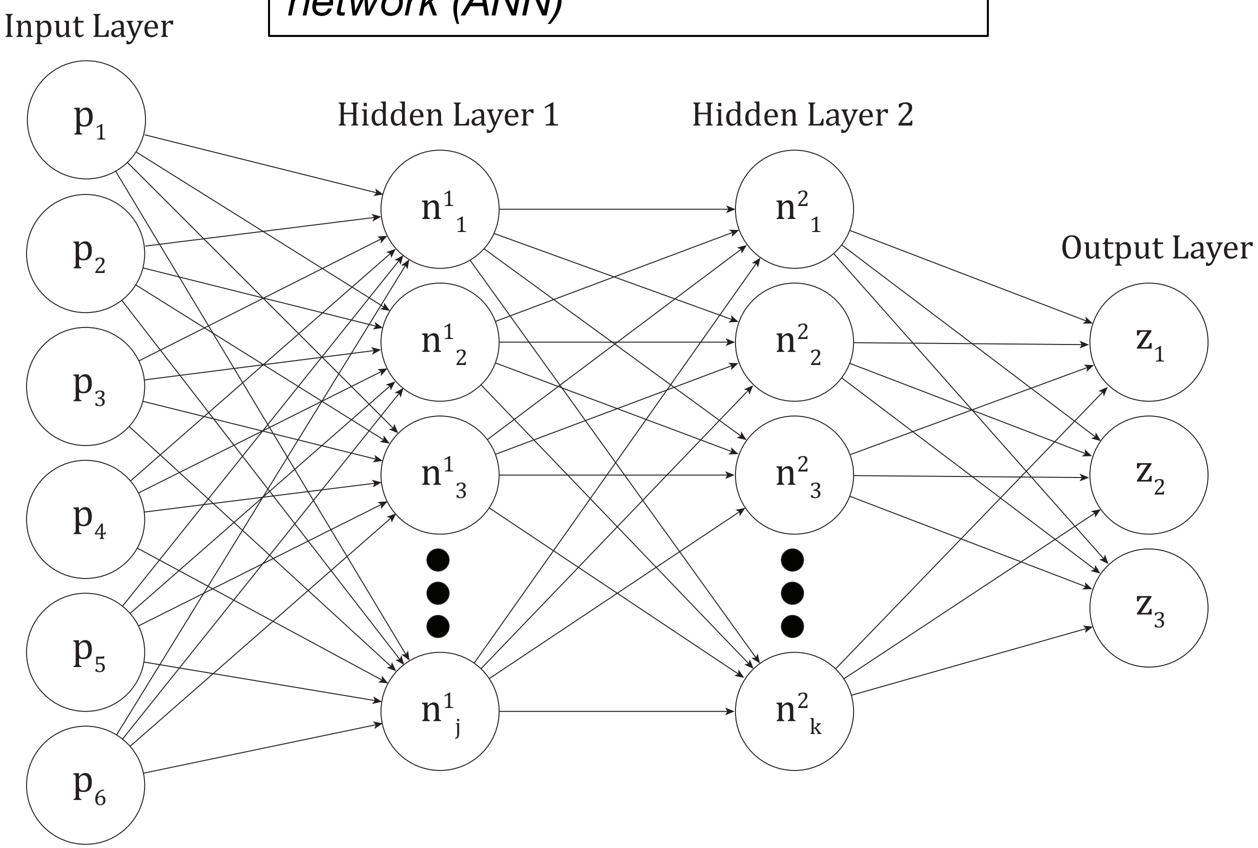
Separate emulator built for first 3 EOFs of latent heat flux (LHF).

Hyperparameter Optimization

Primary ANN configuration options:

- Number of hidden layers
- Number of nodes/neurons in each layer
- Activations between layers (e.g., linear, nonlinear)
- Optimization algorithm
- Learning rate
- Batch size
- Number of training epochs

2-layer feed-forward artificial neural network (ANN)



Hyperparameter Optimization

Primary ANN configuration options:

- Number of hidden layers

2 layers improved performance over a single hidden layer.

- Number of nodes/neurons in each layer

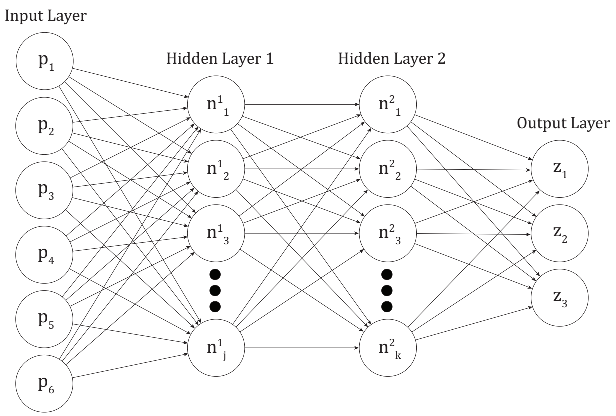
Iteratively test between **5-15 nodes in each layer**, then select best performing configurations based on error metric and predictive skill.

- Activations between layers (e.g., linear, nonlinear)

ReLU improved over linear for first activation; **tanh** improved over sigmoid for second activation.

- Optimization algorithm

RMSprop improved predictive skill over SGD.



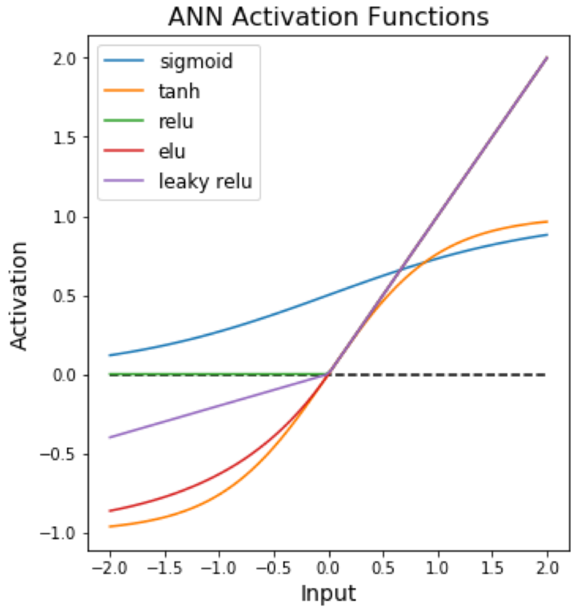


Figure from DJ Gagne

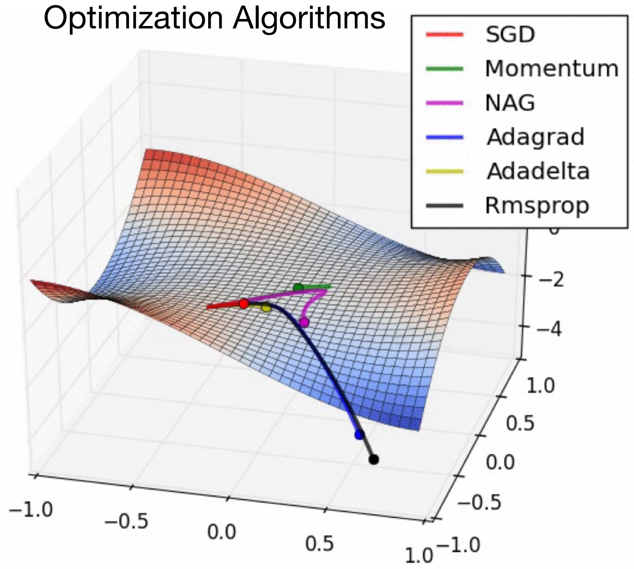


Figure from https://imgur.com/a/Hqolp#NKsFHJb

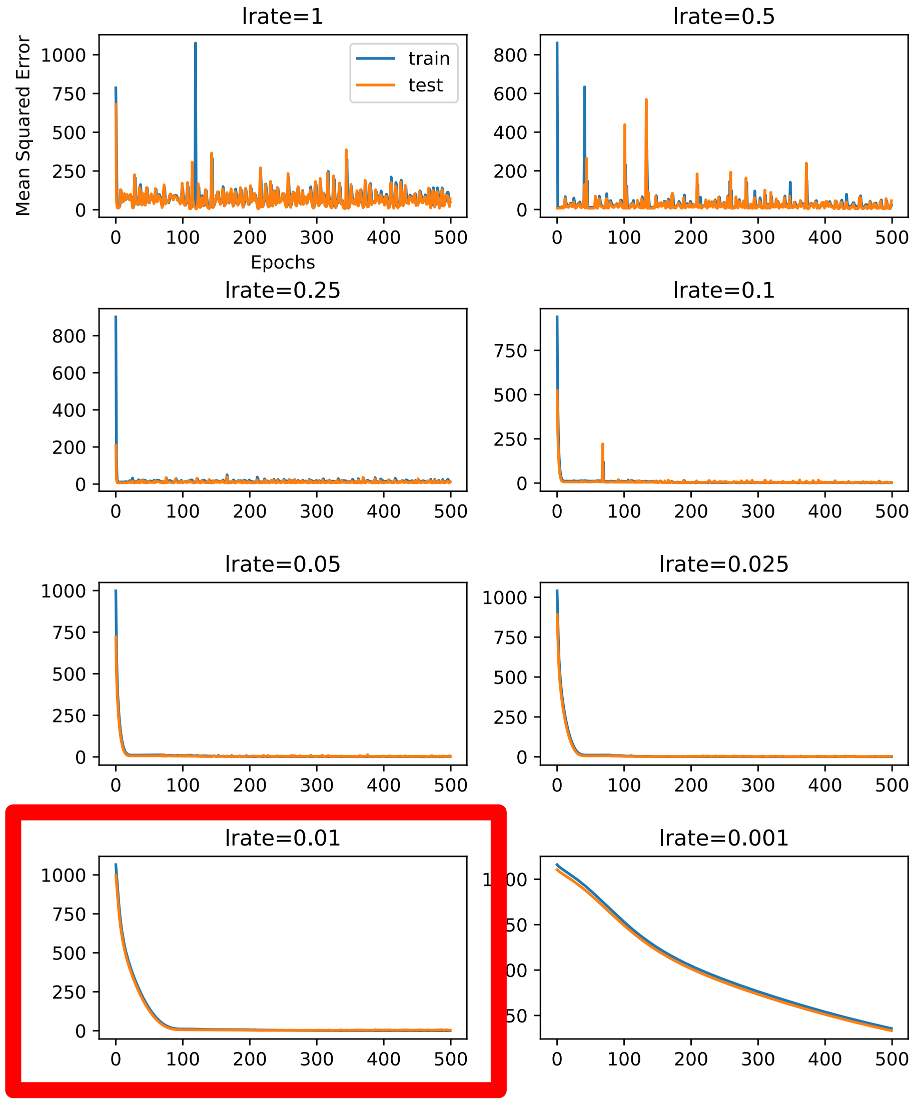
Hyperparameter Optimization

- **Learning rate**: how much does the model change in response to error?

Comparing learning rates and plotting learning curves over the training process.

Learning rate of 0.01 provided a good compromise on convergence and accuracy.

Metric = mean squared error between emulator predictions and actual model output

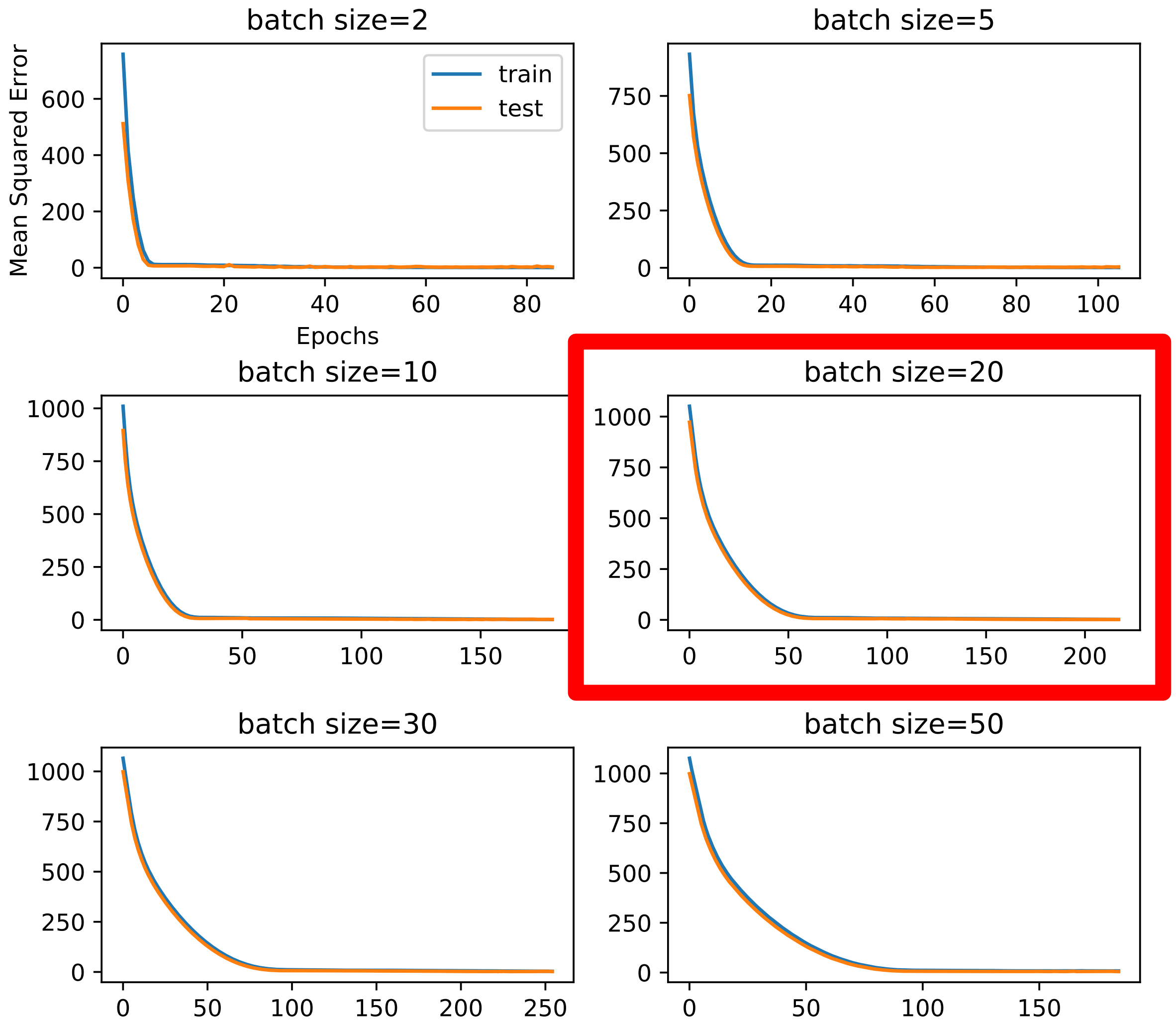


Hyperparameter Optimization

- **Batch size**: number of subsamples used to calculate the error gradient

Comparing batch sizes and plotting learning curves over the training process.

Batch size of 20 provided a good compromise on convergence and accuracy.



Metric = mean squared error between emulator predictions and actual model output

Hyperparameter Optimization

- **Batch size**: number of subsamples used to calculate the error gradient

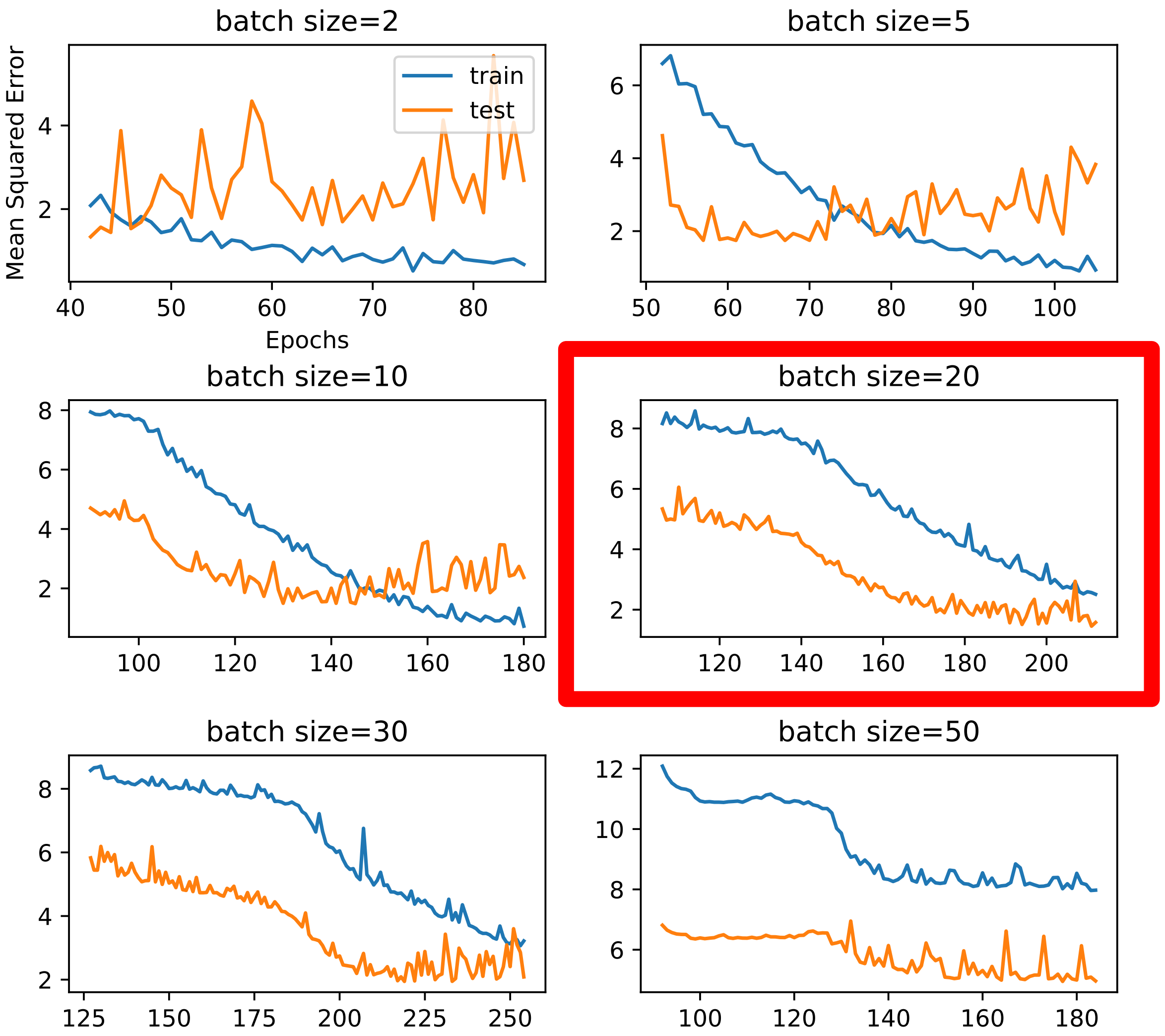
Comparing batch sizes and plotting learning curves over the training process.

Batch size of 20 provided a good compromise on convergence and accuracy.

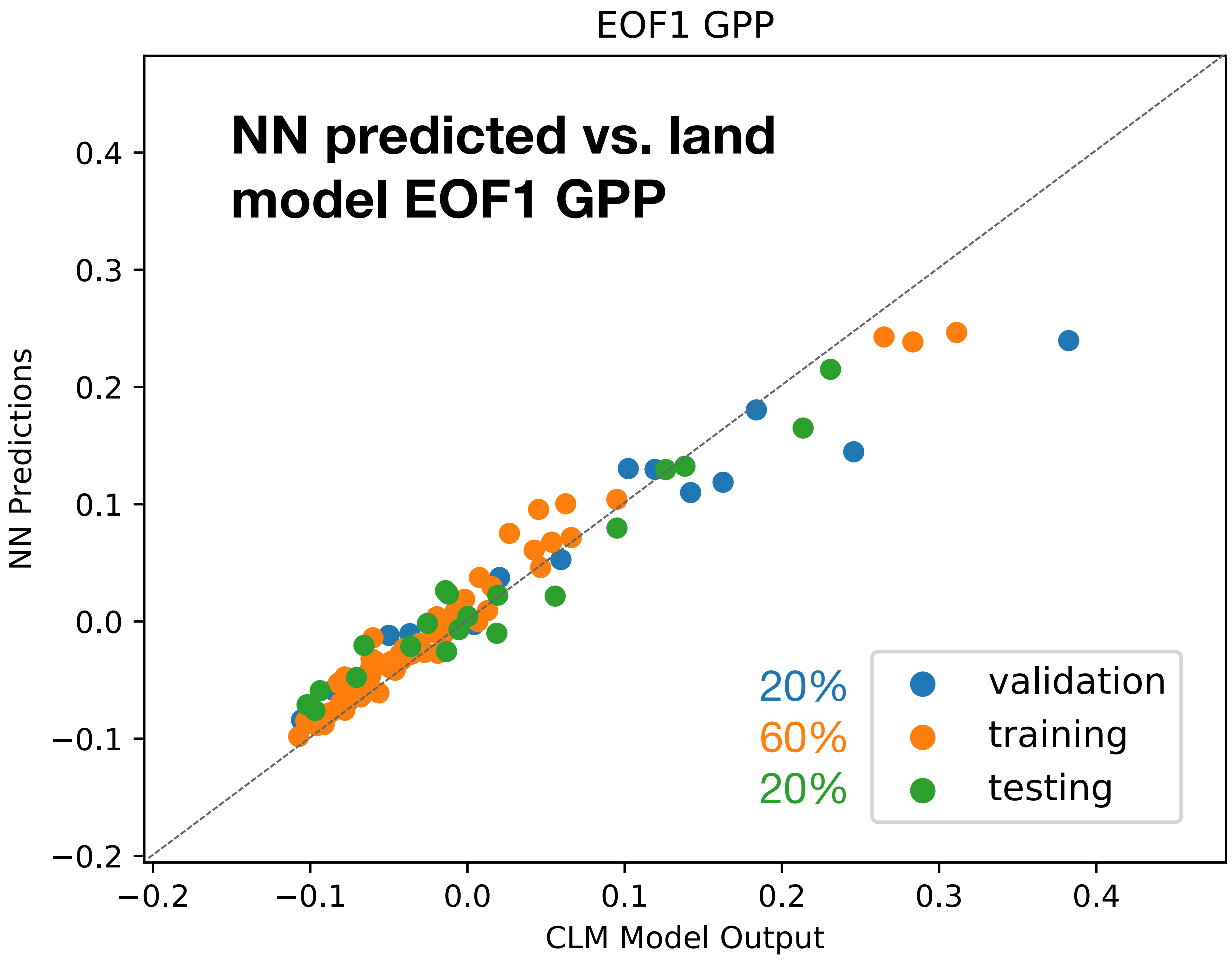
- **Number of training epochs**: how long to run the training process

Early Stopping used to determine number of epochs.

Metric = mean squared error between emulator predictions and actual model output

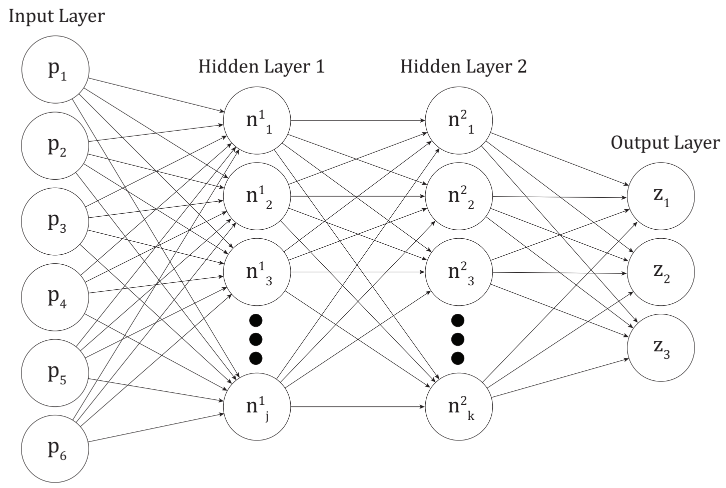


Assessing Emulator Performance



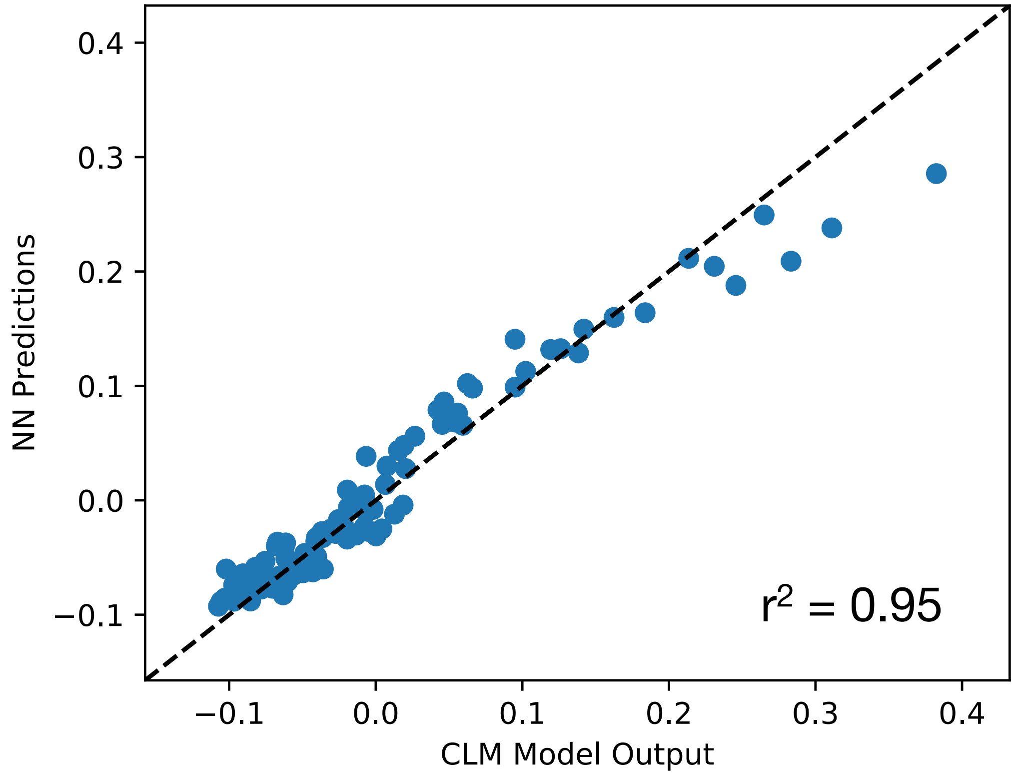
<u>Testing different ANN architectures:</u>

1. Iteratively test ANN hyperparameters, **selecting best performing configurations**.
2. For the best configurations, **randomly resample training data** 100 times to test variability of performance.
3. Select network with **highest skill AND lowest variability** as final configuration.



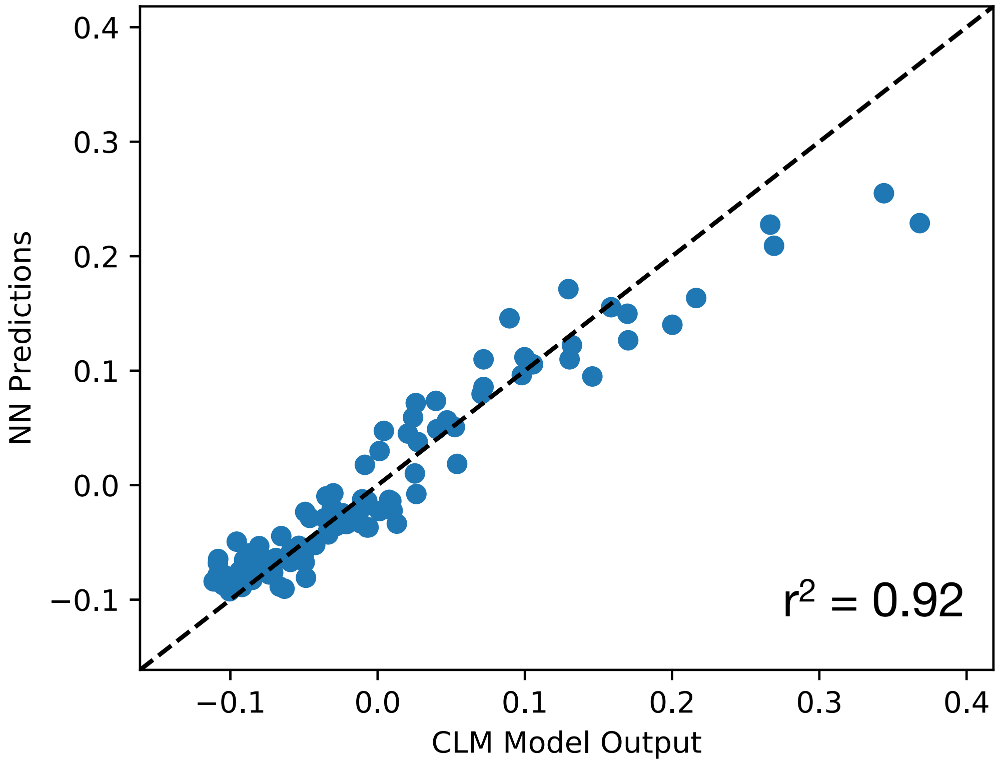
Out-of-Sample Prediction

Original ensemble (EOF1 GPP)



"Best" emulator trained on random parameter values and model output.

Second ensemble (EOF1 GPP), different random parameter values



Same emulator; **different** random parameter values and resulting model output. **Predictive skill is comparable.**

Dagon et al., *in review*

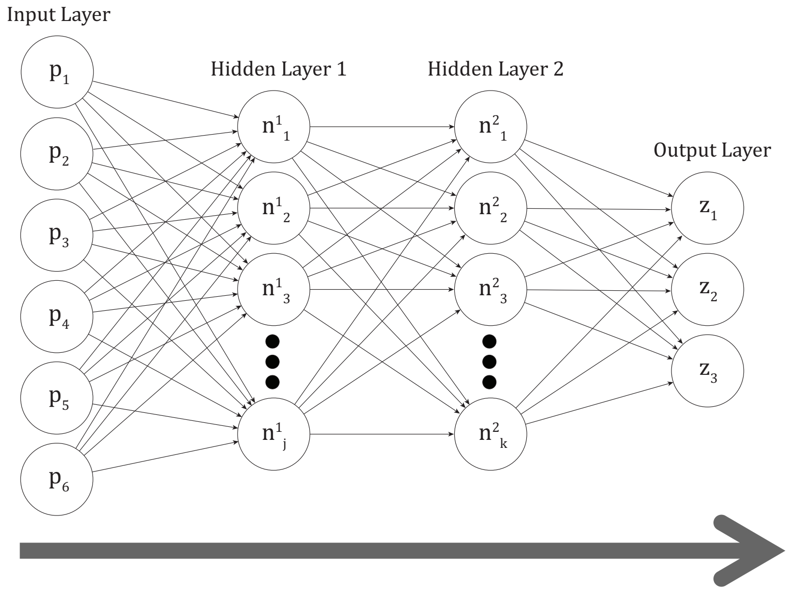
Neural Networks as Land Model Emulators

Step 2: Emulate

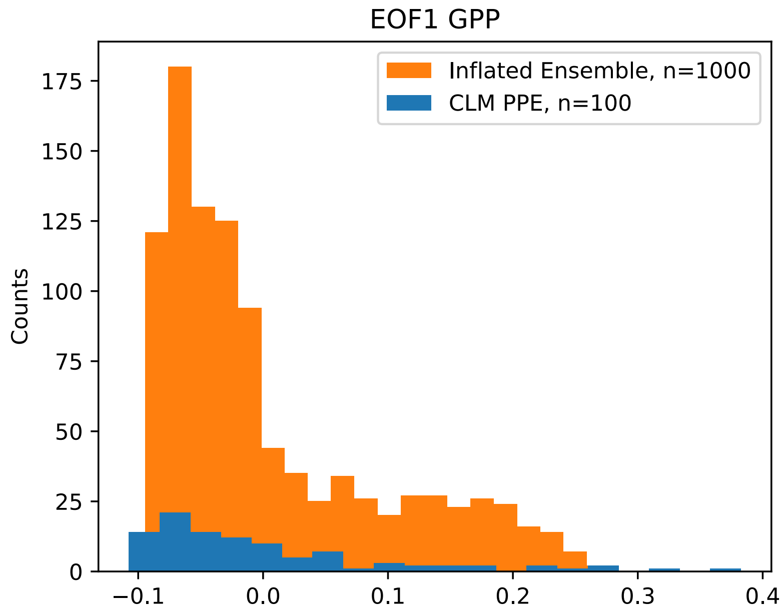
Input: **new parameter values and combinations**

	P1	P2	P3	P4	P5	P6
S1	x1,1	x1,2	x1,3	x1,4	x1,5	x1,6
S2	x2,1	x2,2	x2,3	x2,4	x2,5	x2,6
S3	x3,1	x3,2	x3,3	x3,4	x3,5	x3,6
…	…	…	…	…	…	…
…	…	…	…	…	…	…
S1000	x1000,1	x1000,2	x1000,3	x1000,4	x1000,5	x1000,6

Trained neural network emulator



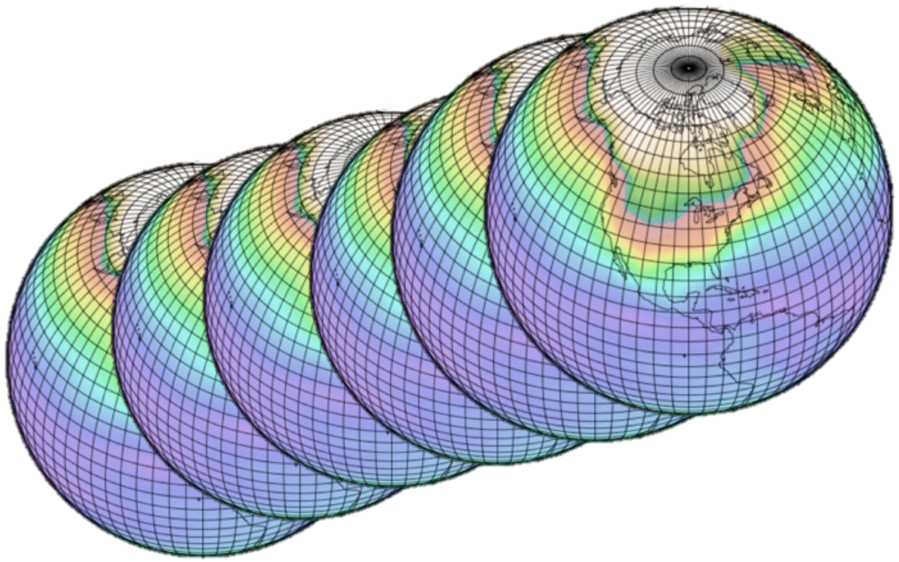
Output: **land model predictions**



The trained neural network can be applied to test new parameter values and combinations, much more quickly and efficiently than running the climate model.

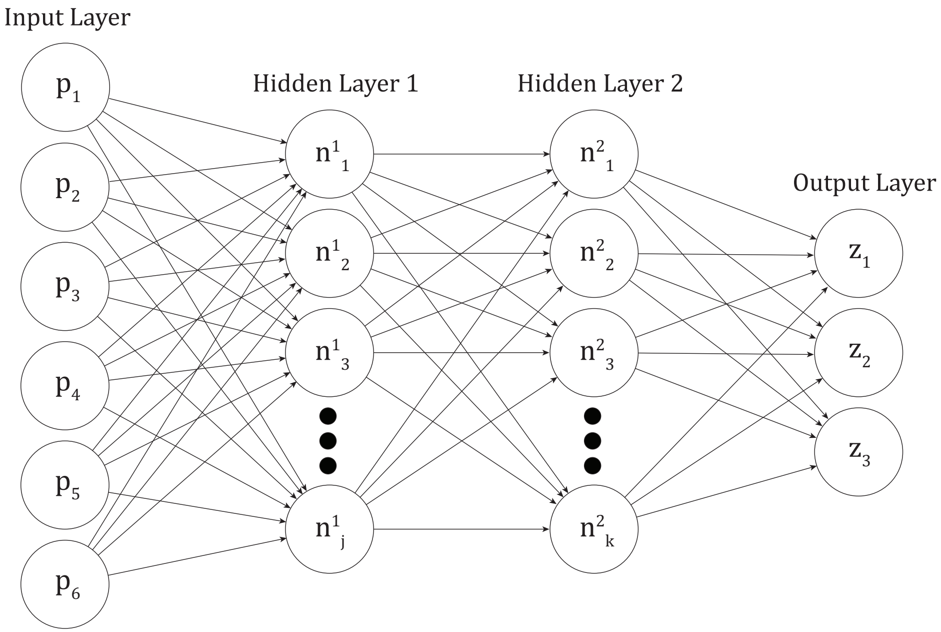
Increase in Computational Efficiency

Land model perturbed parameter ensemble



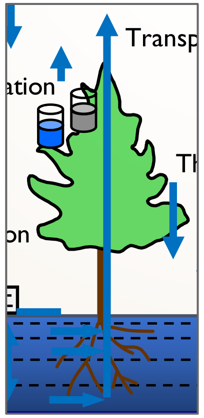
~2 hours per simulation

Machine learning emulator

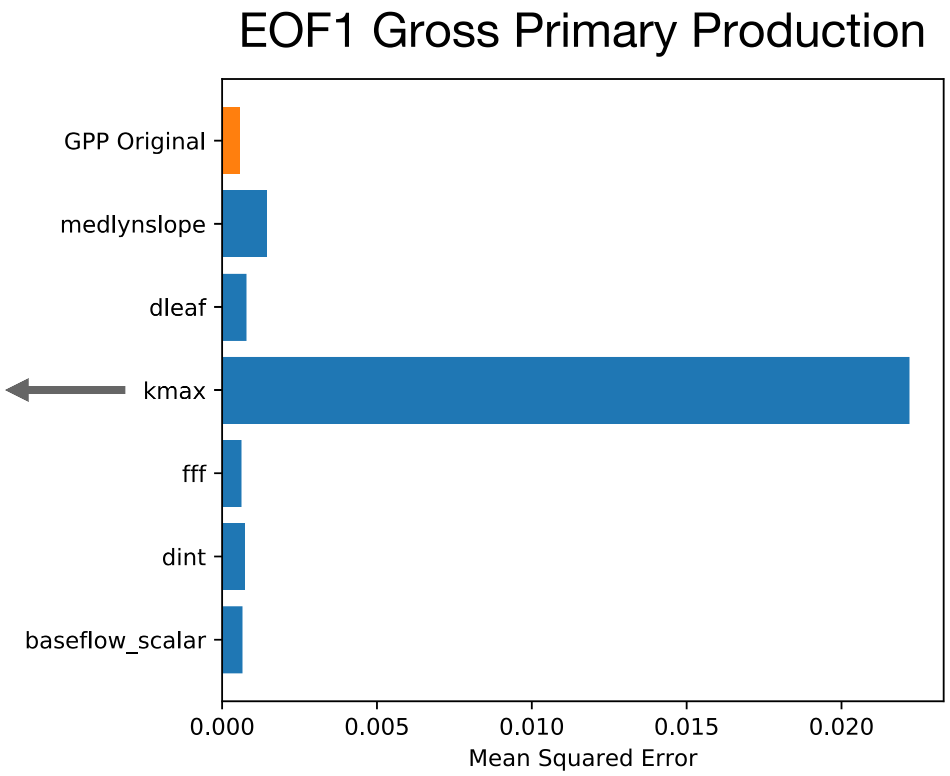


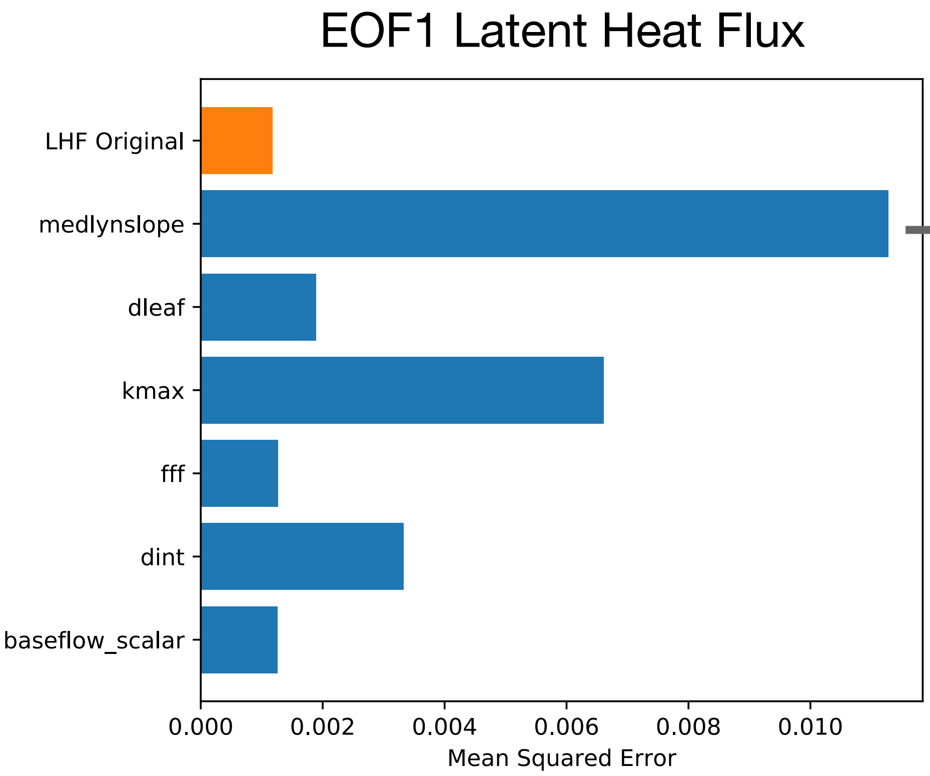
2.6 seconds to generate predictions!

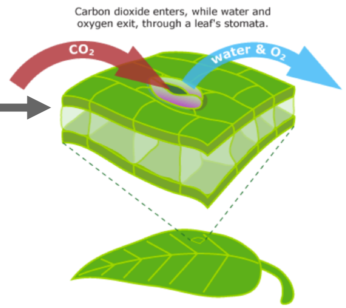
Model Interpretation: Variable/Feature Importance



Kennedy et al. (2019)







Medlyn et al. (2011)

<u>Variable/Feature Importance</u>

- Randomly shuffle values of one parameter (preserving others) and test performance of emulator.
- Skill metric is mean squared error between predictions and actual values.
- Larger bar means the parameter is **more important to the predictive skill** of the emulator.

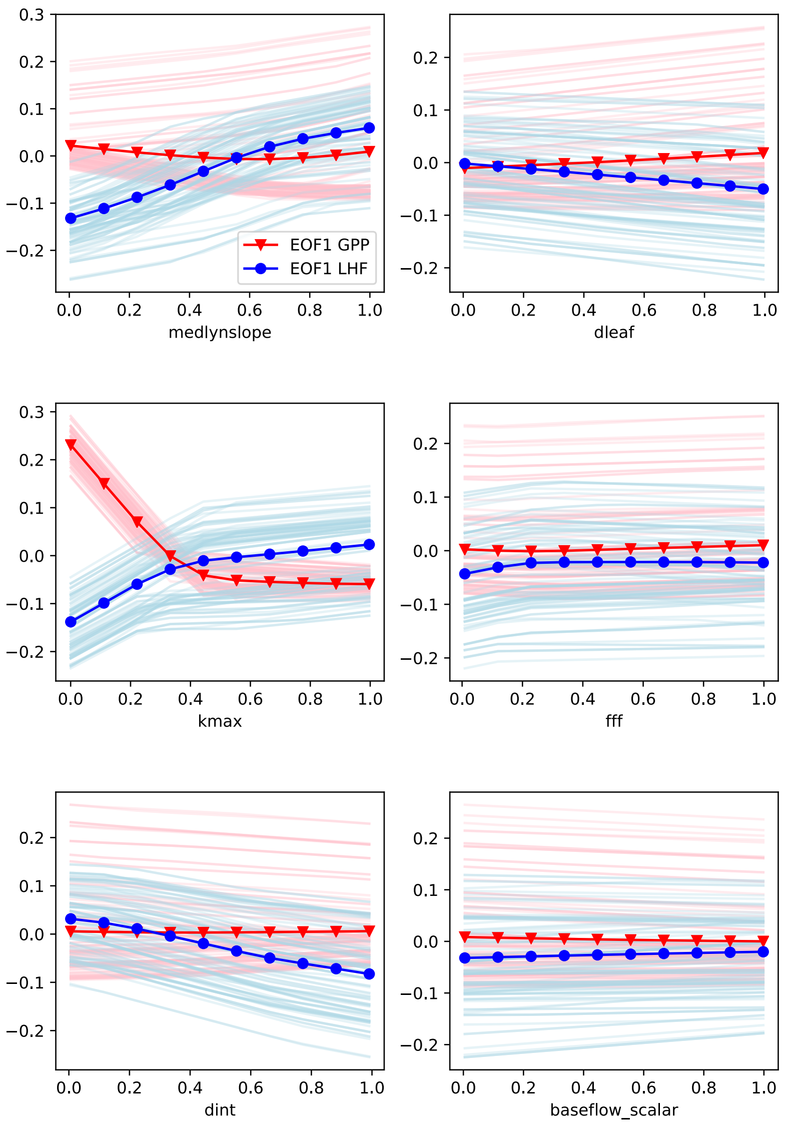
Dagon et al., *in review*

Model Interpretation: Partial Dependence Plots

- Test why a certain parameter is important, and plot where in its uncertainty range it is most important.
- Fix values of each parameter one at a time, and test performance of emulator across ensemble members.
- Regions of non-zero slope indicate **where in the parameter range the emulator is sensitive.**

Light colored lines = individual predictions (n=100)

Bolded lines = average prediction



Dagon et al., *in review*

Machine Learning Roadmap

1. *Train:* Build and train a series of **neural networks (NNs)** to predict land model output, given parameter values as input.

2. *Emulate:* Use trained NNs as **land model emulators** to make predictions with increased computational efficiency.

Questions so far?

Machine Learning Roadmap

1. *Train:* Build and train a series of **neural networks (NNs)** to predict land model output, given parameter values as input.

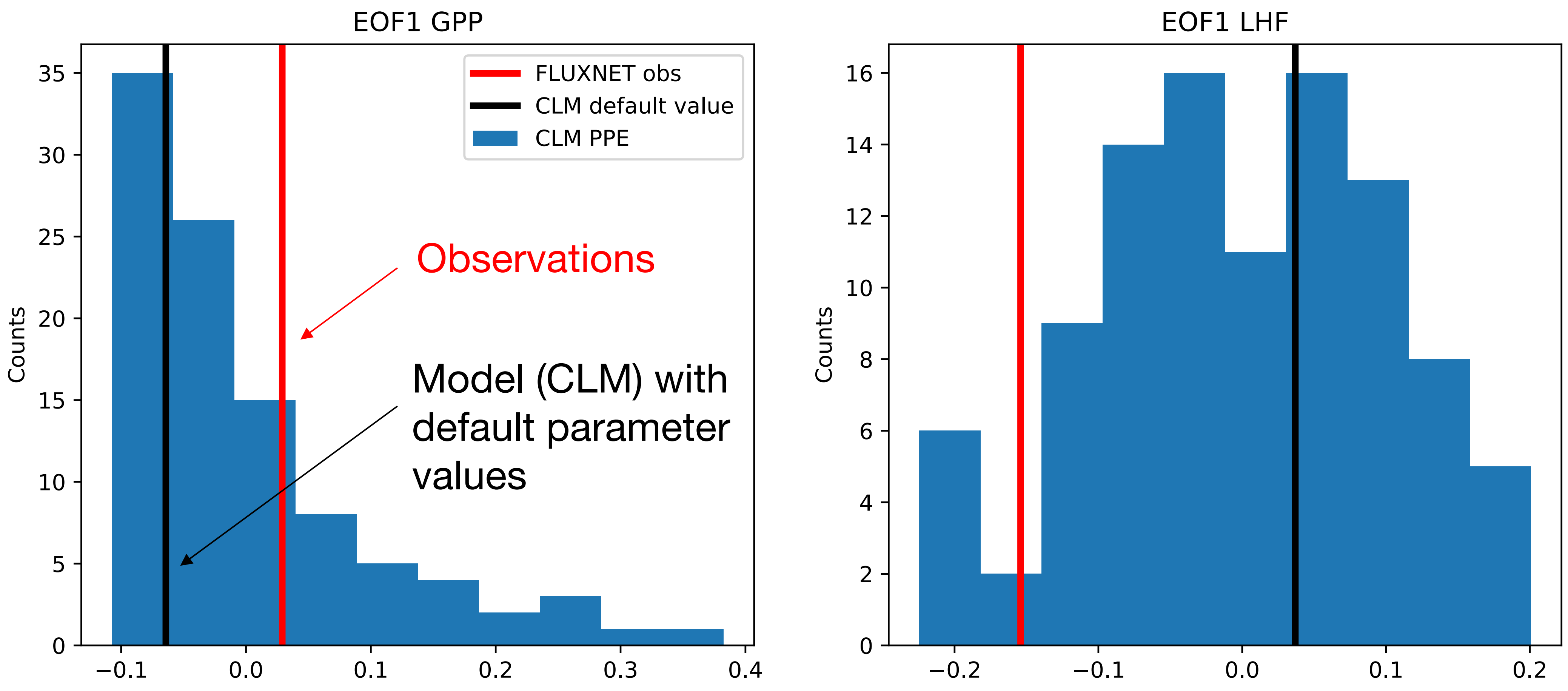
2. *Emulate:* Use trained NNs as **land model emulators** to make predictions with increased computational efficiency.

3. *Calibrate:* **Minimize error in predictions** relative to observations; generate optimal parameter values and distributions.

4. *Test:* Use optimal parameter values to **investigate changes in model predictive skill**.

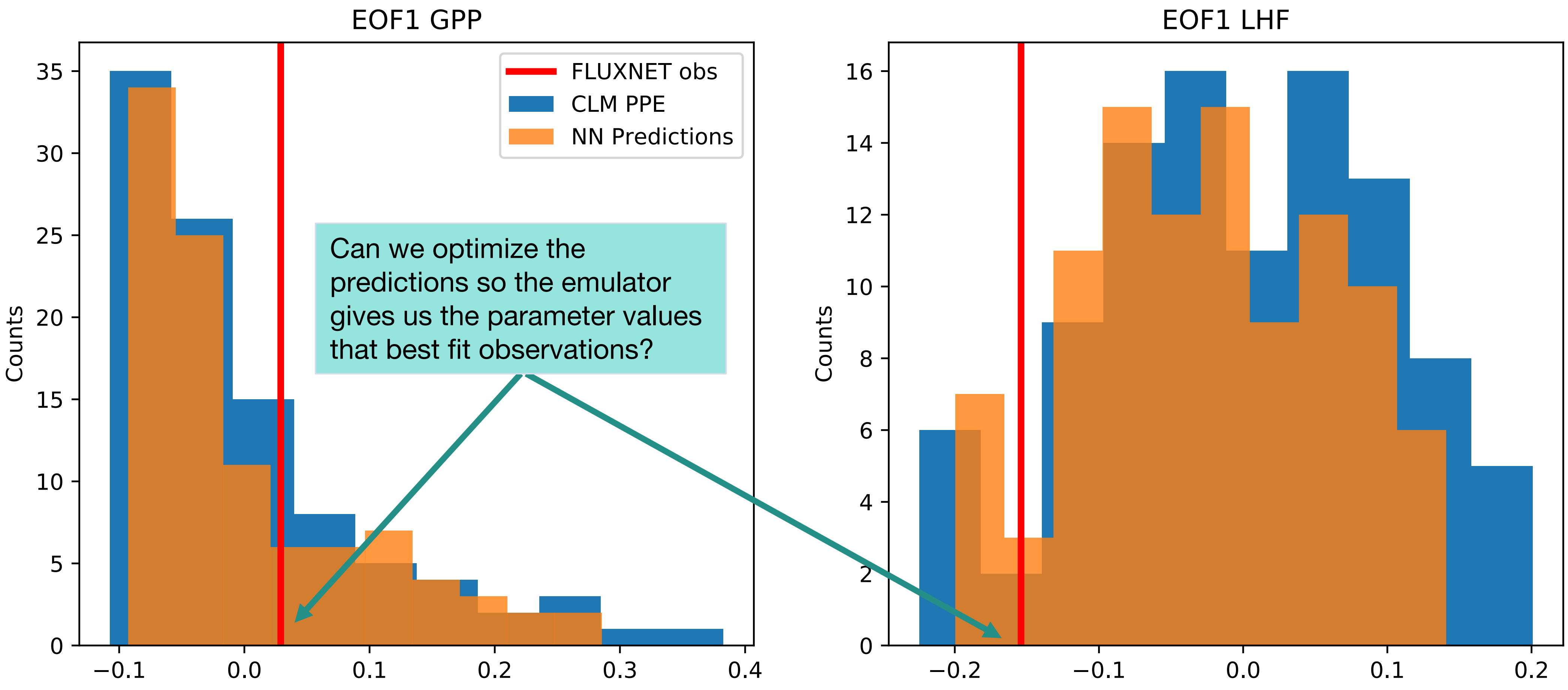
Optimizing the Emulator

Step 3: Calibrate



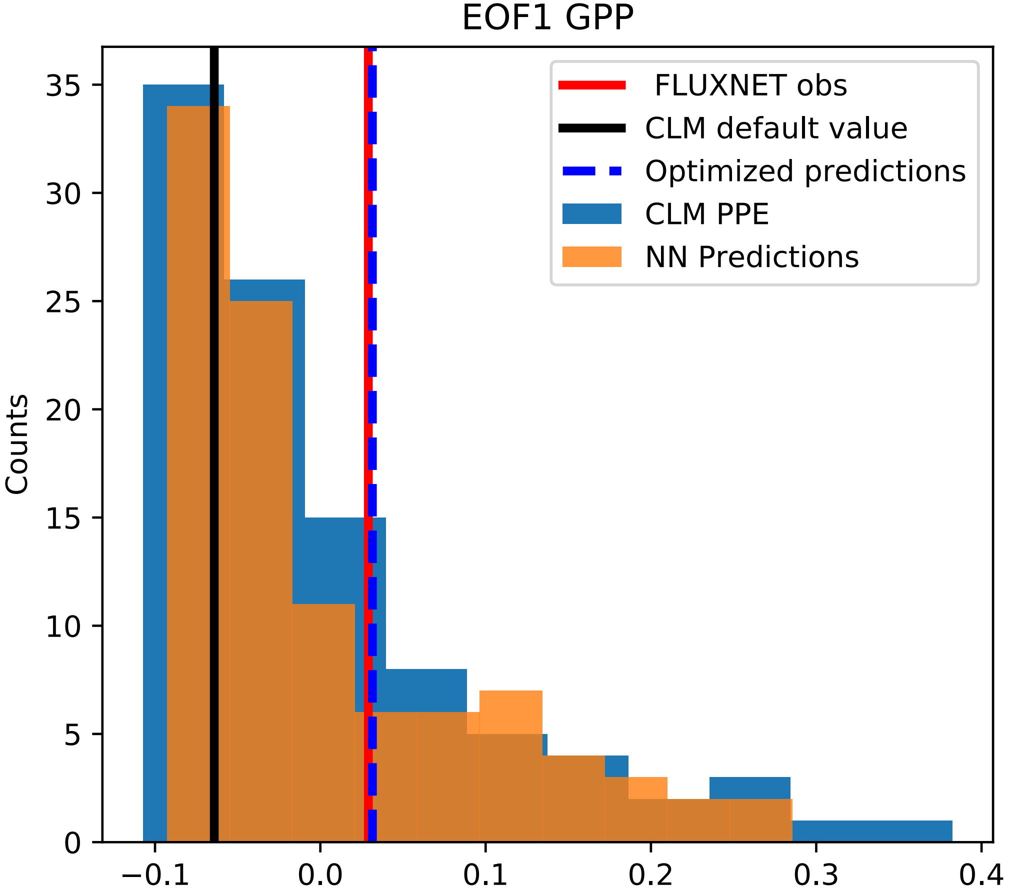
Optimizing the Emulator

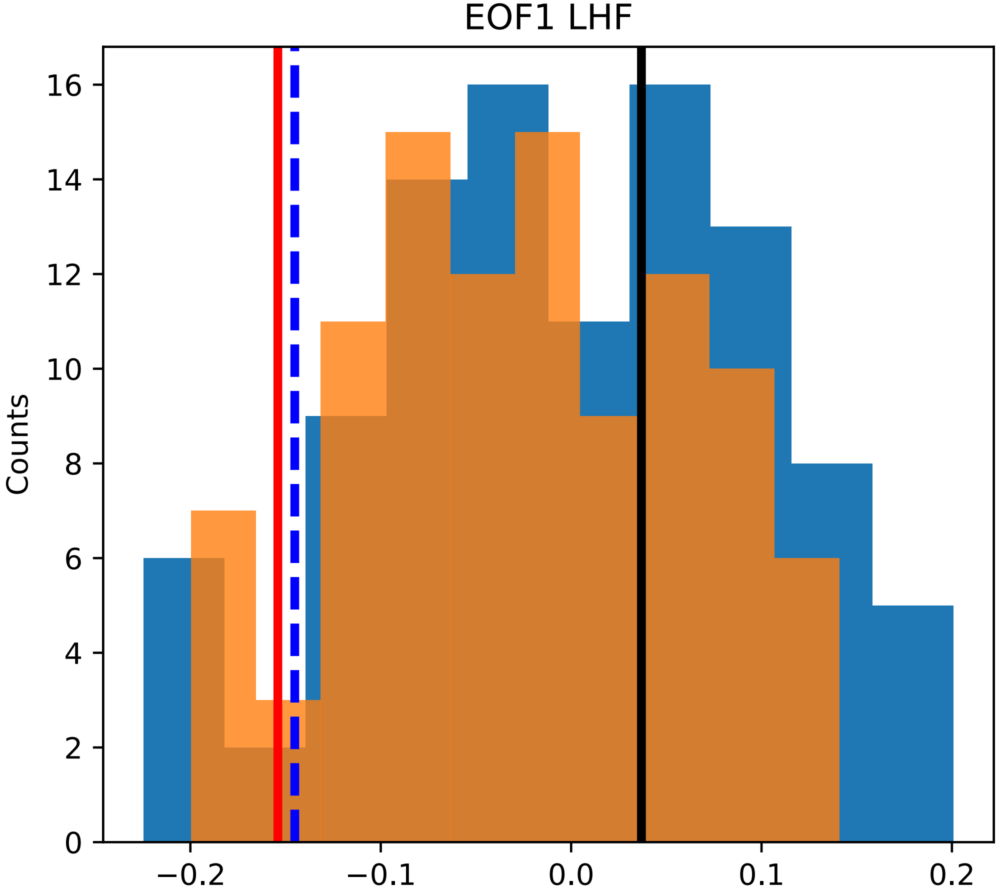
Step 3: Calibrate



Optimizing the Emulator

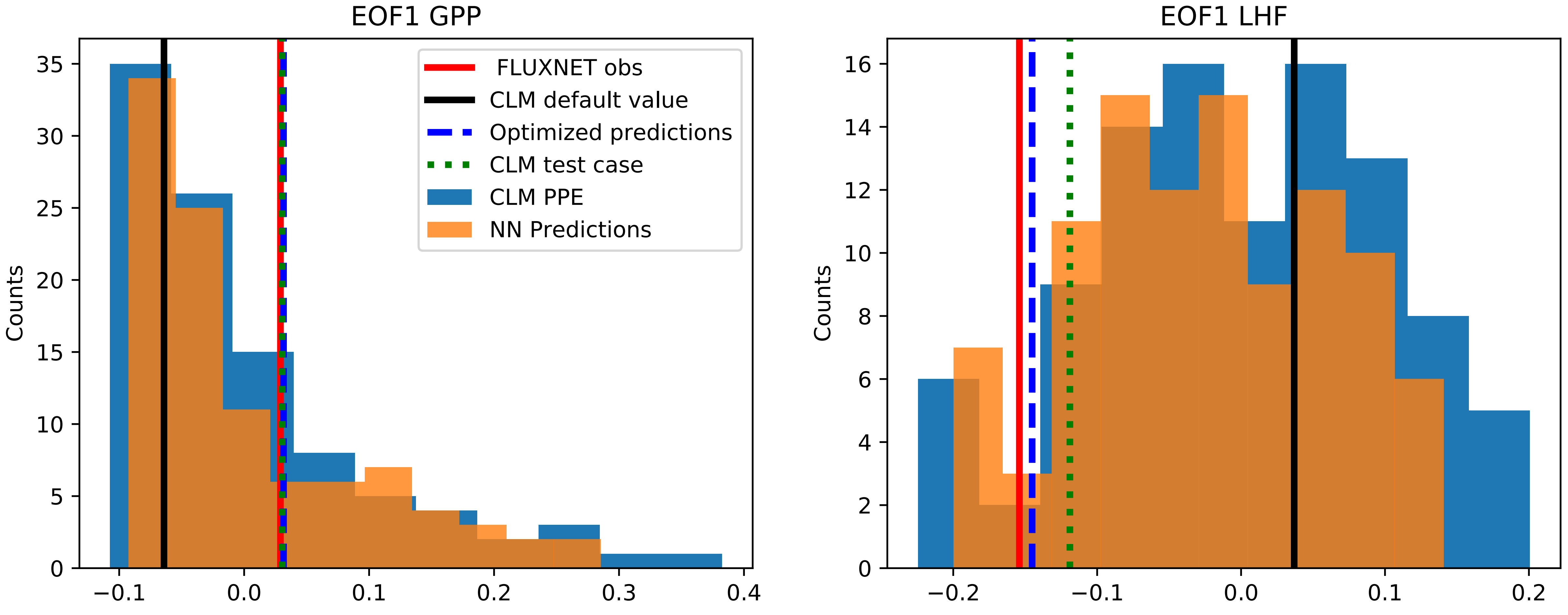
Step 3: Calibrate





Testing the Emulator Predictions

Step 4: Test

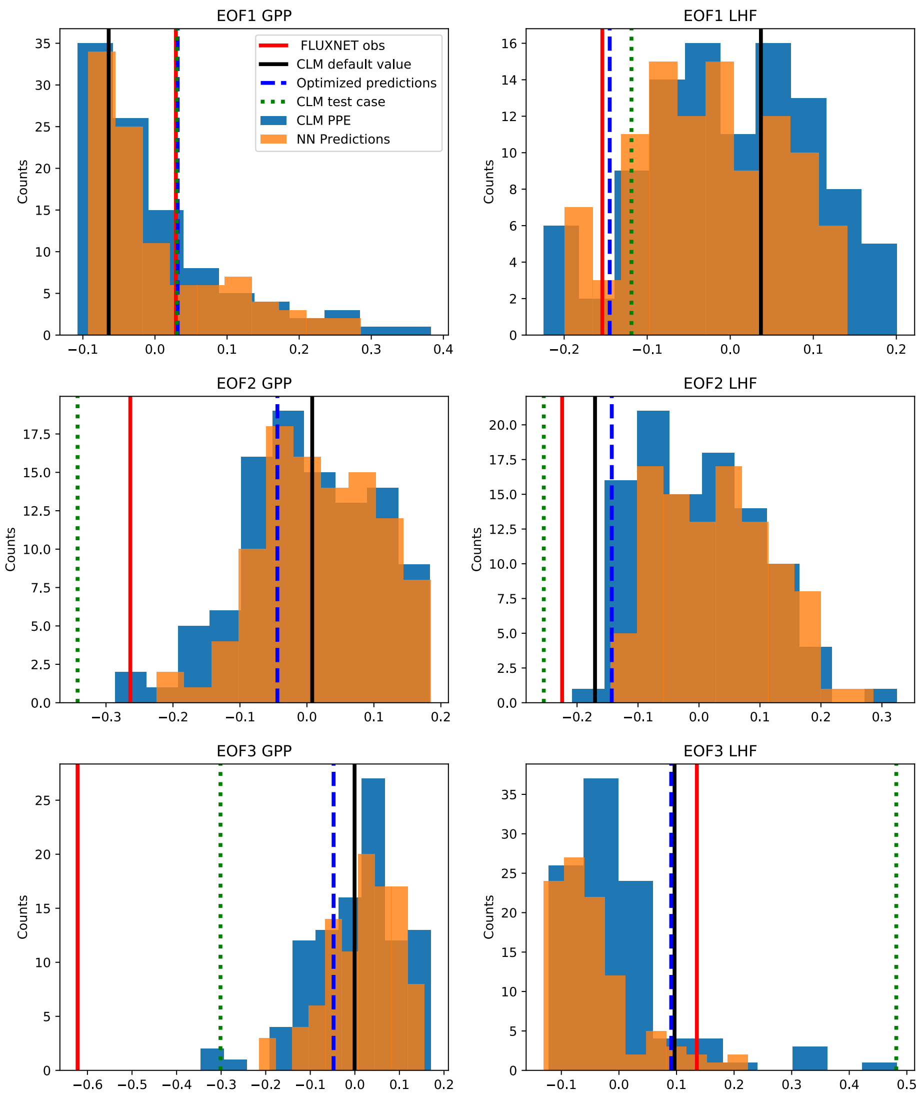


Dagon et al., *in review*

Testing the Emulator Predictions

Step 4: Test

Actually have 6 targets for calibration and optimization!

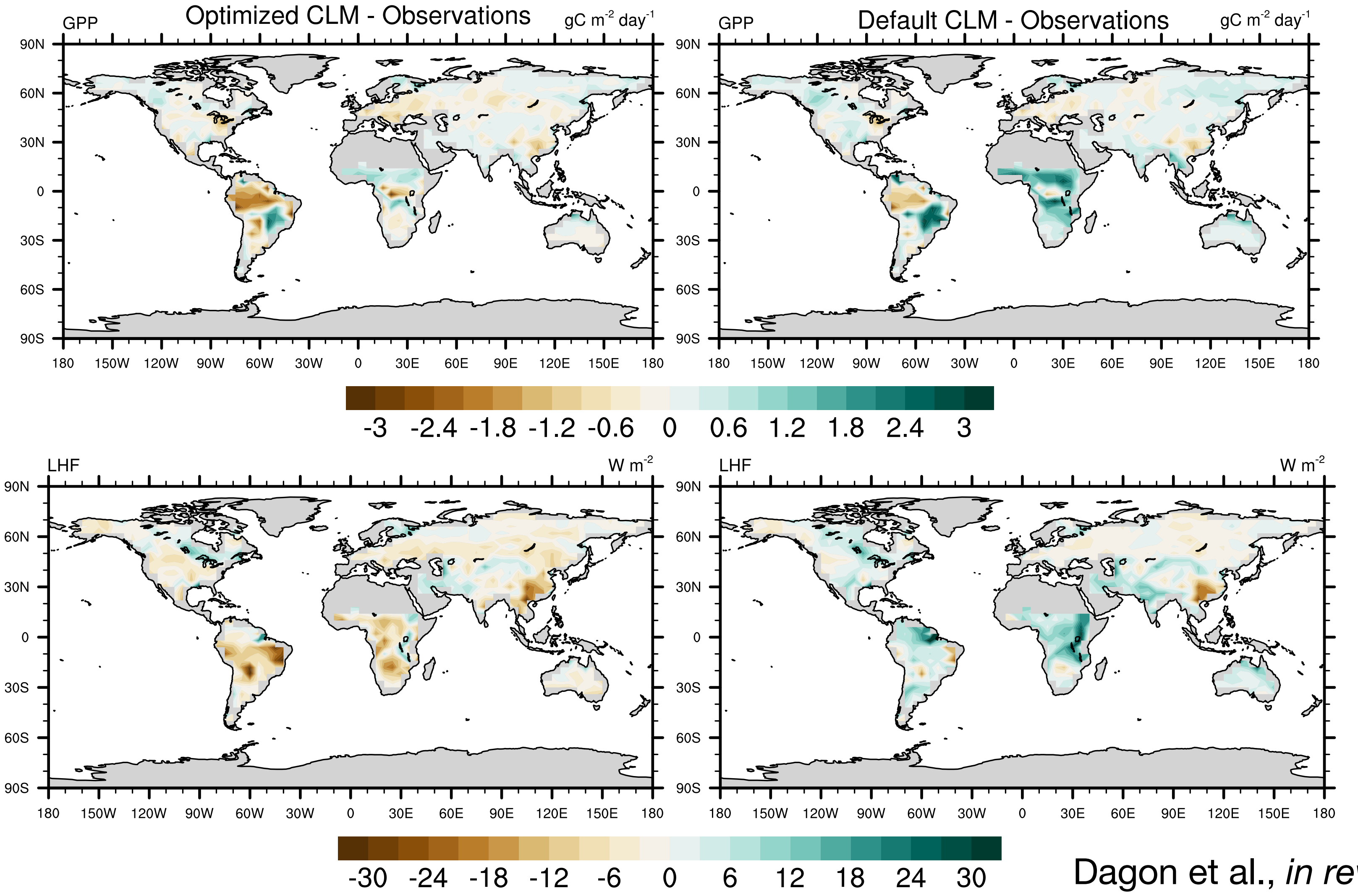


Dagon et al., *in review*

Testing the Emulator Predictions

Step 4: Test

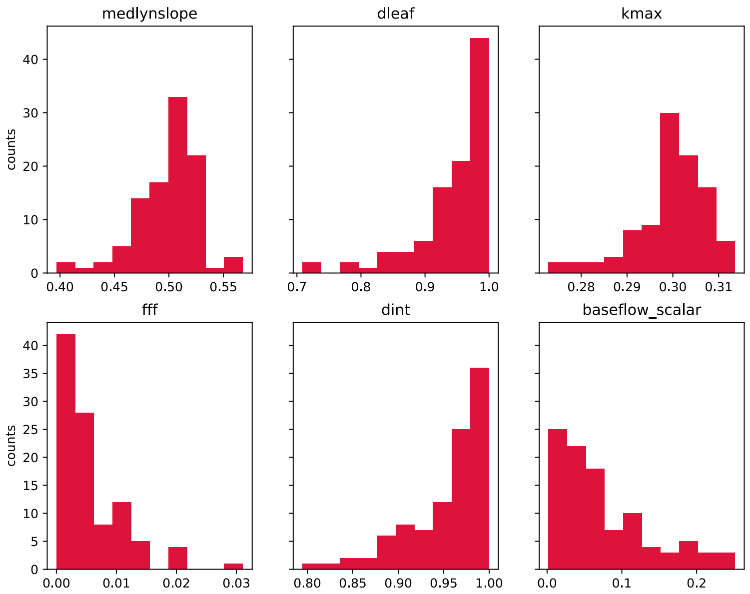
- Additional sources of uncertainty (forcing, observations, structural biases, other parameters)
- Choice of output variables (GPP and LHF)
- Choice of metrics (**annual mean** spatial variability as determined by EOF analysis)



Dagon et al., *in review*

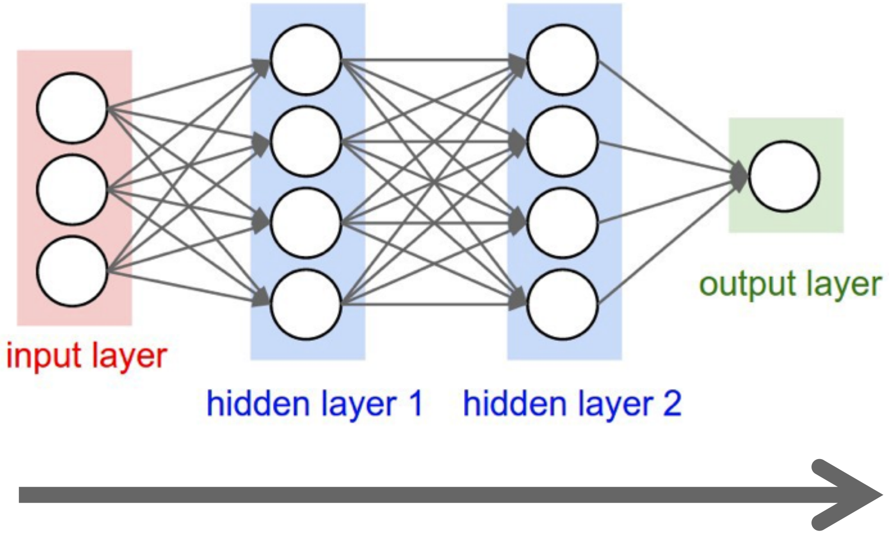
Putting results in the context of climate predictions

Input: Parameter posterior distributions

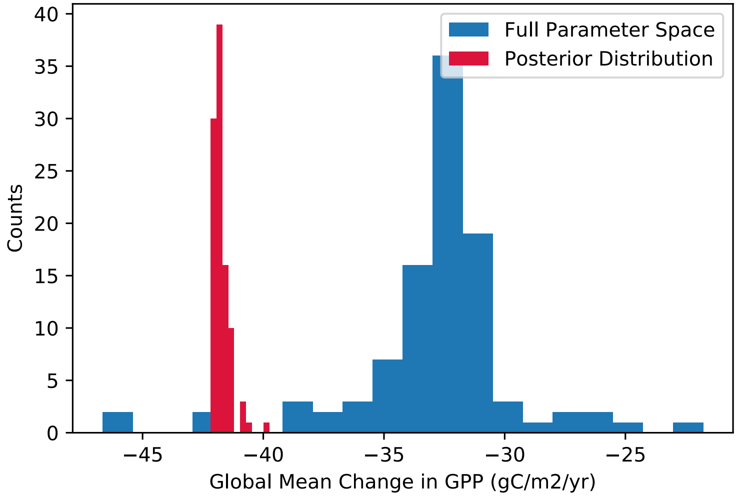


Best estimates for parameter values to match observations

DIFFERENT *neural network to emulate future climate response of land surface model*

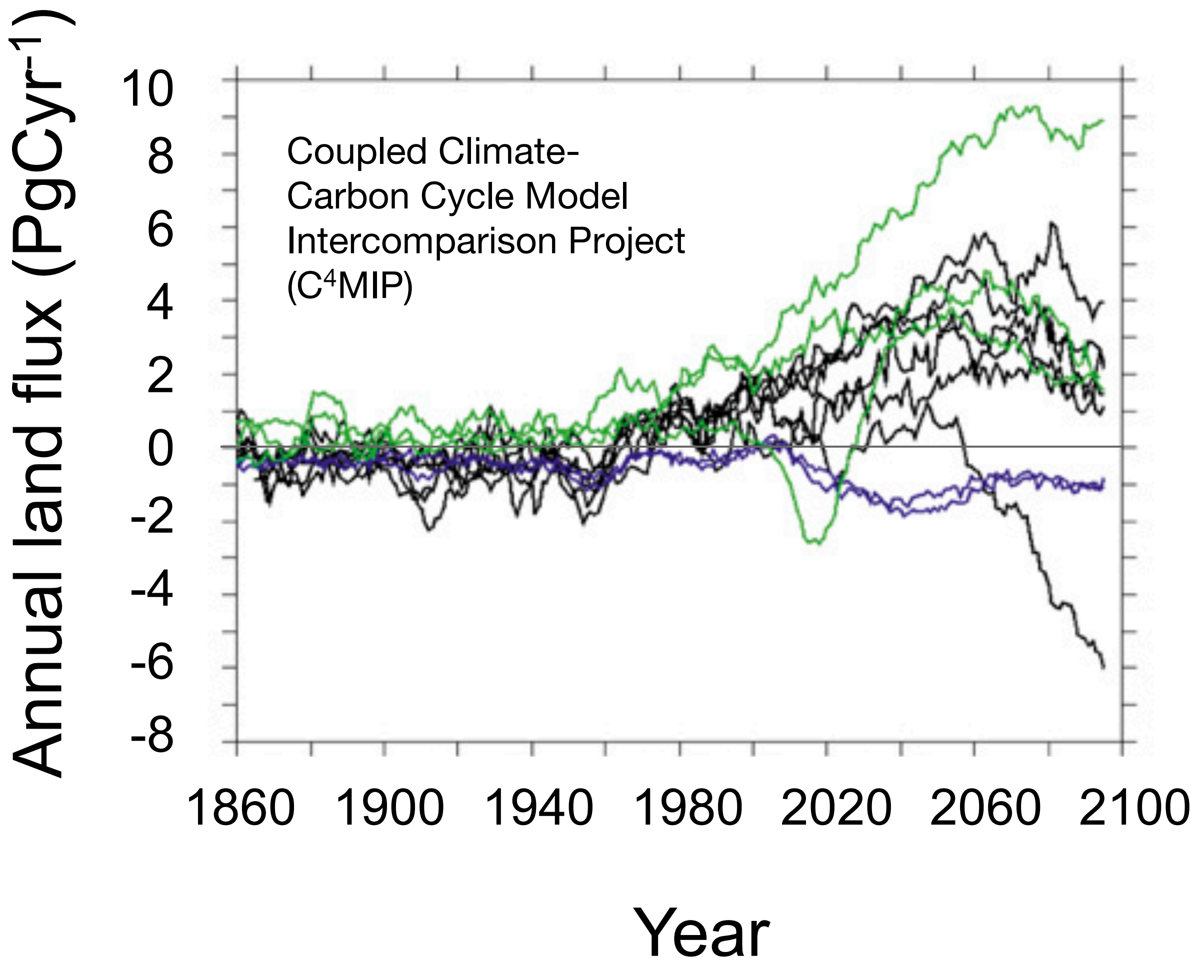


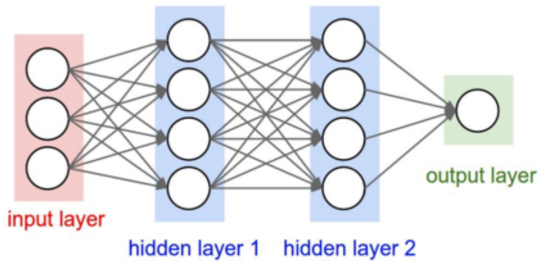
Output: Predicted change in GPP accounting for parameter uncertainty

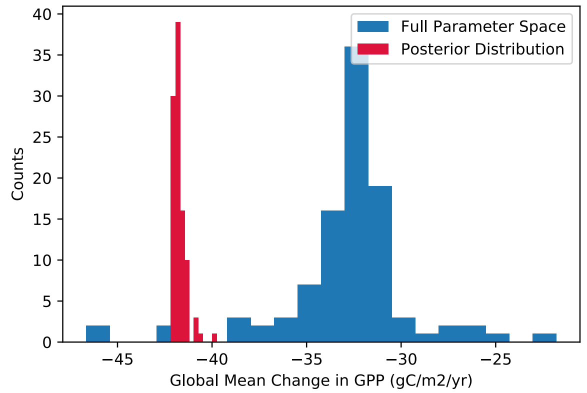


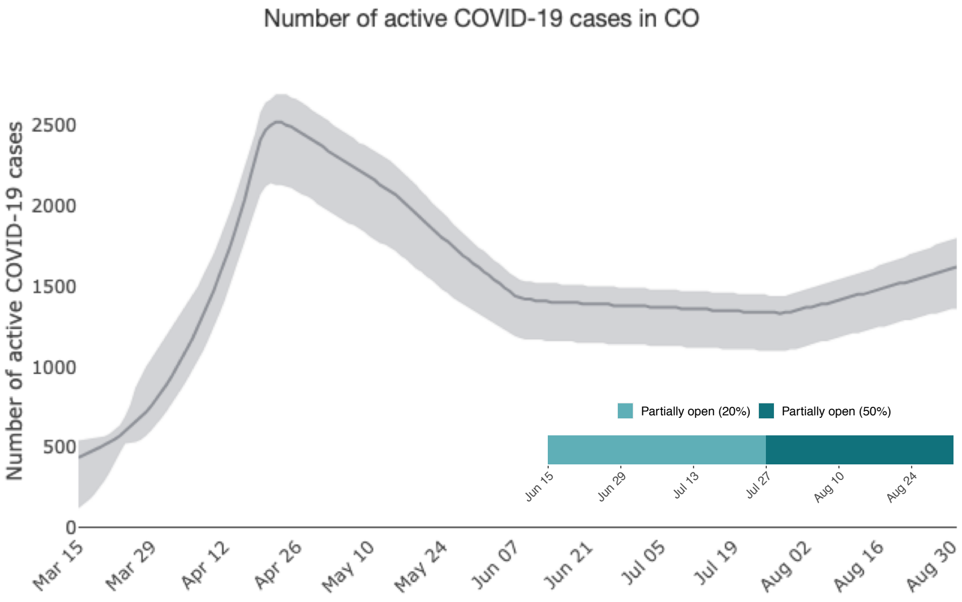
Dagon et al., *in prep*

Understanding and Communicating Uncertainties in Modeling









Source: MGH COVID-19 Simulator
https://analytics-tools.shinyapps.io/covid19simulator/

Summary

- Parameter choices are a **major contributor** to uncertainty in land model predictions.
- **Neural network emulators** can be trained to reproduce land model output with greater computational efficiency.
- Emulator predictions are **optimized to minimize error** between model and observations.
- Machine learning can help us **understand and communicate uncertainty** in modeling climate predictions.

Thanks!
Questions?

kdagon@ucar.edu
@katiedagon

Dagon, K., B.M. Sanderson, R.A. Fisher, and D.M., Lawrence, A machine learning approach to quantify biophysical parameter uncertainty in the Community Land Model, version 5, *in review*.

BACKUP SLIDES

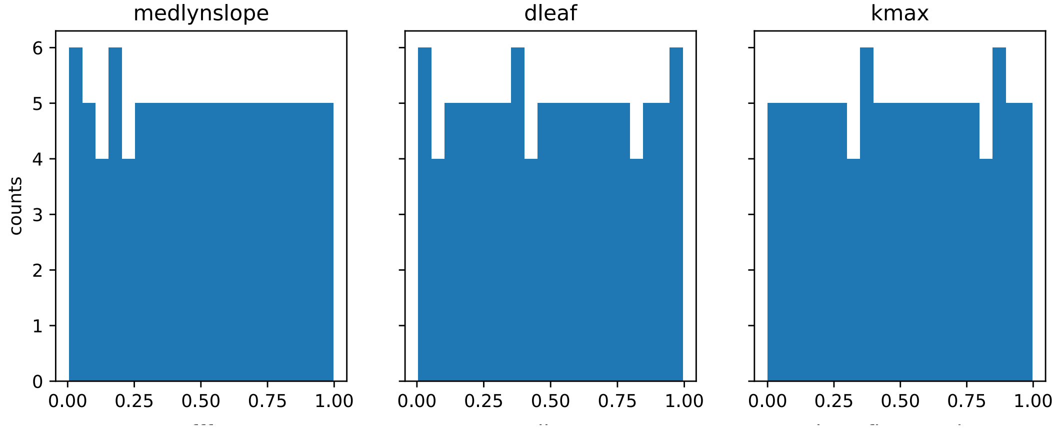
Land Model Parameters

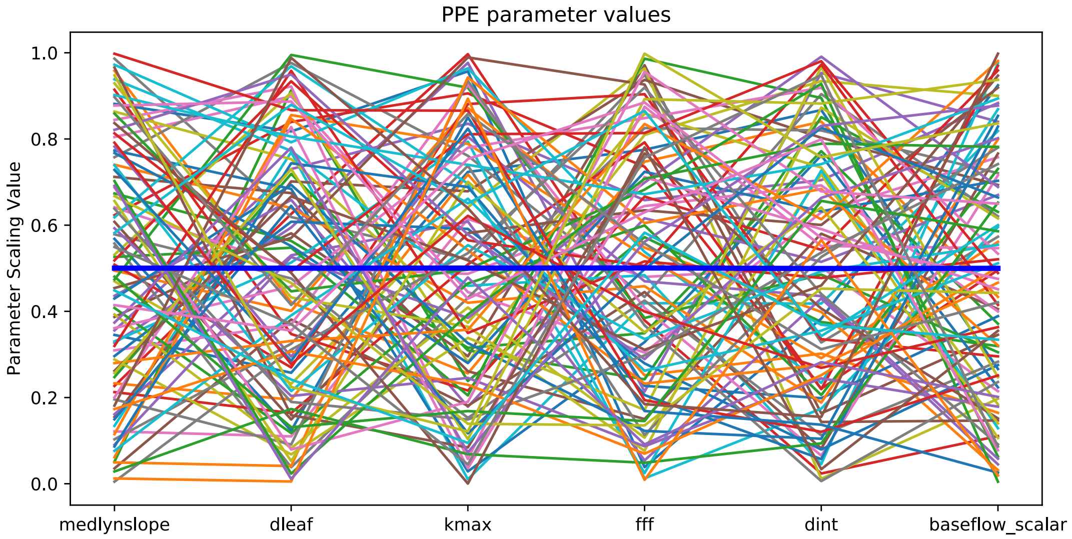
- Biophysical features (e.g., surface energy balance, hydrology, carbon uptake)
- Individual parameter uncertainty ranges determined by literature review, updated observations
- Parameter selection based on a series of sensitivity tests with objective metrics

Name	Biophysical parameter description
medlynslope	Slope of stomatal conductance-photosynthesis relationship
dleaf	Leaf boundary layer resistance parameter
kmax	Plant hydraulic stress parameter
fff	Surface runoff parameter
dint	Soil evaporation parameter
baseflow_scalar	Sub-surface runoff parameter

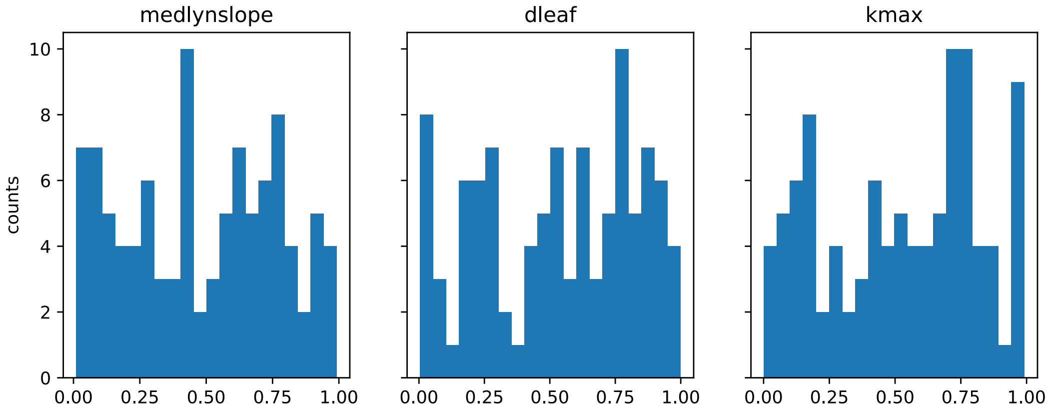
Parameter Sampling for PPE

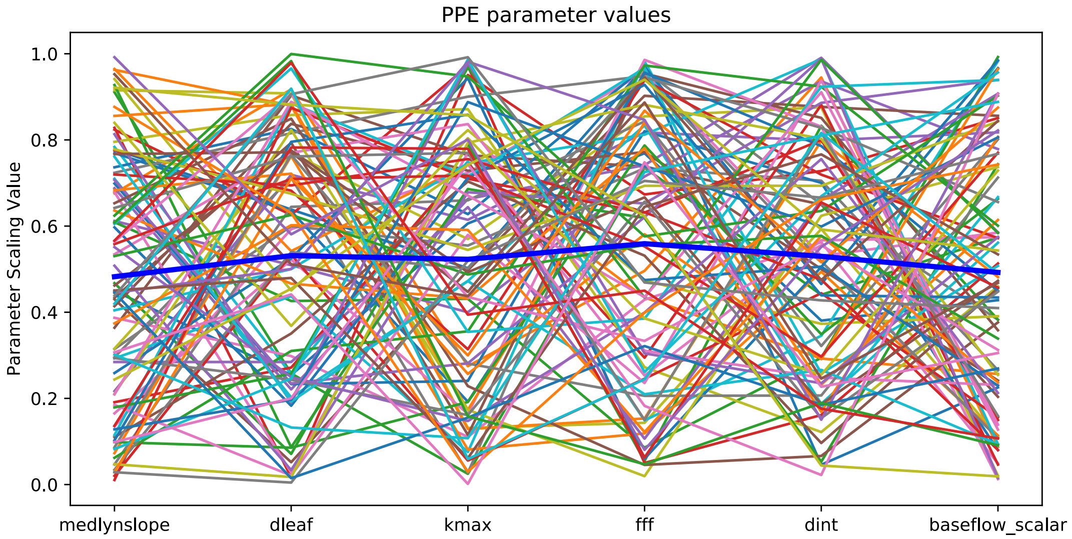
Latin Hypercube Sampling



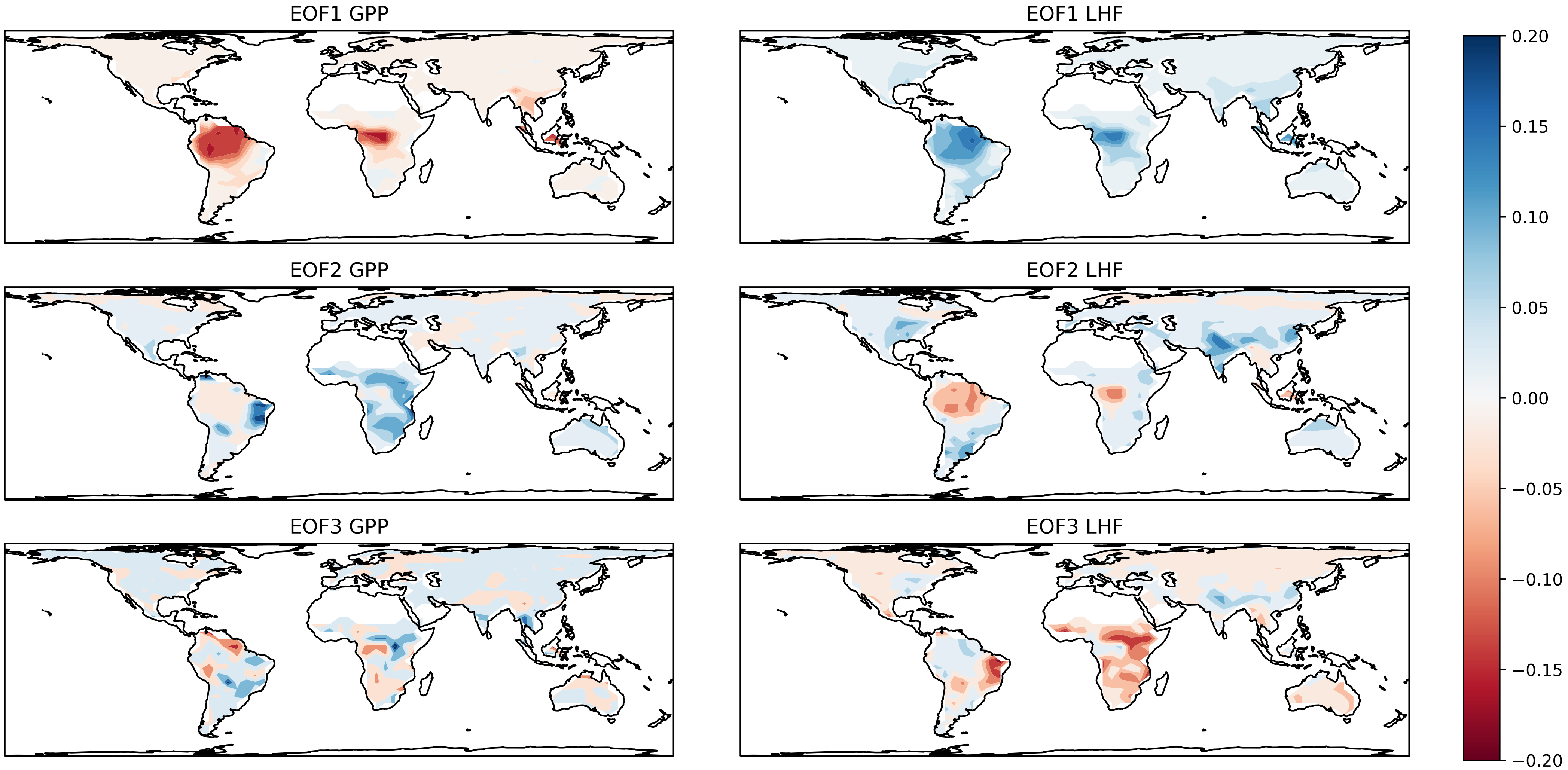


Random Sampling





EOF Analysis



Parameter Regressions