Introduction

Data

Method

Results

Discussion

Conclusion

References

Retrieval of sea surface temperatures (SSTs) from passive microwave measurements

Ann-Sofie, Emy, Marta & Yanet

Niels Bohr Institute, University of Copenhagen

28 May 2020

Overview

1. Introduction
2. Data
3. Method
4. Results
5. Discussion
6. Conclusion

Introduction

Goal

- Retrieval of sea surface temperature (SST) from passive microwave measurements (PMW)

Motivation

- SST is an important input for weather and ocean models and for understanding climate change
- Physical or linear regression-based algorithms are state-of-the-art
- Goal STD: 0.45 K
- Use machine learning to imrpove upon the retrievals

Data

The Advanced Microwave Scanning Radiometer 2 (AMSR2)

- Measures brightness temperatures at 7 different frequencies
- Spatial resolution ranging from approximately 4-50 km

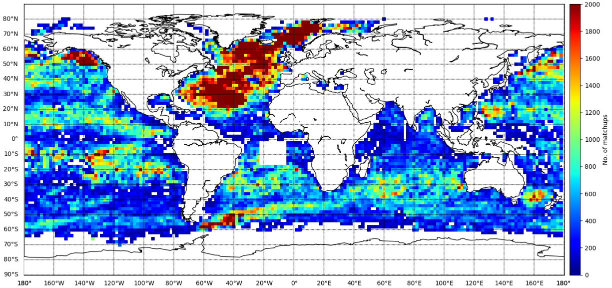
ERA-5 reanalysis product

- Surface winds, sea ice concentration etc.
- Spatial resolution of 31 km

In-situ measurements from drifting buoys

- Measures the SST at a depth of about 20 cm with an accuracy of approximately 0.2 K

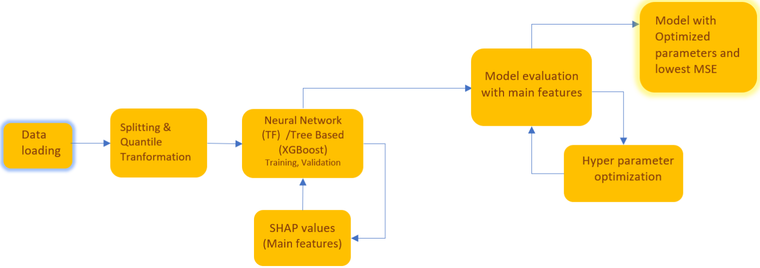
Data



- Matching in-situ measurements with satellite measurements
- Cleaning the data
- Homogeneous distribution

Method

Project Workflow:



Method: Feature Selection

Feature	XGB	TF NN	Feature	XGB	TF NN
TB6V			Lat		
TB6H			Lon		
TB7V			Solaz		
TB7H			Solza		
TB10V			Sataz		
TB10H			Satza		
TB18V			Sga		
TB18H			Rel_dir		
TB23V			Wind_dir		
TB23H			Wind_speed		
TB36V			TCWV		
TB36H			CLWT		
TB89V			SSS		
TB89H			Orbit		

Hyperparameter Optimization

TensorFlow NN

- Performed with SKLearn RandomizedSearchCV for 250 models (50 combinations and CV=5)

```
parameters_RandomizedSearch = {
    'hidden_layers': hidden_layers_opts,
    'activation_input_layer': ['relu','tanh'],
    'activation_hidden_layer': ['relu','tanh'],
    'optimizer_var':[tf.keras.optimizers.Adam,tf.keras.optimizers.SGD],
    'learning_rate':scipy.stats.uniform(0.0001,0.1),
     }
```

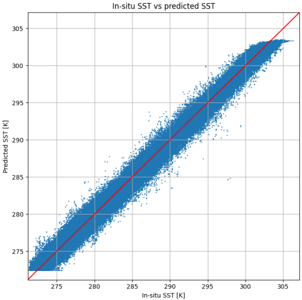
XGBoost

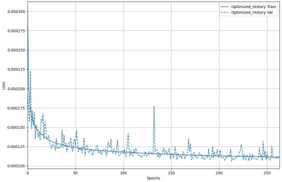
- Performed with SKLearn RandomizedSearchCV for 50 models (10 combinations and CV=5)

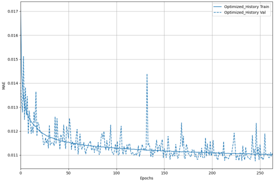
```
parameters_RandomizedSearch = {'n_estimators':scipy.stats.poisson(100), \
                               'max_depth':scipy.stats.poisson(25), \
                               'min_child_weight':scipy.stats.randint(1,5), \
                               'learning_rate':scipy.stats.norm(0.1,0.03), \
                               'subsample':scipy.stats.norm(0.6,0.1), \
                               'colsample_bytree':scipy.stats.norm(0.6,0.1), \
                               'colsample_bylevel':scipy.stats.norm(0.6,0.1)}
```


Introduction
Data
Method
Results
Discussion
Conclusion
References

TensorFlow Keras Dense Model results

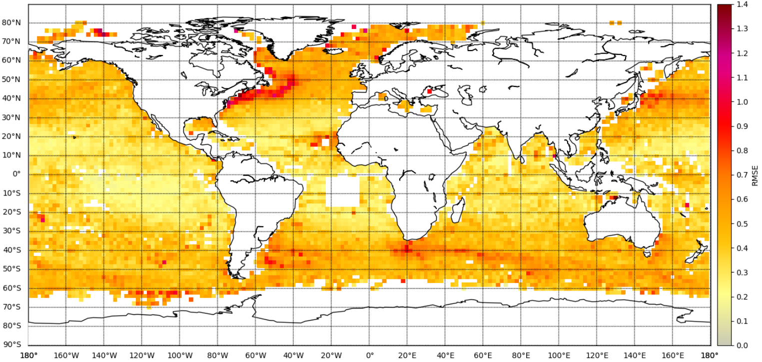




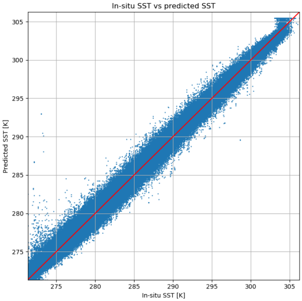


R^2	0.998
MSE	0.202 K^2
MAE	0.330 K
Mean error	-0.076 K
Std.	0.442 K

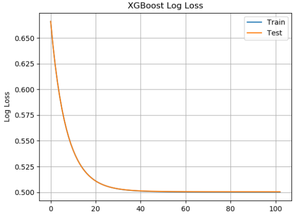
TensorFlow Keras Dense Model results

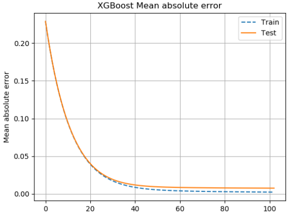


XGBoost results

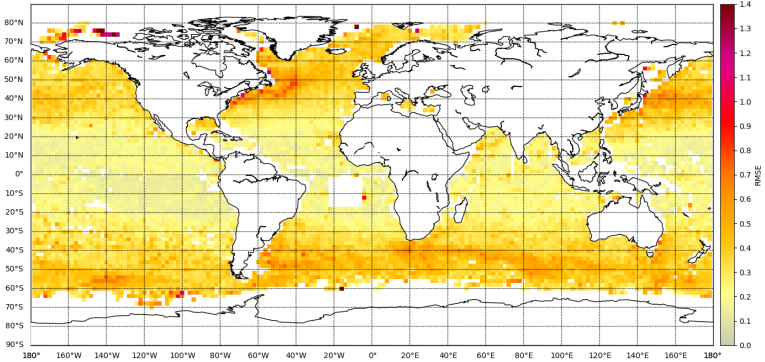


R^2	0.999
MSE	0.118 K^2
MAE	0.215 K
Mean error	0.007 K
Std	0.344 K





XGBoost results



Discussion

Comparison between XGBoost and TensorFlow NN

- XGBoost is easier to work with
- TensorFlow NN is harder to optimize
- TensorFlow NN is much slower to train

Problems

- Uncleaned and unbalanced dataset contains approx. 30 million data points
- Even after cleaning and balancing we had 10 million data points - GPU!
- Google colab run time limit
- Long HPO times with RandomizedSearchCV (+12 h)

Conclusion

- XGBoost performed the best!
- Might have used a too simple NN
- Both models perform better or almost as good as state-of-the-art retrieval algorithms (Alerskans et al., 2020)

	State-of-the-art	XGBoost	TFKeras
Mean error [K]	0.002	0.007	-0.076
Std [K]	0.45	0.344	0.442

Future work

- Validate on a completely independent data set - Argo buoys (much fewer matchups!)
- The precursor of AMSR2, AMSR-E, have observations for the period 2002-2011
- The follow-on satellite, CIMR, will hopefully be launched
- Use the same model(s) to retrieve SST from the other satellite(s)

Questions?

References

Emy Alerskans, Jacob L Høyer, Chelle L Gentemann, Leif Toudal Pedersen, Pia Nielsen-Englyst, and Craig Donlon. Construction of a climate data record of sea surface temperature from passive microwave measurements. *Remote Sensing of Environment*, 236:111485, 2020.

Appendix

Project comments

All group members have contributed to the project equally.

We chose to work on 4 different models in order to be able to compare the performace of tree and NN based algorithms, and to compare how different NN structure affects the final result. Moreover, we wanted to make sure each of us learns completely how to implement a model applied to a real life problem.

Feature Selection

XGB 18 features	tb6V	tb7V	tb10V			tb36V	tb89V
	tb6H	tb7H	tb10H			tb36H	
	lat			satza		sga	solza
	relative_dir	wind_speed	tcwv			lon	clwt
TF1 22 features	tb6V	tb7V	tb10V	tb18V	tb23V	tb36V	tb89V
	tb6H	tb7H	tb10H	tb18H	tb23H	tb36H	tb89H
		solaz	sataz	satza			
	relative_dir	wind_speed	tcwv	sss			
TF2 17 features	tb6V	tb7V	tb10V	tb18V	tb23V	tb36V	tb89V
	tb6H	tb7H	tb10H	tb18H	tb23H	tb36H	tb89H
	lat	solaz					
	relative_dir						
TF3 22 features	tb6V	tb7V	tb10V		tb23V	tb36V	
	tb6H	tb7H	tb10H	tb18H	tb23H	tb36H	tb89H
	lat	solaz	sataz		orbit	sga	solza
	relative_dir	wind_speed		sss	wind_dir		

Table: Features chosen for the models (based on SHAP). The general agreement was to have maximum 22 features and to disregard physical importance.

XGBoost - Data and model details

The cleaned data has been split into a training and testing set by using `train_test_split()` from Scikit-Learn, with a random state of 42 and a test fraction of 0.5.

The reason for the big test set is that we have so much data, that we would run into memory issues if we used a higher fraction to go into the training dataset.

The model took around 13-14 hours to do HPO on when using a standard laptop. Only half of the training data was used to do the optimization due to memory issues (GPU on Google Colab was tried, but due to the long computational time, this was not suited for Google Colab). The model was then fitted/trained twice, first with the same part of the training data as was used for the HPO and then with the second part of the training data. Each fitting/training took around 30-40 min.

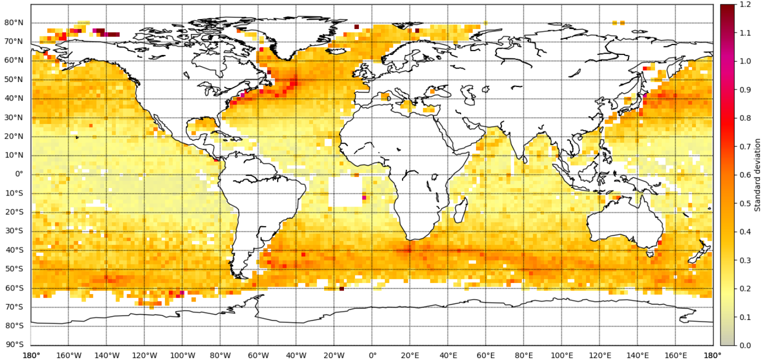
The size of the final model is 547.3 MB.

XGBoost - Standard deviation

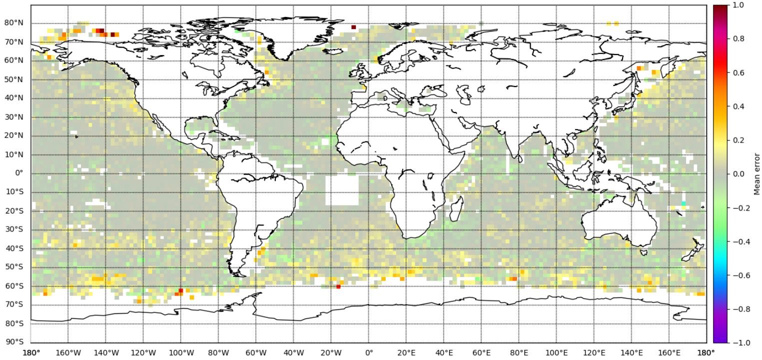
Retrieval of sea surface temperatures (SSTs) from passive microwave measurements

Ann-Sofie, Emy, Marta & Yanet

Appendix



XGBoost - Mean error



TensorFlow Keras NN Algorithm 1 - Datasets

The cleaned and balanced dataset contained approximately 10 million data points. In order to handle all of the data it was split into a train and test dataset using SKLearn's `train_test_split()` with `test_frac=0.5` and `random_state=42`.

When performing the HPO, it was necessary to further split the dataset, in order to not encounter memory problems. Since a lot of combinations were tested, the HPO took roughly 9 hours, which also makes it not suited to run on google colab due to the connection being lost after 90 min (if you're not active). Hence HPO was performed on half of the training dataset.

TensorFlow Keras NN Algorithm 1 - Model details

Model code:

```
def build_keras_base(hidden_layers, feature_input_dim, n_outputs,
                     activation_input_layer, activation_hidden_layer,
                     loss_var, optimizer_var=tf.keras.optimizers.Adam,
                     learning_rate,metrics_var,dropout_rate):

    model = tf.keras.Sequential()
    for layer_index, neurons_in_layer in enumerate(hidden_layers):
        if (layer_index == 0):
            # Input layer:
            # + Specify the input_dim to be the number of features for the first layer
            model.add(tf.keras.layers.Dense(neurons_in_layer, activation=activation_input_layer, input_dim=feature_input_dim))
        else:
            # Hidden layers:
            model.add(tf.keras.layers.Dense(neurons_in_layer, activation=activation_hidden_layer))

    # Dropout rate
    if dropout_rate:
            model.add(tf.keras.layers.Dropout(p=dropout_rate))

    # Output layer
    model.add(tf.keras.layers.Dense(n_outputs))

    # Compile the model
    model.compile(loss=loss_var, optimizer=optimizer_var(learning_rate), metrics=metrics_list)

    # Print a model sumary
    print(model.summary())

    return model
```

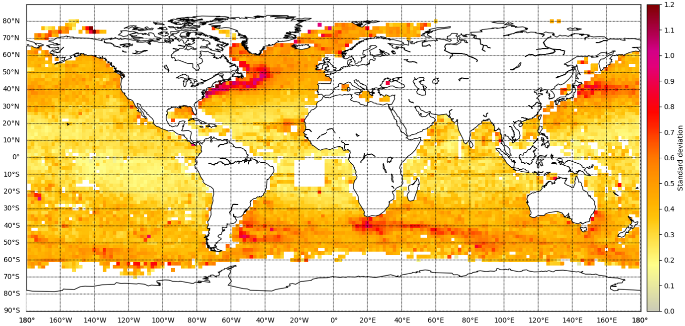
Optimized hyperparameters:

```
parameters = {'hidden_layers': (256, 32, 32, 128, 128, 32, 64, 64),
              'activation_input_layer': 'relu',
              'activation_hidden_layer': 'relu',
              'optimizer_var': tf.keras.optimizers.Adam,
              'learning_rate': 0.001426496115986653}
```

Fixed parameters:

```
fixed_params = {'feature_input_dim': Nfeature_max,
                'n_outputs':1,
                'loss_var': 'logcosh',
                'dropout_rate': 0,
                'metrics_var':('mse','mae')}
```

TensorFlow Keras NN Algorithm 1 - Standard deviation

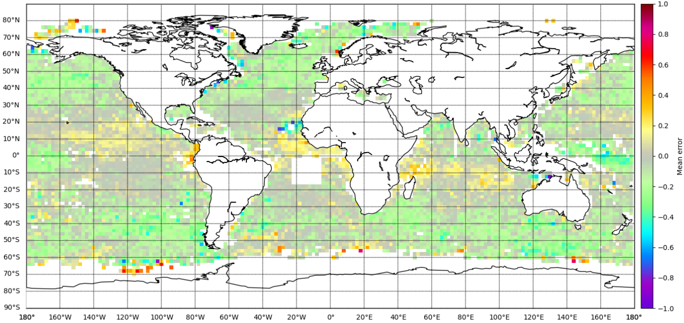


TensorFlow Keras NN Algorithm 1 - Mean Error

Retrieval of sea surface temperatures (SSTs) from passive microwave measurements

Ann-Sofie, Emy, Marta & Yanet

Appendix



TensorFlow Keras NN Algorithm 2: About the Model

This model was developed in google colabs, therefore it could handle many data at once, but not for long periods of time. The method for developing this model was influenced by this fact. The data it was split into a train and test dataset using SKLearn's `train_test_split()` with `test_frac=0.2`.

The HPO was performed on quarter of the dataset, and the number of total iterations was 50. Therefore, HPO took no more than 3 hours.

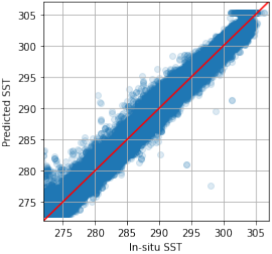
TensorFlow Keras NN Algorithm 2: RandomizedSearchCV

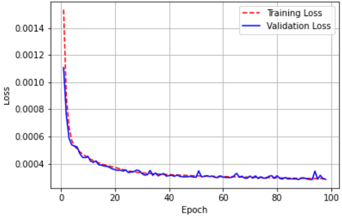
Features for this model are given in slide 19.
Below, the search parameters are listed with their distributions. Indexes 1, 2, 3, 4 correspond to the consecutive layers. The search ran for 10 iterations with CV=5, ie. 50 iterations in total. The final model structure is showed below (right). The model ran on 150 epochs maximum, but early stopping was implemented with patience 10. The batch size for this model is 64.

```
layer1 = [16, 32, 64, 128]
layer2 = [8, 16, 32, 64]
layer3 = [4, 8, 16, 32]
act2 = ['elu', 'relu', 'sigmoid', None]
act3 = ['elu', 'relu', 'sigmoid', None]
act4 = ['elu', 'relu', None]
dropout = [0., 0.25, 0.5, 0.75]

n_iter_search = 10
```

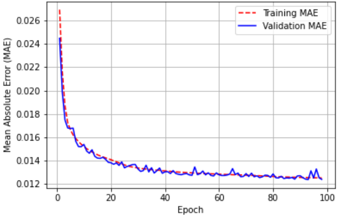
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

model = keras.Sequential([
    Dense(128, input_dim=X_train.shape[1]),
    Dense(64, activation='sigmoid'),
#     Dropout(0.5),
    Dense(4, activation='relu'),
    Dense(1, activation=None)
])

my_callbacks = EarlyStopping(monitor='val_loss',
                             mode='min',
                             verbose=1,
                             patience=10)

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss="mse",
              metrics=["mae", "mse"])
```

TensorFlow Keras NN Algorithm 2: Error analysis

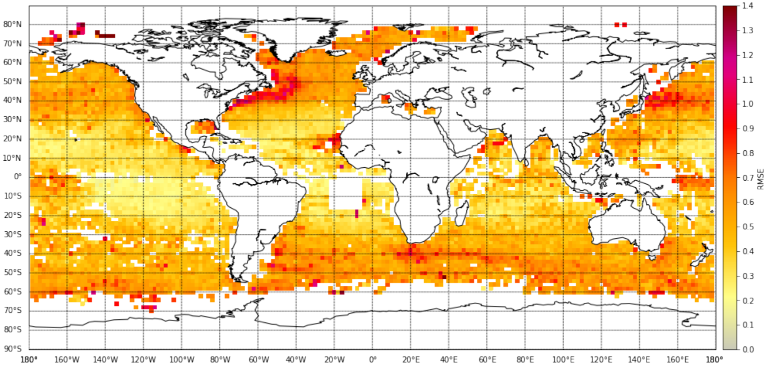




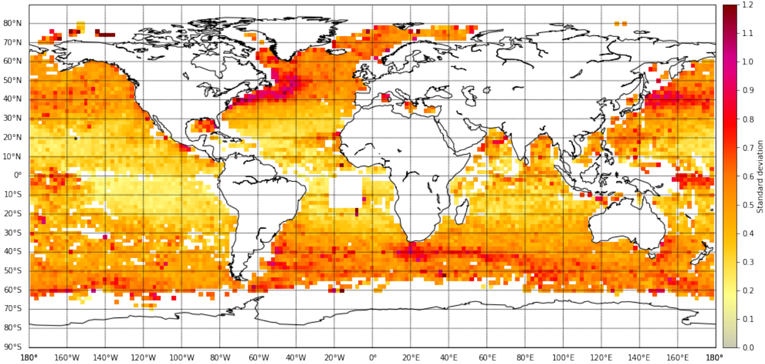
R^2	0.997 K
MSE	0.273 K
MAE	0.375 K
Mean error	0.002 K
Std	0.522 K



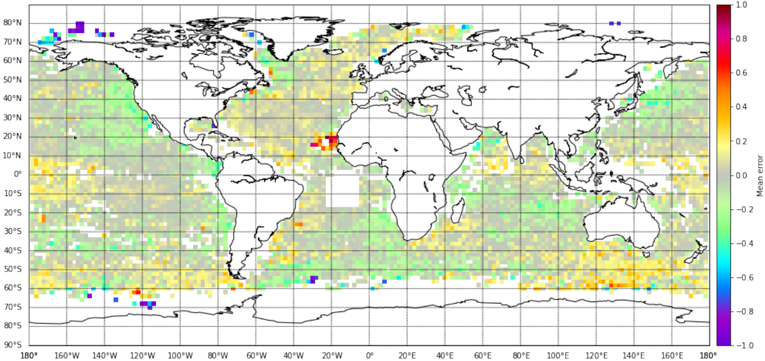
TensorFlow Keras NN Algorithm 2: Root Mean Squared Error



TensorFlow Keras NN Algorithm 2: Standard Deviation



TensorFlow Keras NN Algorithm 2: Mean Error

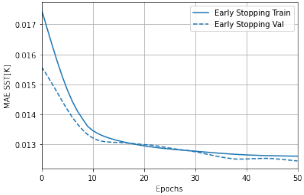


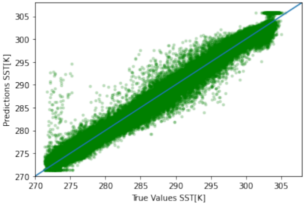
TensorFlow Keras NN Algorithm 2: Comments

This model had the smallest amount of features chosen for training (17) and the RandomizedSeachCV for this model had less iterations than the TF model that scored the best. In those ways, the model could easily be improved. However, the reduced amount of features and HPO iterations make this model computationally preferable, and the STD is only 0.07 K larger than the state-of-the-art linear regression based models.

TensorFlow Keras NN Algorithm 3

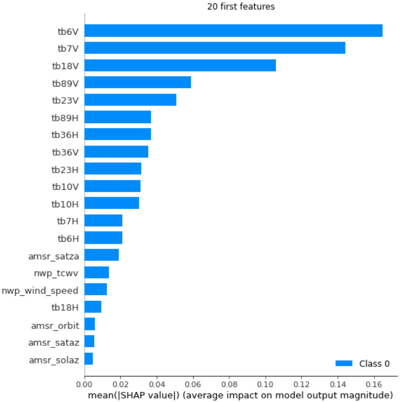
A number of epochs was assesed with early stopping to find the necessary epochs. A general model of 2 layers of 5 neurons, was trained first to find the main features of importance. In regards of the large dataset, a cheap model was prefered to train and validate, so as by means of the optimization of hyperparameter was executed over the Number of neural units in each two hidden layers, and optimizer. he training process was inspected through TensorBoard.





TensorFlow Keras NN Algorithm 3

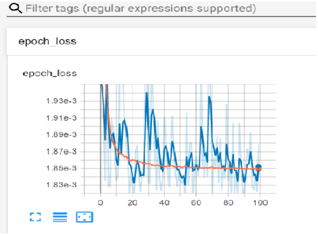
	Feat nro	mean[SHAP]	variable
0	12	0.164780	tb6V
1	26	0.144108	tb7V
2	16	0.105922	tb18V
3	22	0.059116	tb89V
4	18	0.050894	tb23V
5	23	0.037063	tb89H
6	21	0.036831	tb36H
7	20	0.035222	tb36V
8	19	0.031595	tb23H
9	14	0.031059	tb10V
10	15	0.030357	tb10H
11	27	0.021109	tb7H
12	13	0.021092	tb6H
13	4	0.019017	amsr_satza
14	10	0.013852	nwp_tcwv
15	9	0.012379	nwp_wind_speed
16	17	0.009598	tb18H
17	0	0.005839	amsr_orbit
18	6	0.005702	amsr_sataz
19	5	0.004813	amsr_solaz

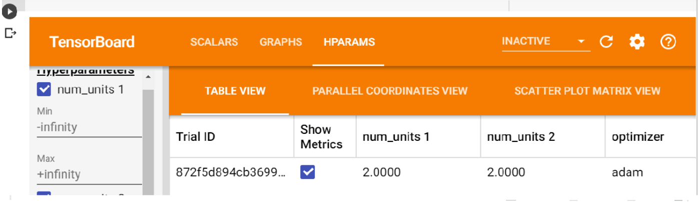


TensorFlow Keras NN Algorithm 3

Given the discrete values of [2,5,10] for the 2 dense layers, and optimizers: adam and sgd. Results:(num units 1: 2), (num units 2: 2), (optimizer: adam).

```
session_num = 0
for num_units1 in HP_NUM_UNITS1.domain.values:
  for num_units2 in HP_NUM_UNITS2.domain.values:
    for optimizer in HP_OPTIMIZER.domain.values:
      hparams = {
          HP_NUM_UNITS1: num_units1,
          HP_NUM_UNITS2: num_units2,
          HP_OPTIMIZER: optimizer,
      }
      run_name = "run-%d" % session_num
      print('--- Starting trial: %s' % run_name)
      print({h.name: hparams[h] for h in hparams})
      run('logs/hparam_tuning/' + run_name, hparams)
      session_num += 1
```





Trial ID	Show Metrics	num_units 1	num_units 2	optimizer
872f5d894cb3699...	☑	2.0000	2.0000	adam

Model performances: summary

Retrieval of sea surface temperatures (SSTs) from passive microwave measurements

Ann-Sofie, Emy, Marta & Yanet

Appendix

	XGBoost	TFKeras 1	TFKeras 2	TFKeras 3
R^2	0.999	0.998	0.997	0.992
MSE [K^2]	0.118	0.202	0.273	0.680
MAE [K]	0.215	0.330	0.375	0.561
Mean error [K]	0.007	-0.076	0.002	-0.003
Std [K]	0.344	0.442	0.522	0.820