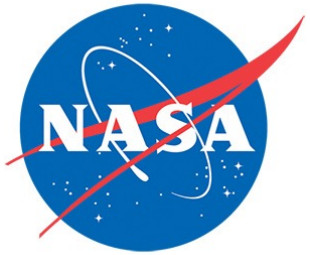
Shape-based Feature Engineering for Solar Eruption Forecasting

Varad Deshmukh

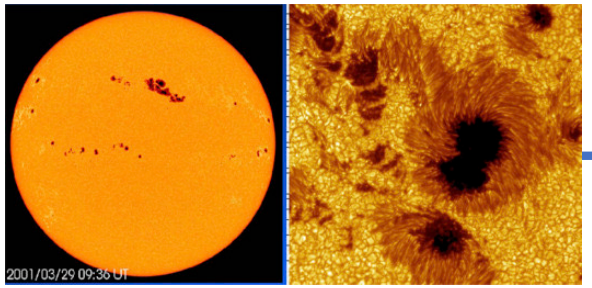
University of Colorado Boulder



Outline

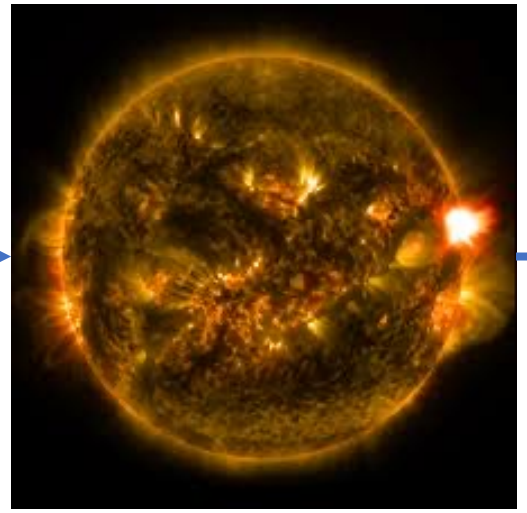
- Problem statement
- ML-based flare forecasting pipleine
- Shape-based feature engineering
- Multi-layer Perceptron Model Design and Evaluation
- Results
- Conclusions
- Resources

Magnetic Solar Eruptions



Release of Magnetic Energy on Active Regions (Complex Magnetic Field Structures)

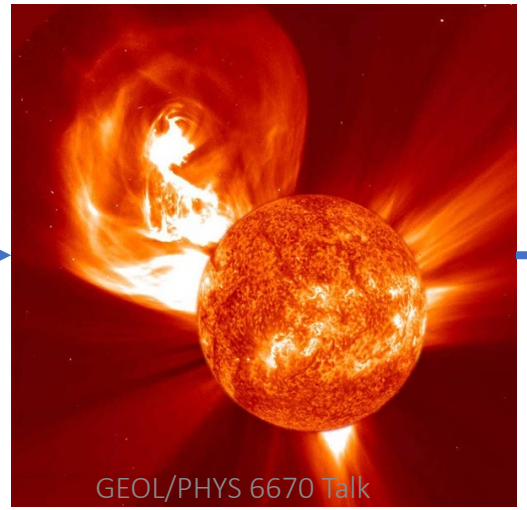
Solar Flares



Flare classification based on magnitude

A,B,C (Weak flares)
M,X (Strong flares)

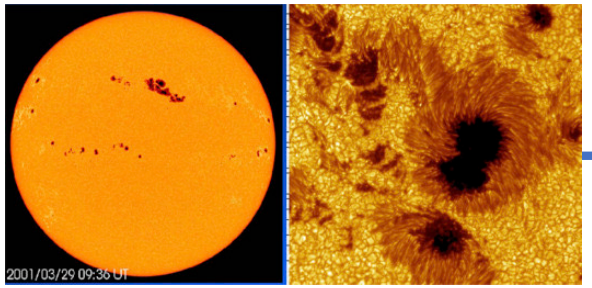
Coronal Mass Ejection



Impact of CMEs on Earth

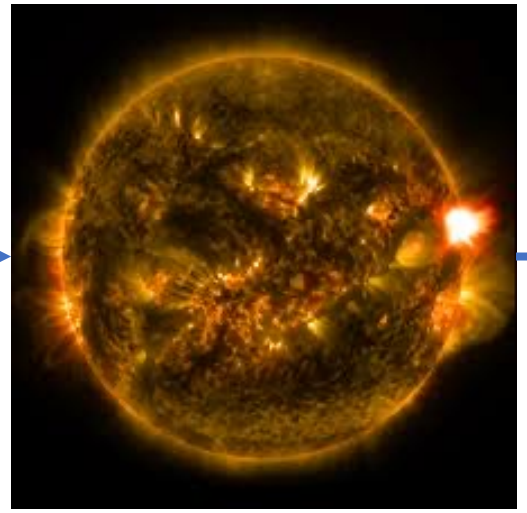
GPS scintillations,
Radiation Hazards,
Power Grid Failures,
Spacecraft damages,
….

Magnetic Solar Eruptions



Release of Magnetic Energy on Active Regions (Complex Magnetic Field Structures)

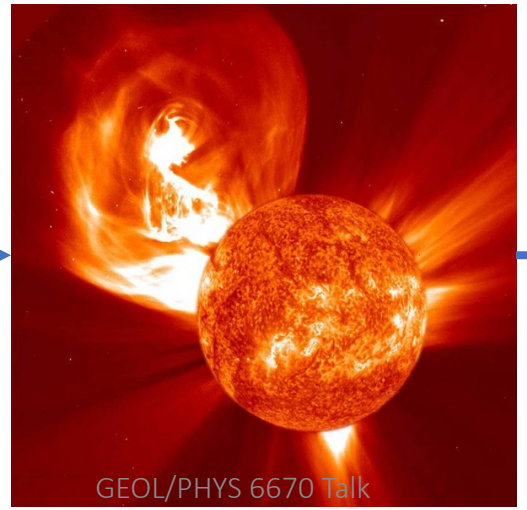
Solar Flares



Flare classification based on magnitude

A,B,C (Weak flares)
M,X (Strong flares)

Coronal Mass Ejection



Impact of CMEs on Earth

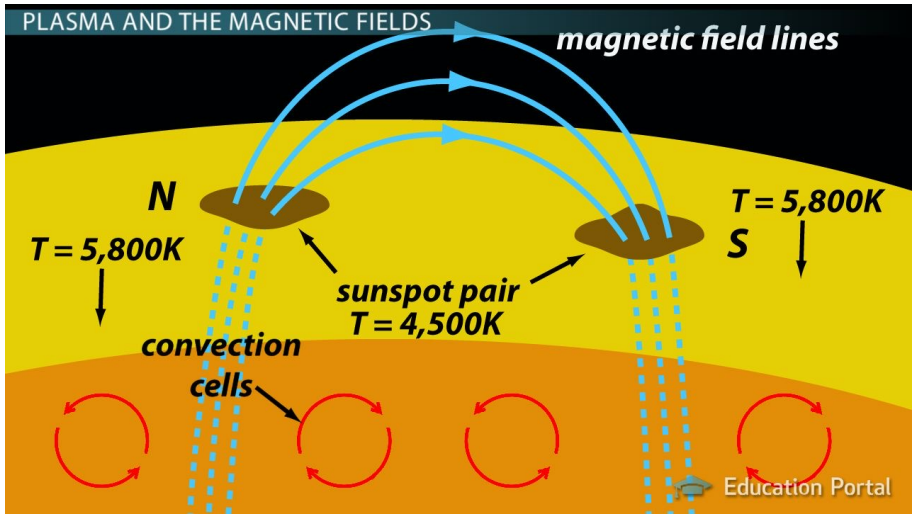
GPS scintillations,
Radiation Hazards,
Power Grid Failures,
Spacecraft damages,
….

Trillions of dollars in economic losses!

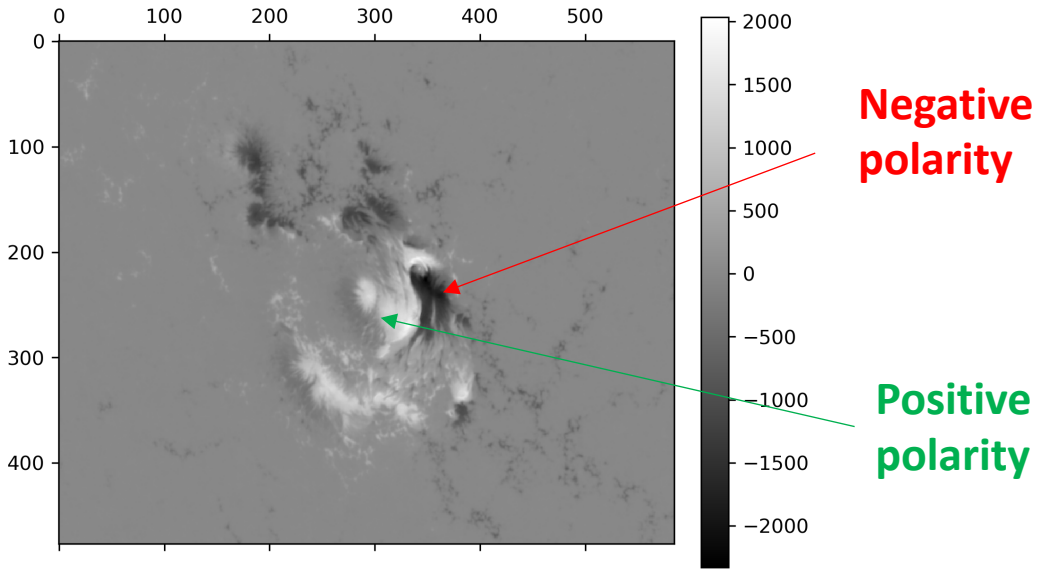
Active Regions and Magnetic Reconnections

- Solar Active Regions (AR) are hot-beds for solar eruptions.
- Magnetic field is aligned vertically to the surface, causing dark formations we call *sunspots*.
- A complex AR with sheared and interspersed polarities can cause eruptions.

Cartoon Model of an AR

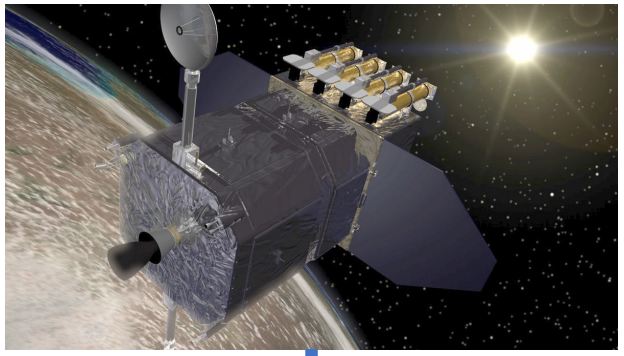


Top View 2D Measurement (Magnetogram)



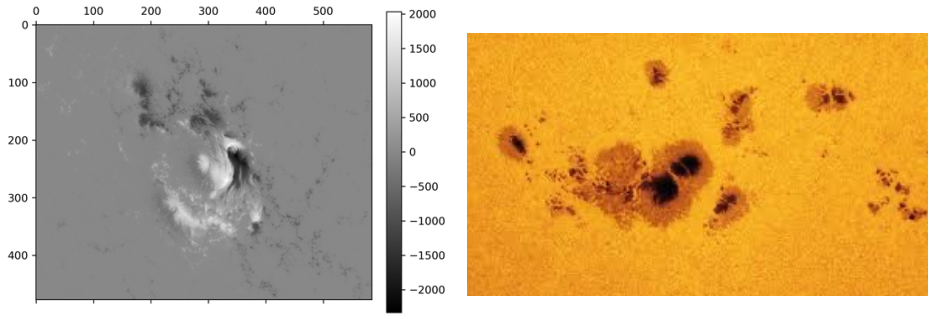
Observing Active Regions

Solar Dynamics Observatory

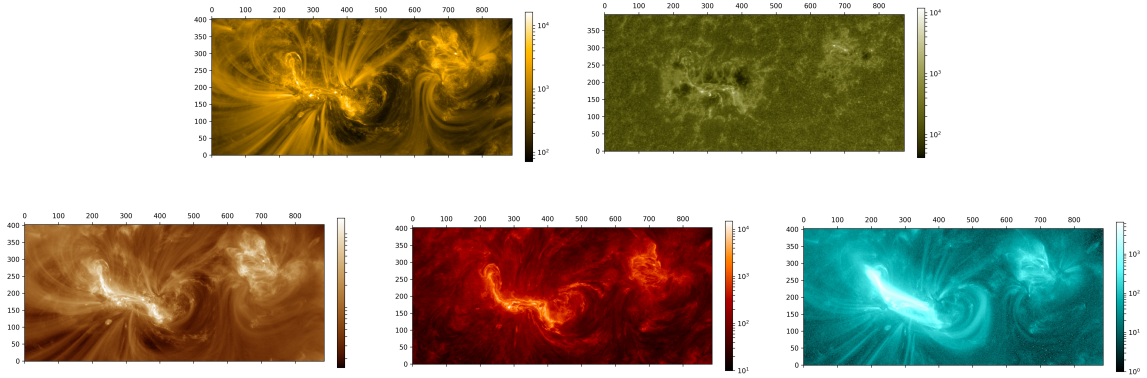


Instrument:
Helioseismic and Magnetic Imager

Instrument:
Atmospheric Image Assembly



Continuum and Photospheric Magnetic Images



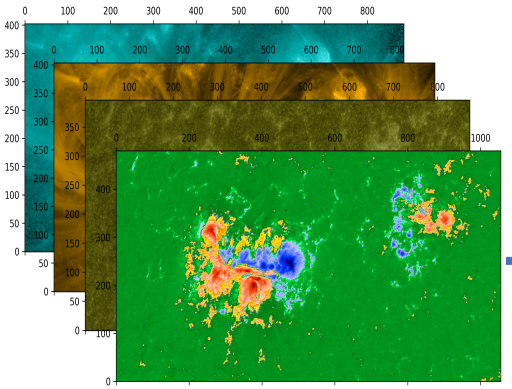
Coronal/Chromospheric Images

ML-based Flare Forecasting

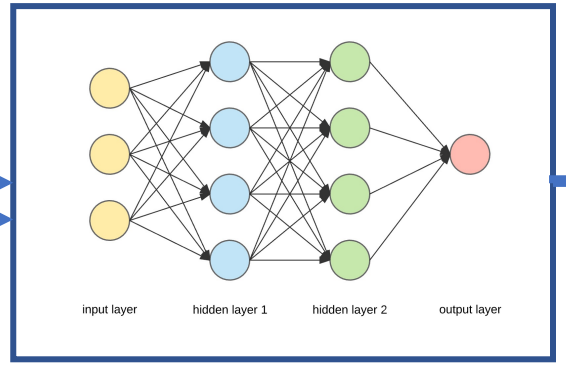
Can we use solar active region observations or attributes to determine the probability of a flare occurrence in the near future?

ML-based Flare Forecasting

Can we use solar active region observations or attributes to determine the probability of a flare occurrence in the near future?



Featurization



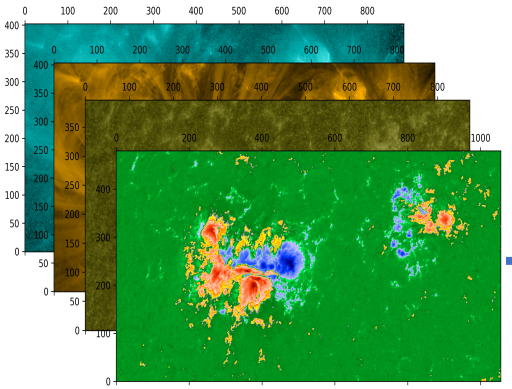
Categorical:
Flare/No-flare in next k hours

Probabilistic:
Prob(flare) in next k hours

Regression:
Flare Magnitude in next k hours

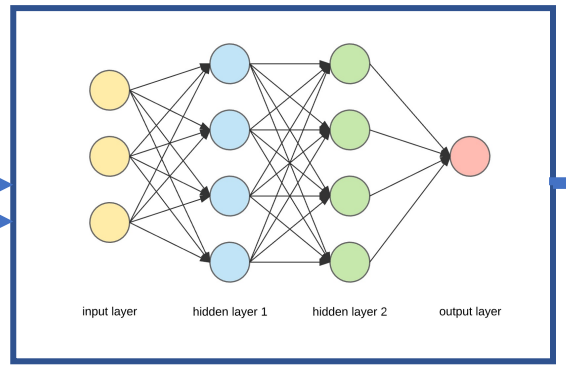
ML-based Flare Forecasting

Can we use solar active region observations or attributes to determine the probability of a flare occurrence in the near future?



Typically physics-based

Featurization



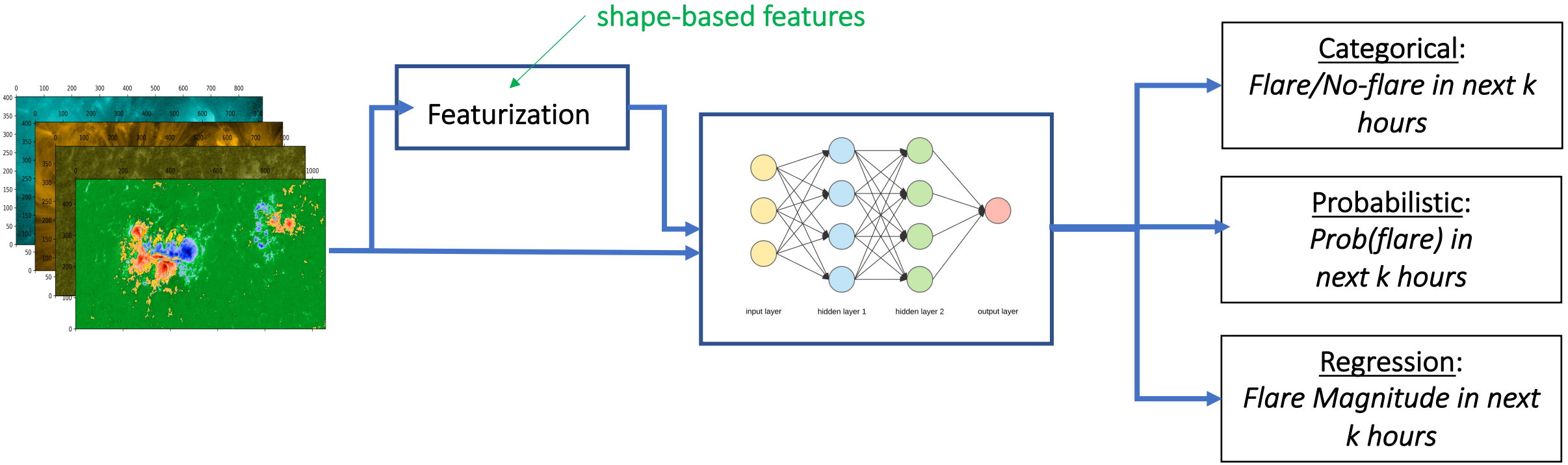
Categorical:
Flare/No-flare in next k hours

Probabilistic:
Prob(flare) in next k hours

Regression:
Flare Magnitude in next k hours

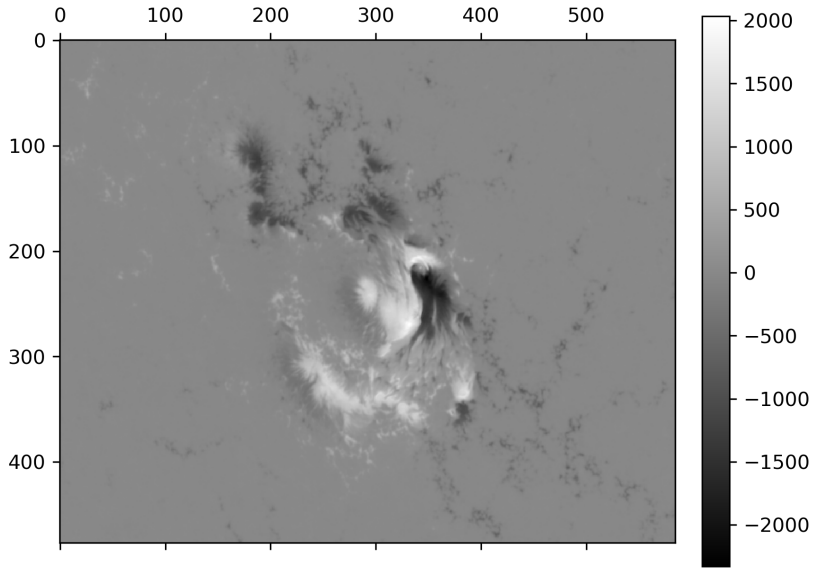
Can the “shape” be a useful indicator?

We propose a feature engineering approach based on formalizing the shape of the radial magnetic flux for flare prediction using Topological Data Analysis and Computational Geometry.



Dataset Images

We choose vector magnetic field AR observations from 2010 to 2016 at a cadence of 1 hour, only selecting the radial magnetic field component for feature extraction.



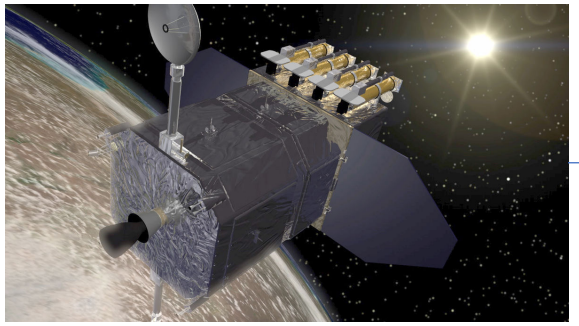
3691 ARs

141 ARs flared at least once

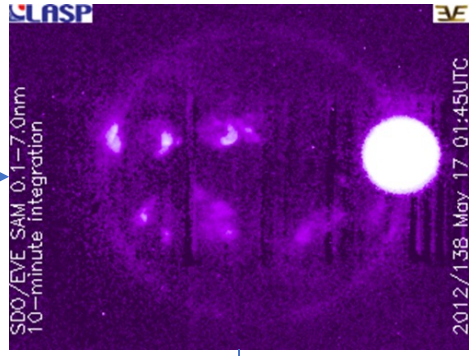
438,539 total images

Dataset Labels

Geostationary Operational Environmental Satellite



X-Ray Observations



Active Region Details

Flare Magnitude (X,M,C,B,A)

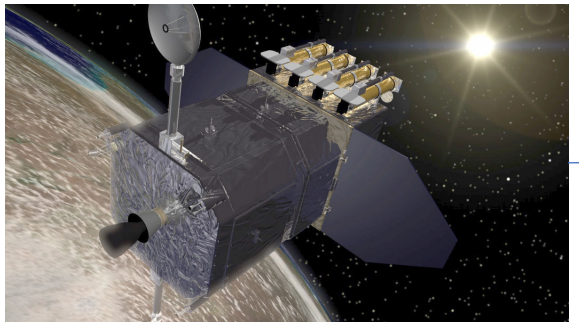
Flare Location (AR #)

Flare onset and duration (UT)

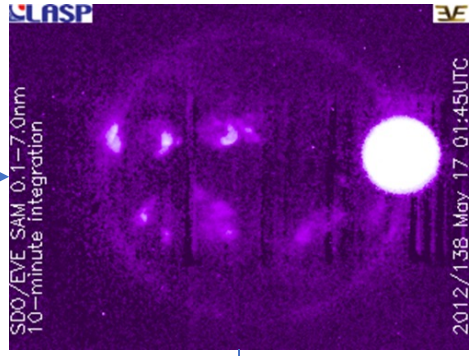
Did an active region produce an M1.0+ class flare in the next *k* hours?

Dataset Labels

Geostationary Operational Environmental Satellite



X-Ray Observations



Active Region Details

Flare Magnitude (X,M,C,B,A)

Flare Location (AR #)

Flare onset and duration (UT)

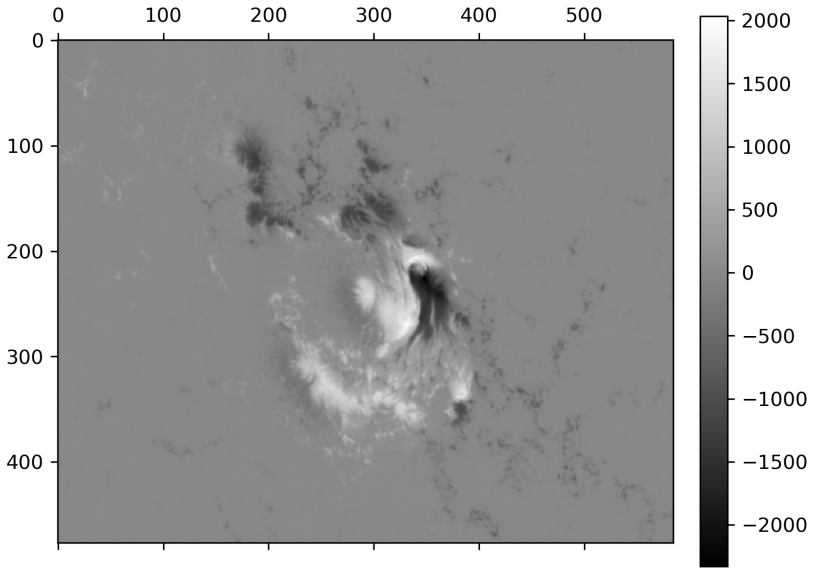
Did an active region produce an M1.0+ class flare in the next *k* hours?

Labels: {0,1}

Dataset Images

We choose vector magnetic field AR observations from 2010 to 2016 at a cadence of 1 hour, only selecting the radial magnetic field component for feature extraction.

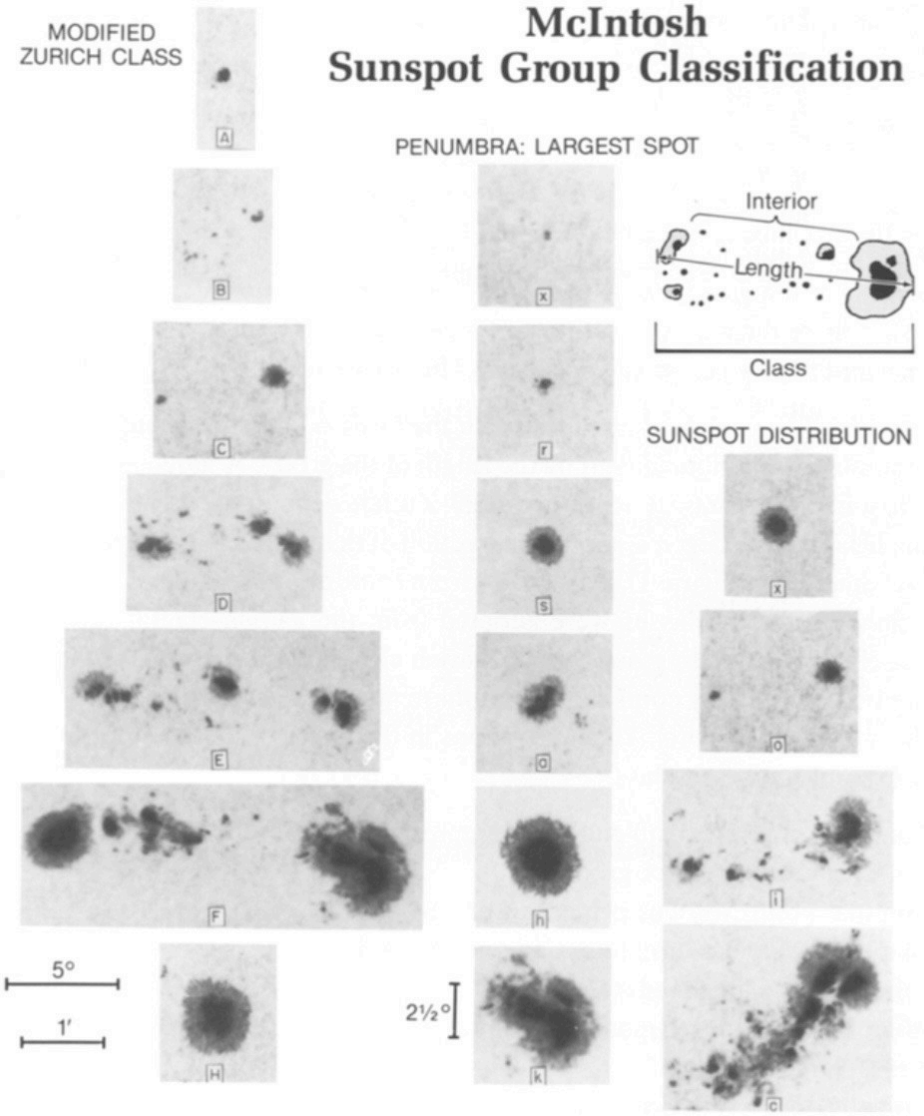
3691 ARs
141 ARs flared at least once
438,539 total images



432,821 non-flaring images
5538 flaring images

Imbalance ratio: 78:1!

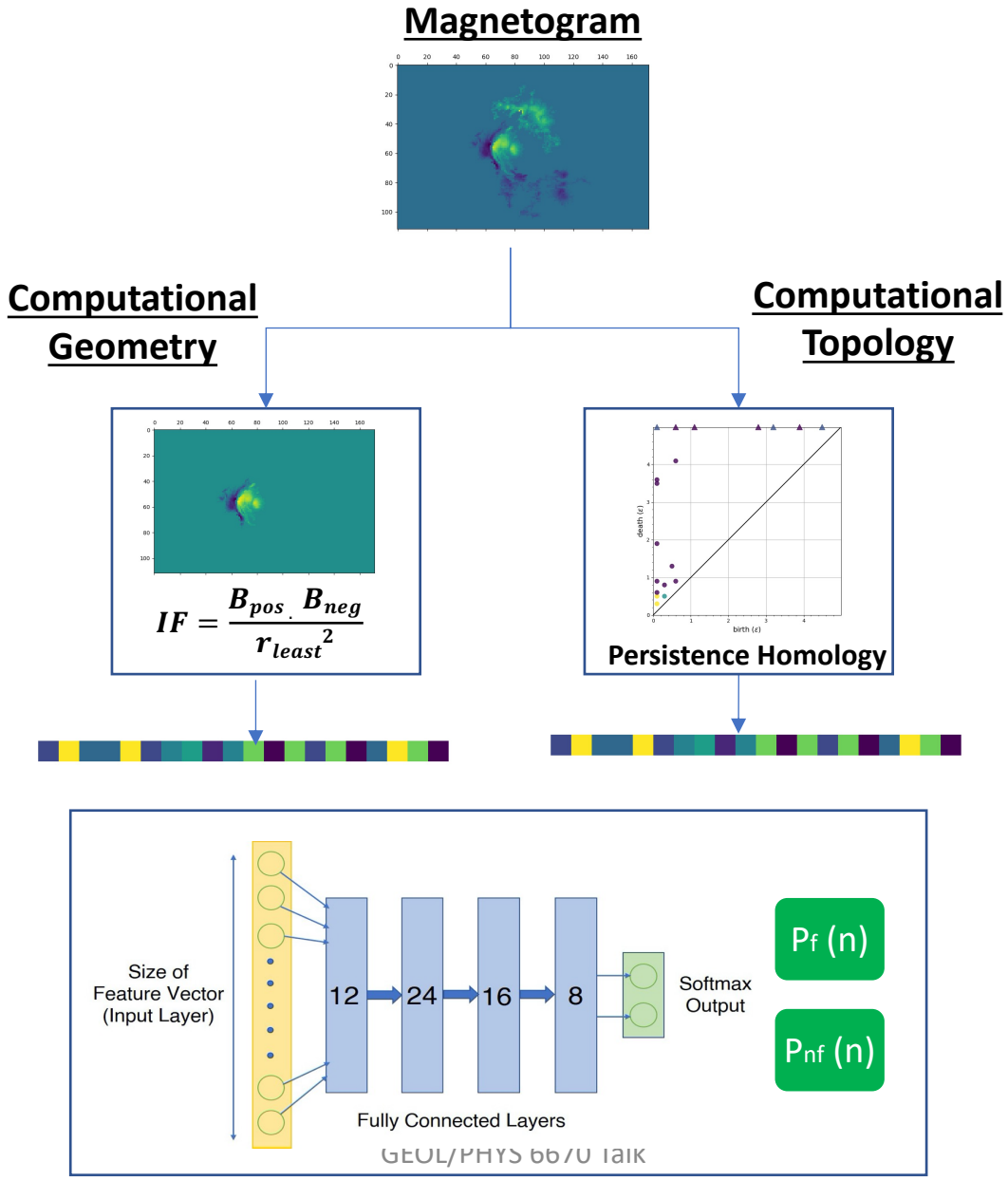
Motivation for Shape-based Featurization



- Categorize sunspots into multiple classes based on the shape as observed in the continuum observations.
- Compute the historical flaring rate for each class, which is the flaring probability.

Shape-based Approach

Can we quantify the interaction of polarity structures in Active Regions?

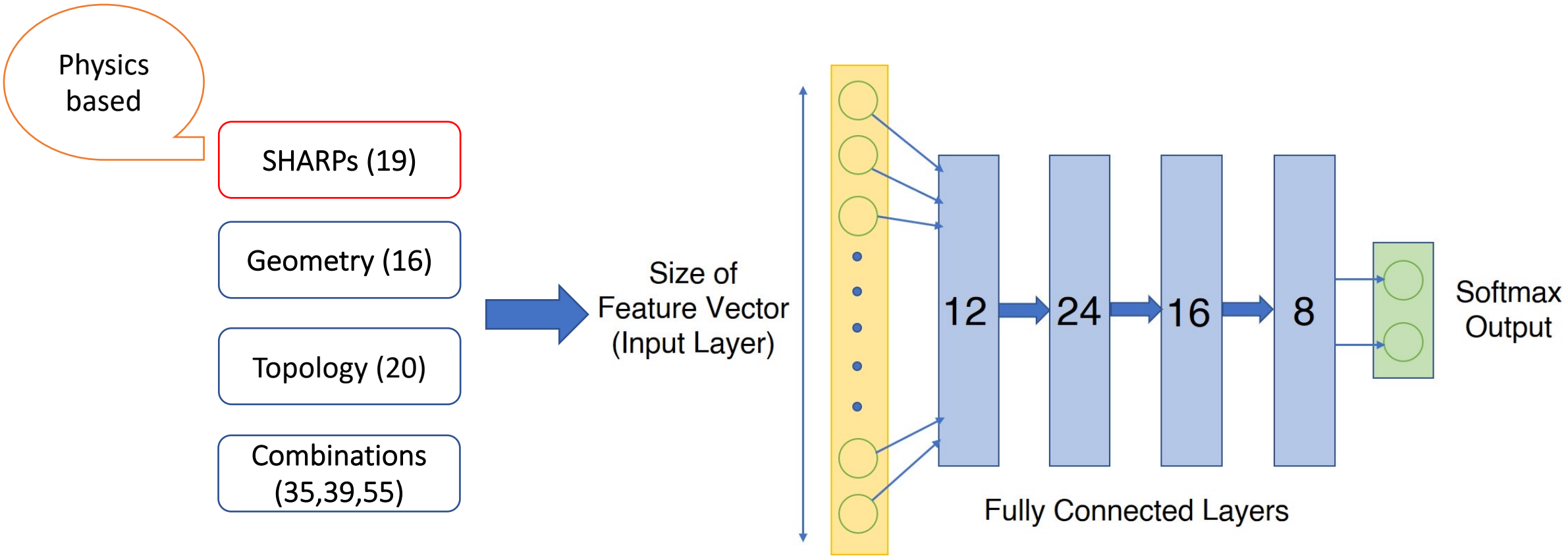


Can we quantify the "landscape" of a magnetogram based on the arrangement of "hills" and "valleys" of the radial magnetic flux?

Baseline Feature Set: 19 SHARPs Features

Acronym	Description	Units
LAT_FWT	Latitude of the flux-weighted center of active pixels	degrees
LON_FWT	Longitude of the flux-weighted center of active pixels	degrees
AREA_ACR	Line-of-sight field active pixel area	micro hemispheres
USFLUX	Total unsigned flux	Mx
MEANGAM	Mean inclination angle, gamma	$degrees$
MEANGBT	Mean value of the total field gradient	G/Mm
MEANGBZ	Mean value of the vertical field gradient	G/Mm
MEANGBH	Mean value of the horizontal field gradient	G/Mm
MEANJZD	Mean vertical current density	mA/m^2
TOTUSJZ	Total unsigned vertical current	A
MEANALP	Total twist parameter, alpha	$1/Mm$
MEANJZH	Mean current helicity	G^2/m
TOTUSJH	Total unsigned current helicity	G^2/m
ABSNJZH	Absolute value of the net current helicity	G^2/m
SAVNCPP	Sum of the absolute value of the net currents per polarity	A
MEANPOT	Mean photospheric excess magnetic energy density	$ergs/cm^3$
TOTPOT	Total photospheric magnetic energy density	$ergs/cm^3$
MEANSHR	Mean shear angle (measured using B_{total})	$degrees$
SHRGT45	Percentage of pixels with a mean shear angle greater than 45 degrees	$percent$

Features and Model

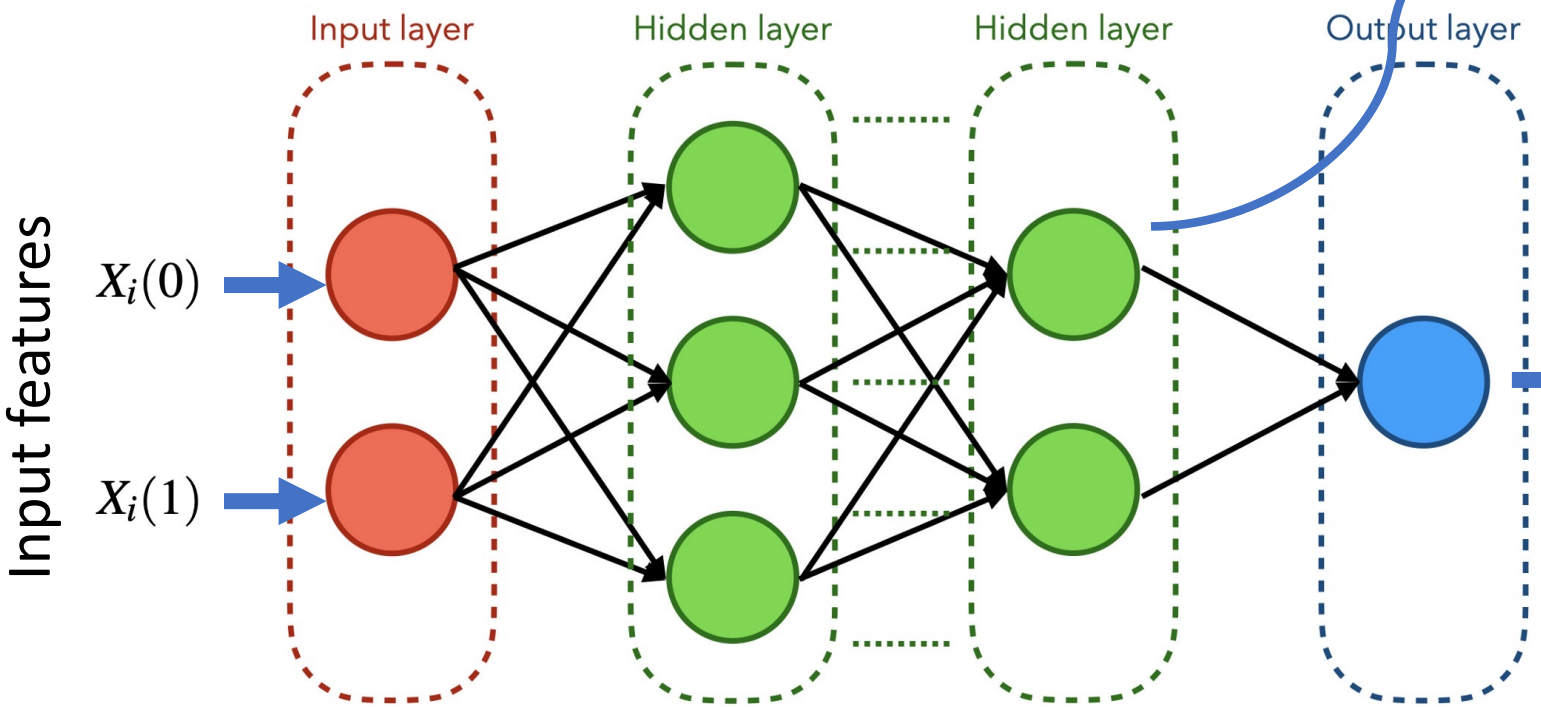


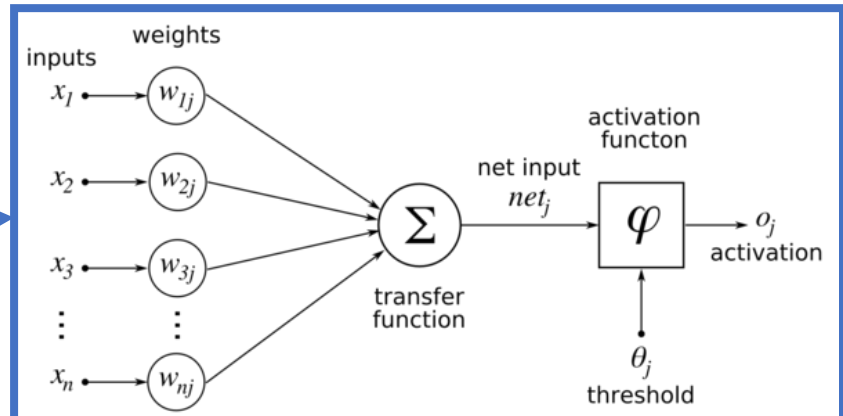
Feature Sets

Multilayer Perceptron Model

Neural Networks

$$D = \{(X_1, y_1), (X_2, y_2), \cdots\cdots, (X_n, y_n)\}$$





Evaluation of each neuron

$$o_j = \phi\left(\sum_{i=1}^{n} w_{ij} x_j\right)$$

Model Prediction

Example: ReLU

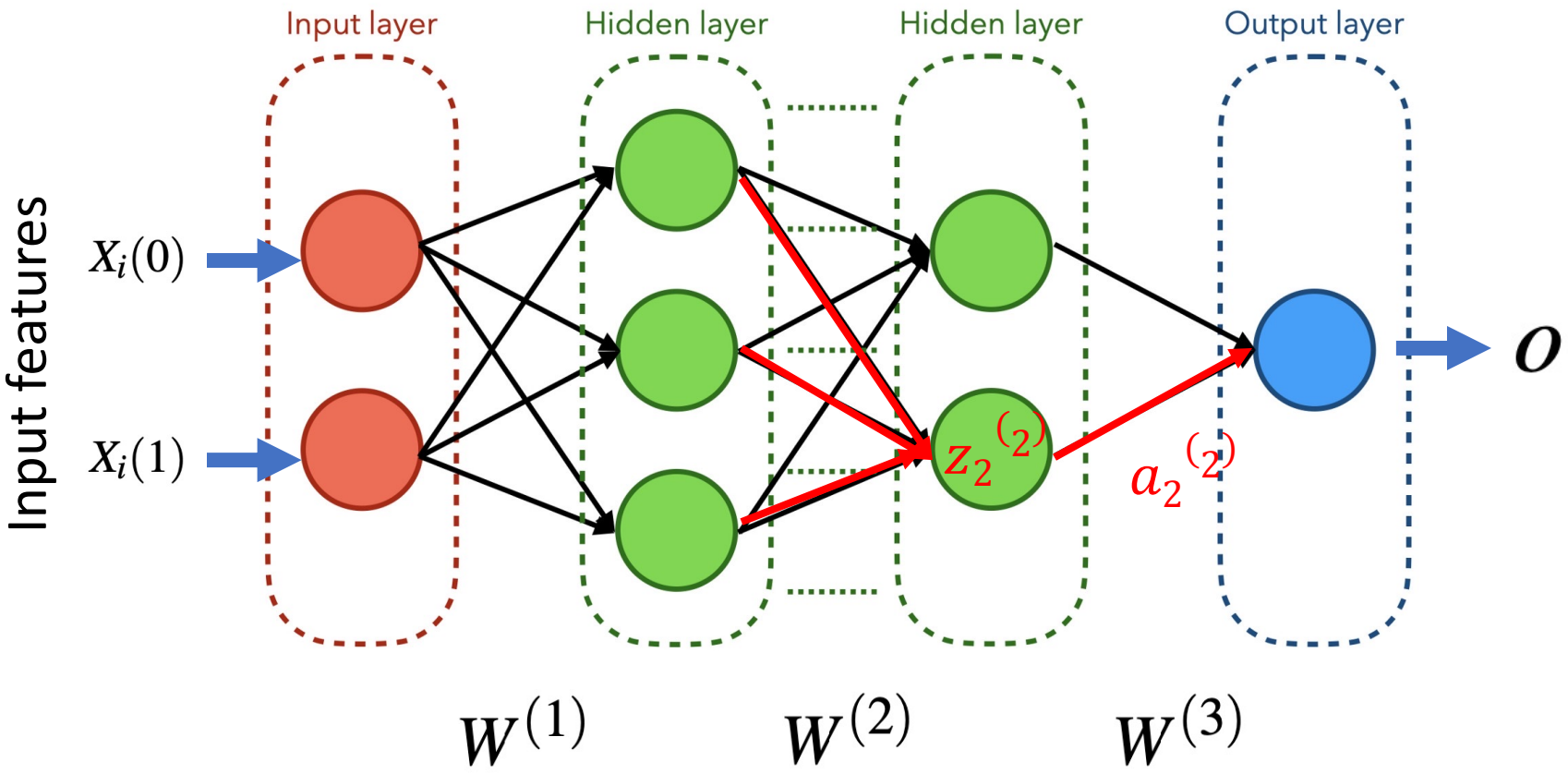
$$f(x) = x^+ = \max(0, x)$$

Introducing non-linearity: Activation Functions

Source: https://mlfromscratch.com/neural-networks-explained/

Evaluating the model: Forward Pass

$$z^{(1)} = W^{(1)}x, \quad z^{(2)} = W^{(2)}a^{(1)},$$
$$a^{(1)} = f(z^{(1)}) \quad a^{(2)} = f(z^{(2)}) \qquad o = W^{(3)}a^{(2)}$$



Each edge in layer l between nodes j and k is given by $w_{jk}^{(l)}$.

$$W^{(l)} = \{w_{jk}^{(l)}\}$$

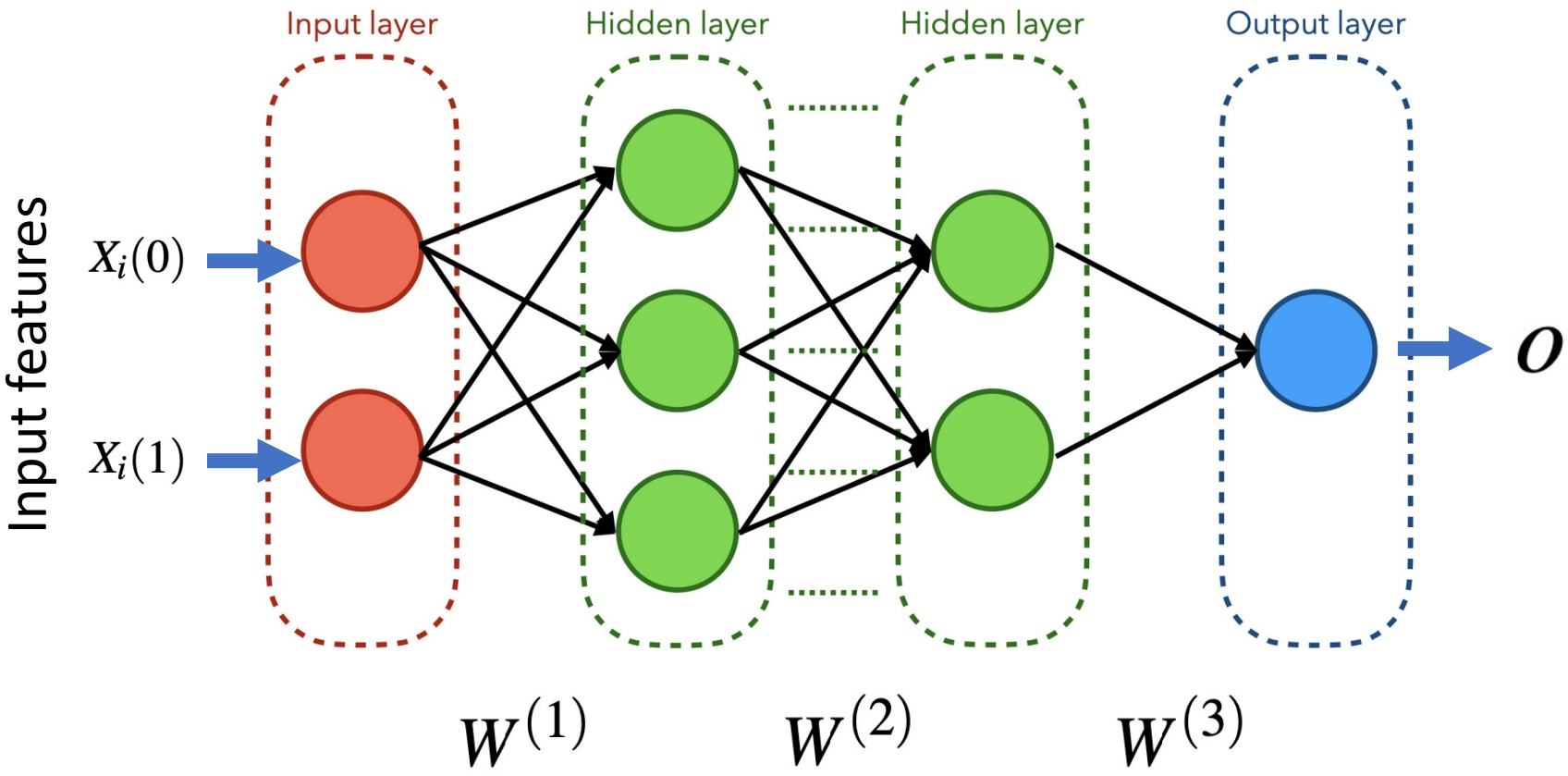
Activation of the second neuron in the second layer is given by:

$$z_2^{(2)} = \sum_{j=1}^{3} w_{j2}^{(2)}$$

$$a_2^{(2)} = f(z_2^{(2)})$$

Training the Model: Backpropagation

$$D = \{(X_1, y_1), (X_2, y_2), \cdots\cdots, (X_n, y_n)\}$$



Compute the loss with the ground truth

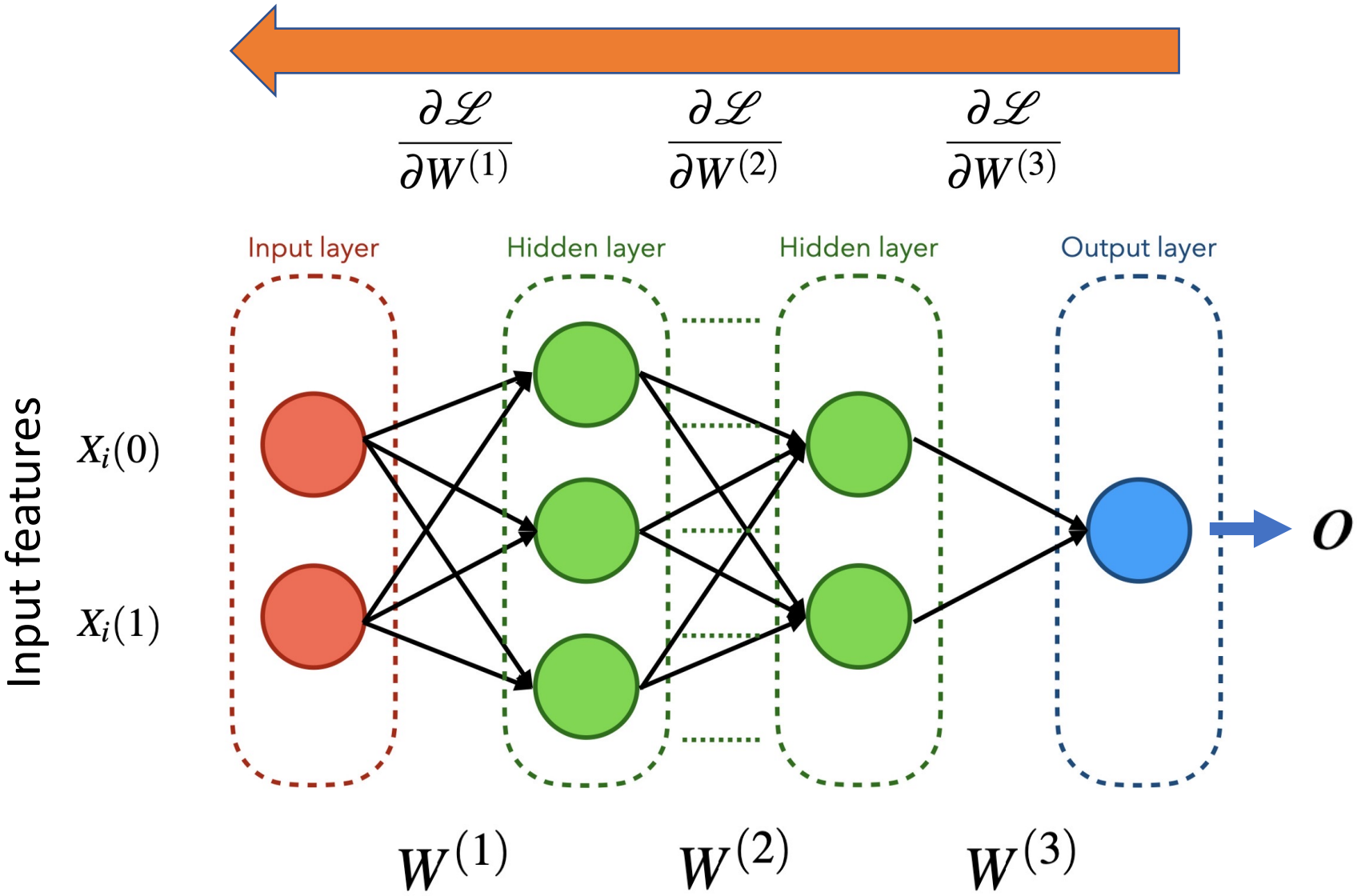
$$\mathscr{L}(y_i, o) = ||y_i - o||^2$$

Use gradient descent to update the weights

$$w_{jk}^{(l)} = w_{jk}^{(l)} - \alpha \frac{\partial \mathscr{L}}{\partial w_{jk}^{(l)}}$$

Learning rate

Updating the Weights: Chain Rule



Compute the loss with the ground truth

$$\mathcal{L}(y_i, o) = ||y_i - o||^2$$

Use gradient descent to update the weights

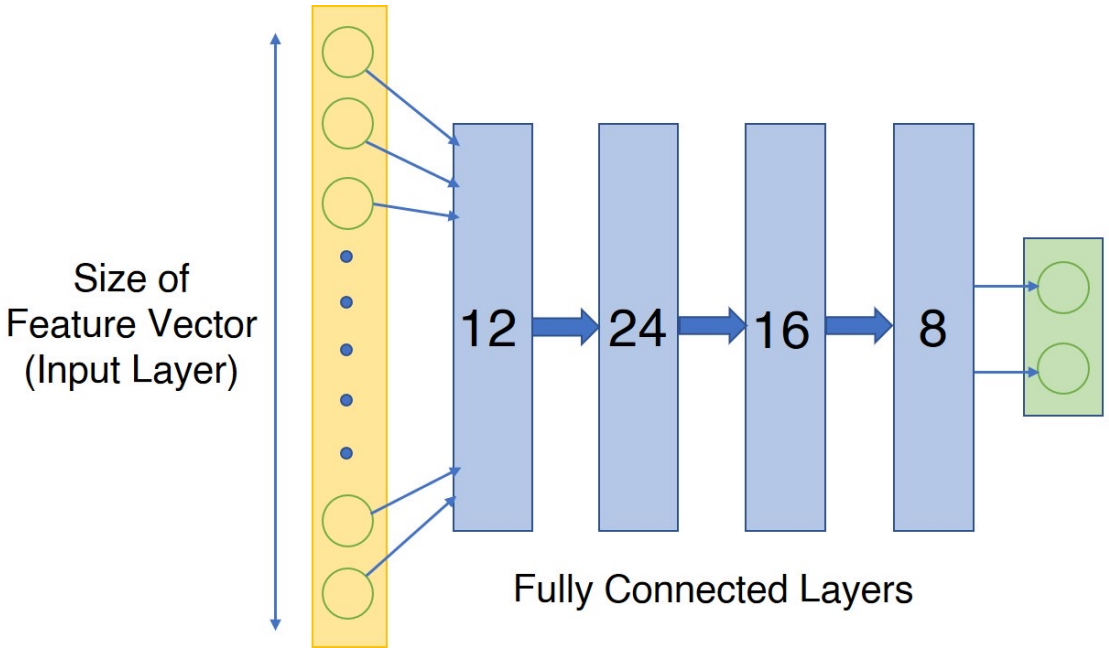
$$w_{jk}^{(l)} = w_{jk}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}}$$

Updating the Weights: Chain Rule

- Some useful resources for additional reading:
  - https://mlfromscratch.com/neural-networks-explained/
  - https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

- Videos explaining backpropagation from 3B1B:
  - https://www.youtube.com/watch?v=Ilg3gGewQ5U
  - https://www.youtube.com/watch?v=tIeHLnjs5U8

Some model specifics



The softmax function converts the output to probabilities (summing to 1).

$$p(flare) = \frac{e^{o_{flare}}}{e^{o_{flare}} + e^{o_{no-flare}}}$$

$$p(no-flare) = \frac{e^{o_{no-flare}}}{e^{o_{flare}} + e^{o_{no-flare}}}$$

Use a weighted binary cross-entropy loss function.

$$\mathcal{L}(y_i, p) = -w_{flare} \times y_i \times log(p(flare)) - w_{no\ \ flare} \times (1 - y_i) \times log(p(no\_flare)) + \beta ||w_{jk}^{(l)}||^2$$

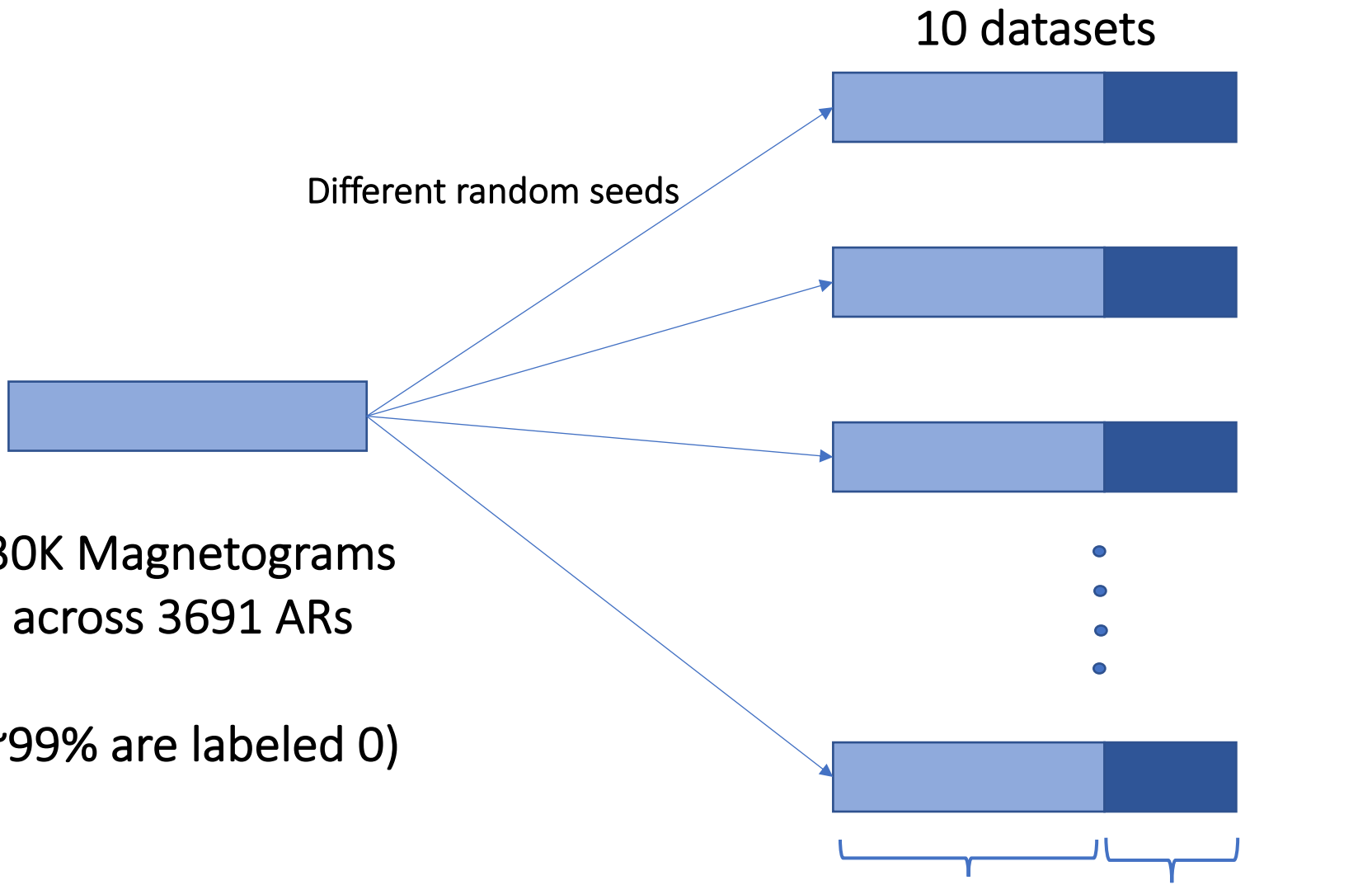
Regularization

Some algorithm specifics: Gradient Descent

Algorithm	Description	Loss computed
Gradient Descent	Compute the loss on the entire dataset, then backprop.	$\mathscr{L} = \sum_{i=1}^{N} \|\|y_i - o_i\|\|^2$
Stochastic Gradient Descent	Compute the loss on a single data point, then backprop.	$\mathscr{L} = \|\|y_i - o_i\|\|^2$
Mini-batch Gradient Descent	Compute the loss on a randomly sampled batch of input data, then backprop.	$\mathscr{L} = \sum_{i=1}^{m} \|\|y_i - o_i\|\|^2$

- Gradient descent is computationally slow if the dataset is huge and may converge to local minimum for non-convex loss.
- Stochastic Gradient Descent may converge faster because of frequent weight updates, and possibly converge to a global maximum/better local minimum, but computationally slow.
- Mini-batch SGD provides a compromise by providing benefits of SGD, but can be vectorized to run fast.

Dataset Splitting into Training-Testing Sets



Performance Metrics: Contingency Table

Categorical forecasting:

Flaring if p(flare) > 0.5, else non-flaring

Observation	Prediction	
	Flare	No flare
Flare	True Positive (TP)	False Negative (FN)
No flare	False Positive (FP)	True Negative (TN)

What metrics do we use?

True Skill Statistic

$$TSS = \frac{TP}{TP + FN} - \frac{FP}{FP + TN}$$

Metric	Formula
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
Heidke Skill Score (HSS)	$\frac{TP\times TN-FP\times FN}{(TP+FP)(FP+TN)+(TP+FN)(FN+TN)}$
Frequency Bias (FB)	$\frac{TP+FP}{TP+FN}$

- Useful for evaluating imbalanced datasets.
- Score Range [-1,1]
- Reports a score of 0 for random or always no-flare forecast.

Model Hyperparameters

Hyperparameters: Pre-determined model settings that determine how the model is trained.

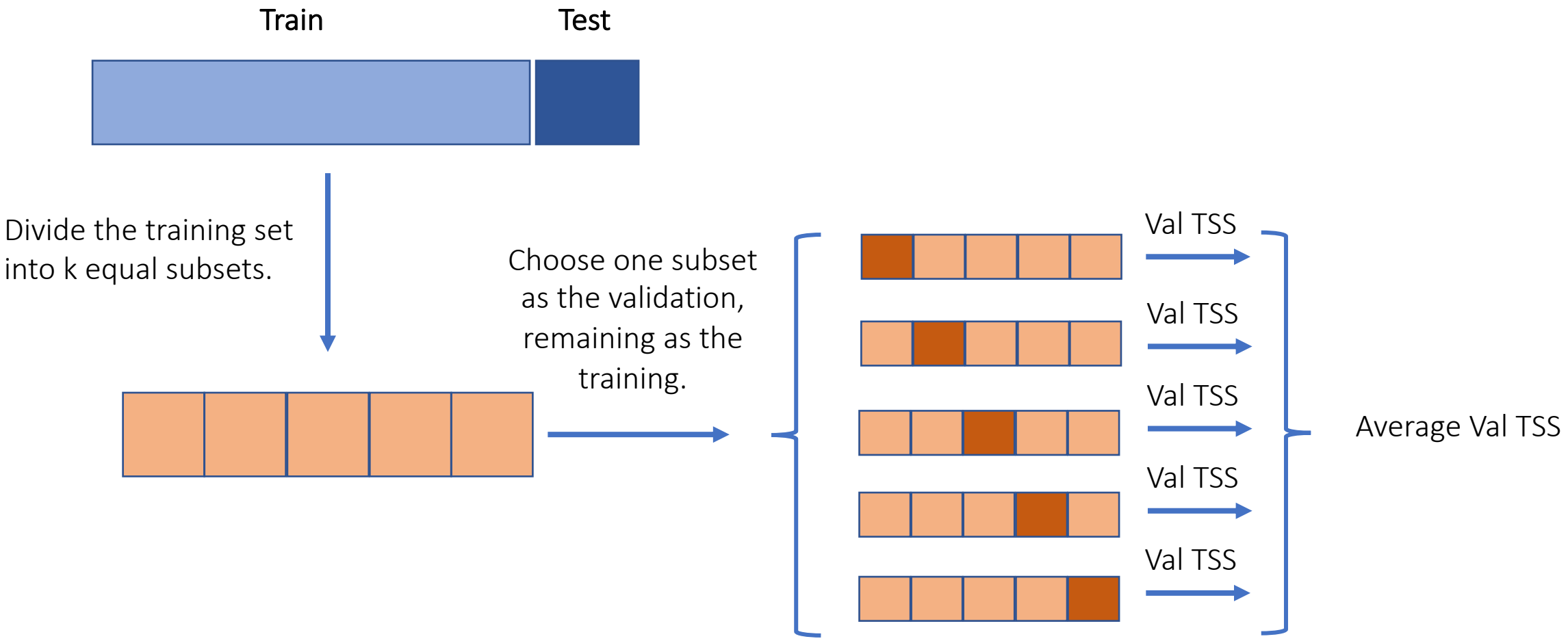
Manually set hyperparameters	Settings
Total hidden layers	4
Hidden layer activation function	ReLU
Output layer activation function	Sotfmax
Training batch size	128
Loss function	Weighted Binary Cross Entropy
Optimization method	Adagrad

Predetermined hyperparameters using a trial-and-error approach

Automatically tuned hyperparameters
L_2 regularization decay
Learning rate
Rate decay
Binary cross-entropy weights

Automatically tuned hyperparameter values using k-fold cross-validation.

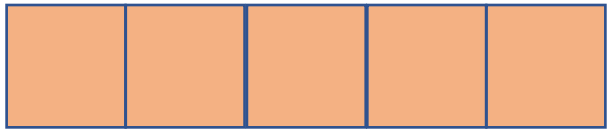
Hyperparameter Tuning: k-fold cross validation



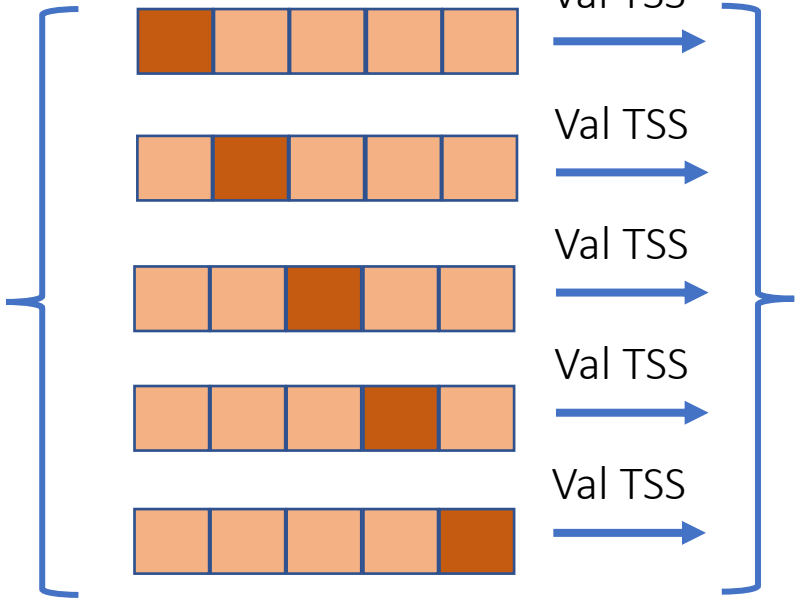
Hyperparameter Tuning: k-fold cross validation

1. Repeat for different hyperparameter values.
2. Choose one with the highest Val TSS.
3. Use on the test set.

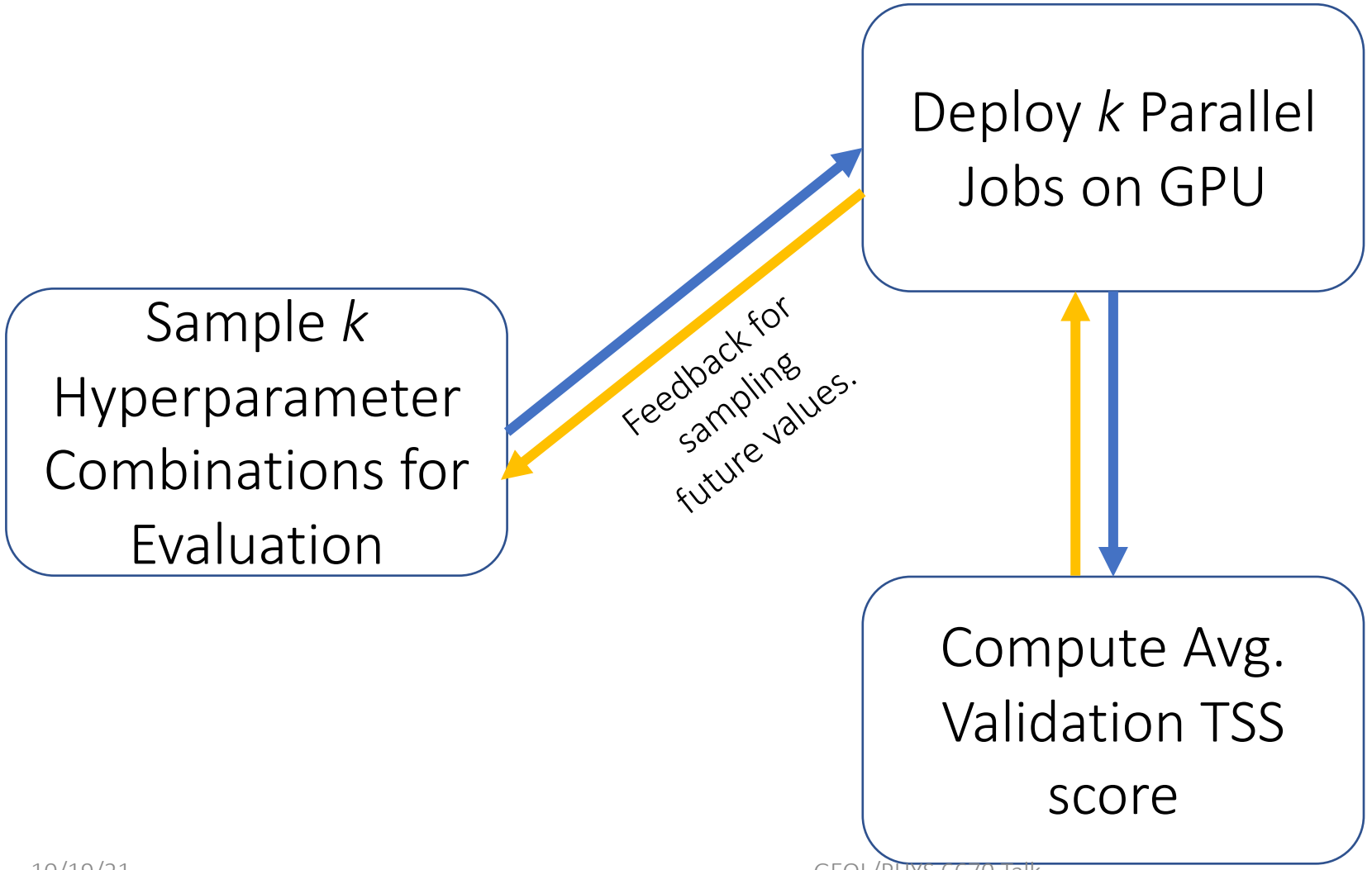
Divide the training set into k equal subsets.



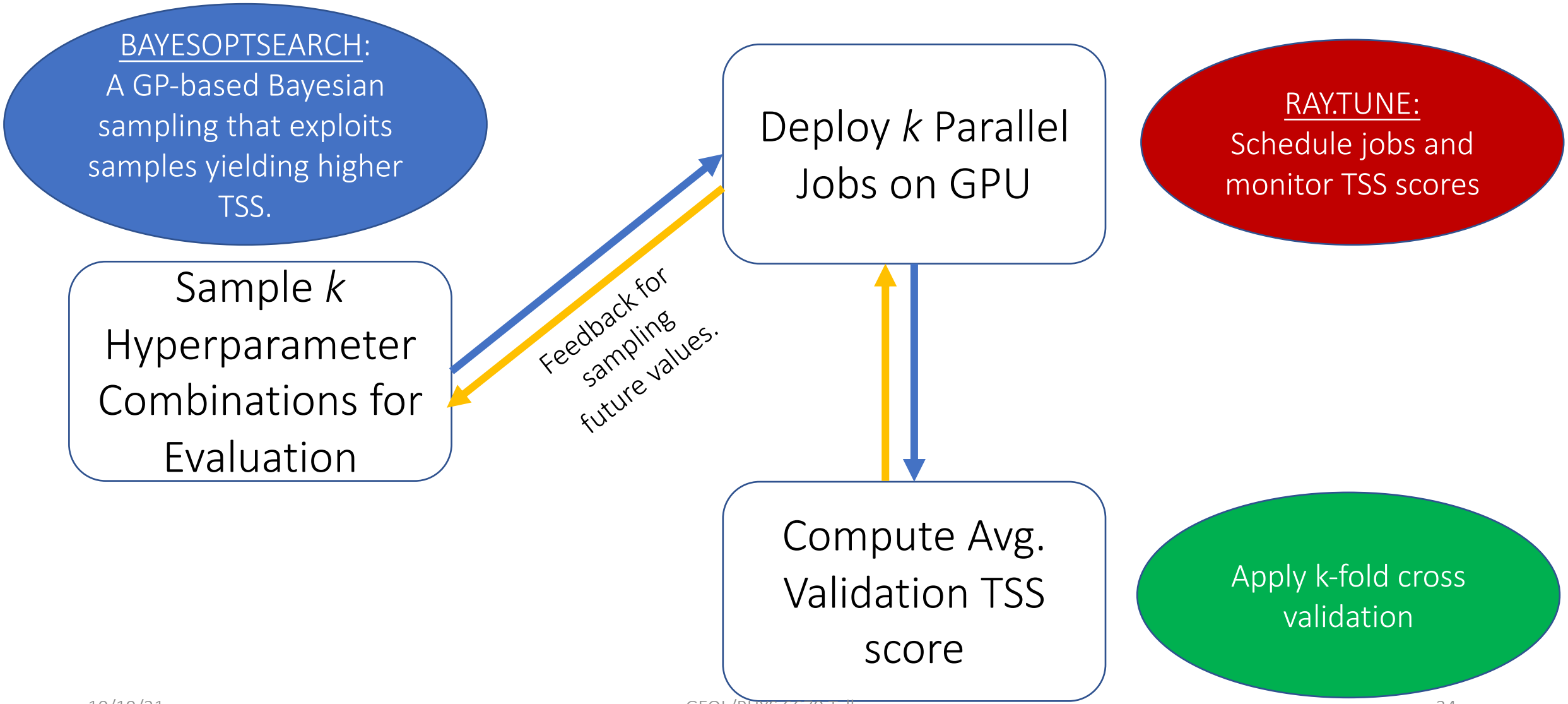
Choose one subset as the validation, remaining as the training.



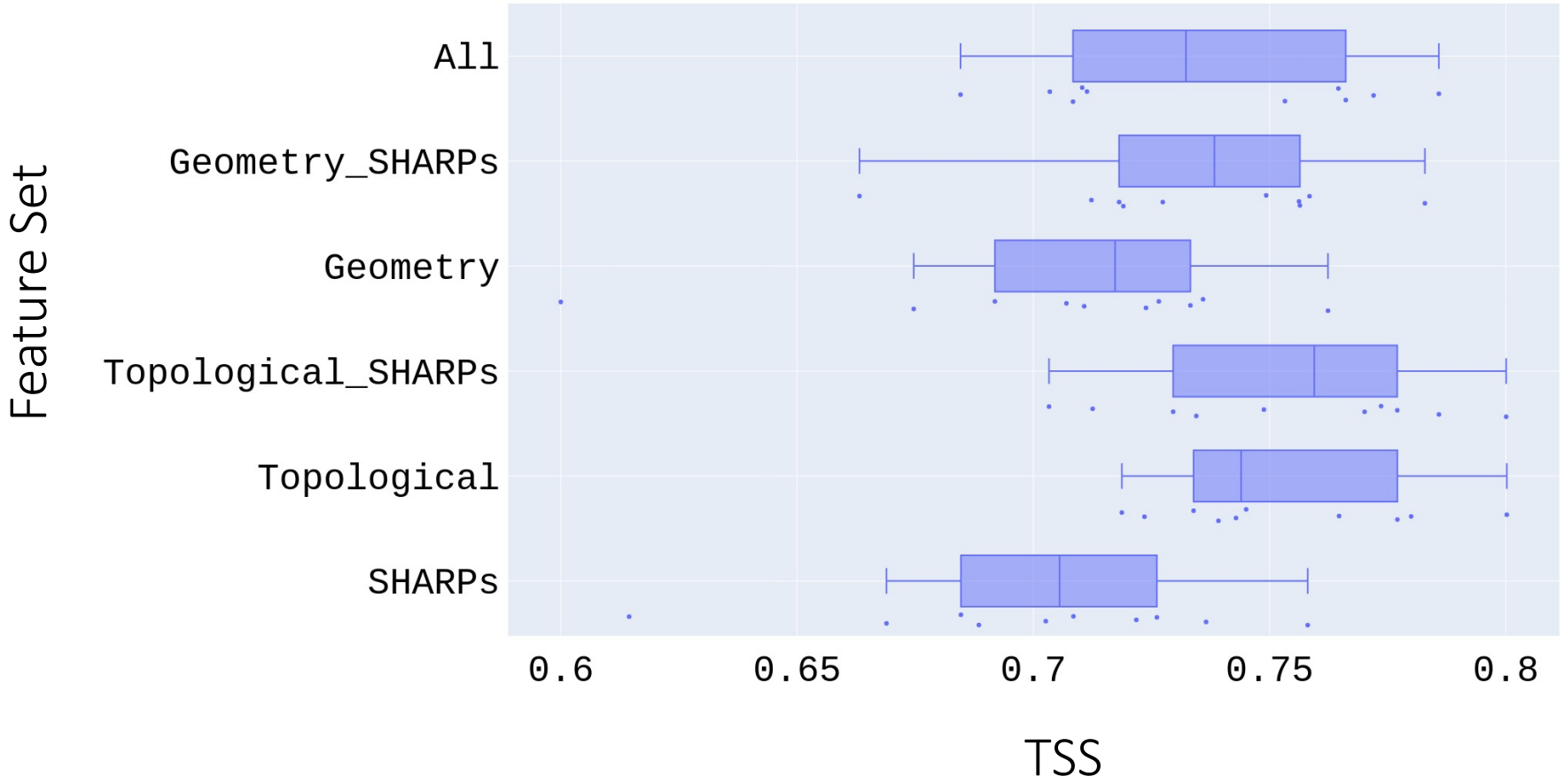
Hyperparameter Tuning Workflow



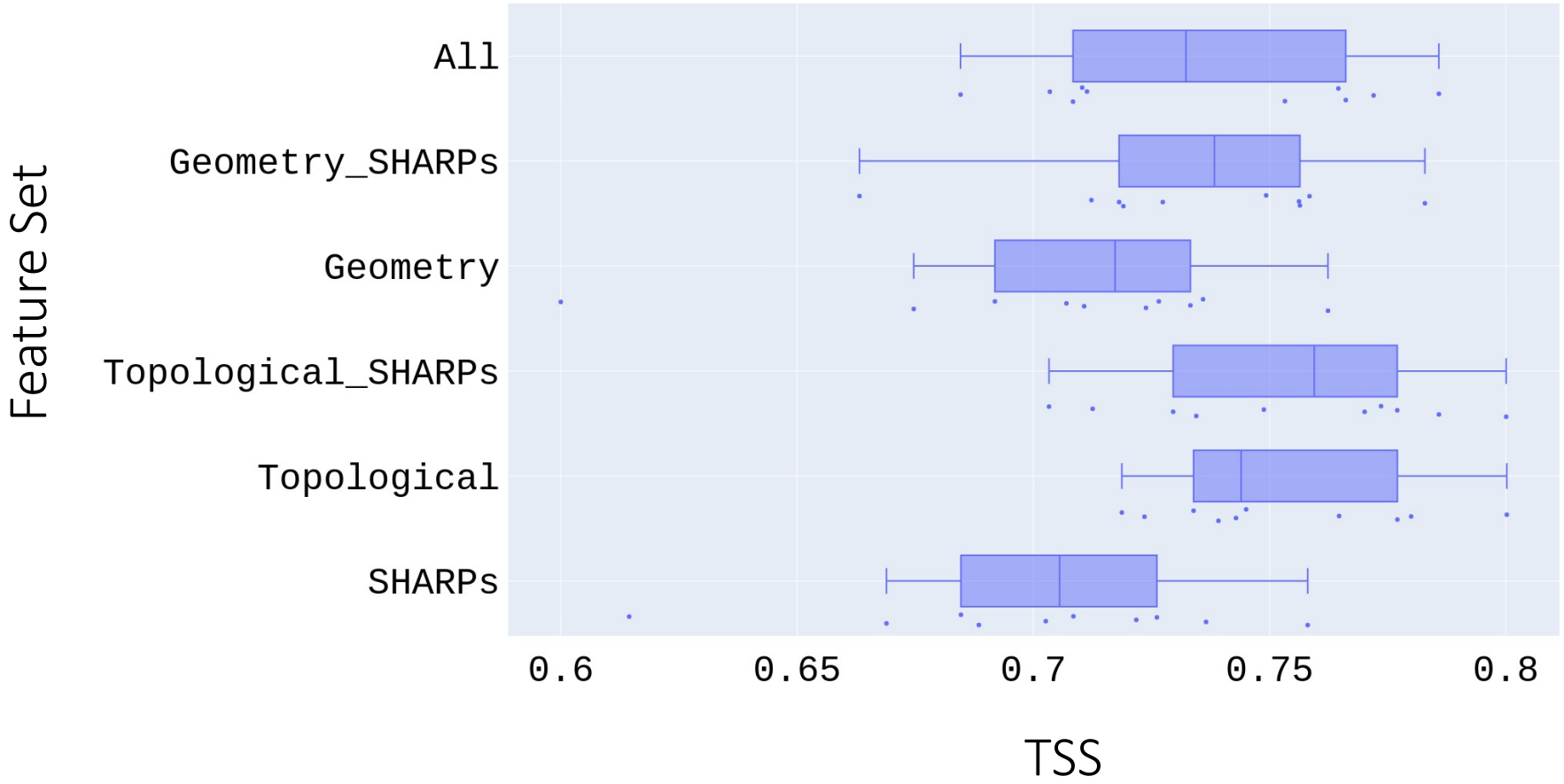
Hyperparameter Tuning Workflow



Results


Feature Set
All
Geometry_SHARPs
Geometry
Topological_SHARPs
Topological
SHARPs
0.6
0.65
0.7
0.75
0.8
TSS

Results



Shape-based features perform better than the physics-based ones, showing an improvement of ~0.05 in the TSS score.

How well is the model doing?

Model Maximized TSS = $\frac{TP}{TP+FN} - \frac{FP}{FP+TN}$

	Accuracy	Precision	Recall	FB	TSS	HSS
SHARPs (19)	0.84 ± 0.02	0.06 ± 0.01	0.87 ± 0.05	13.84 ± 1.93	0.70 ± 0.01	0.09 ± 0.02
Geometry (16)	0.82 ± 0.01	0.06 ± 0.01	0.89 ± 0.04	14.89 ± 1.15	0.71 ± 0.04	0.09 ± 0.01
Topology (20)	0.86 ± 0.02	0.08 ± 0.01	0.90 ± 0.02	12.20 ± 1.96	0.75 ± 0.03	0.12 ± 0.02
SHARPs + Geometry (35)	0.84 ± 0.02	0.07 ± 0.01	0.89 ± 0.05	13.24 ± 1.98	0.73 ± 0.03	0.11 ± 0.01
SHARPs + Topology (39)	0.86 ± 0.01	0.08 ± 0.01	0.89 ± 0.03	11.55 ± 1.06	0.75 ± 0.03	0.12 ± 0.01
SHARPs + Topology + Geometry (55)	0.86 ± 0.01	0.08 ± 0.01	0.87 ± 0.04	11.77 ± 1.27	0.74 ± 0.03	0.11 ± 0.01

Table 1: Performance of the various feature sets based on the various metrics described in Eqn. 1. For all the metrics except for FB, higher is better.

How well is the model doing?

Model Maximized TSS = $\frac{TP}{TP+FN} - \frac{FP}{FP+TN}$ $\frac{TP}{TP+FP}$ $\frac{TP+FP}{TP+FN}$

	Accuracy	Precision	Recall	FB	TSS	HSS
SHARPs (19)	0.84 ± 0.02	0.06 ± 0.01	0.87 ± 0.05	13.84 ± 1.93	0.70 ± 0.01	0.09 ± 0.02
Geometry (16)	0.82 ± 0.01	0.06 ± 0.01	0.89 ± 0.04	14.89 ± 1.15	0.71 ± 0.04	0.09 ± 0.01
Topology (20)	0.86 ± 0.02	0.08 ± 0.01	0.90 ± 0.02	12.20 ± 1.96	0.75 ± 0.03	0.12 ± 0.02
SHARPs + Geometry (35)	0.84 ± 0.02	0.07 ± 0.01	0.89 ± 0.05	13.24 ± 1.98	0.73 ± 0.03	0.11 ± 0.01
SHARPs + Topology (39)	0.86 ± 0.01	0.08 ± 0.01	0.89 ± 0.03	11.55 ± 1.06	0.75 ± 0.03	0.12 ± 0.01
SHARPs + Topology + Geometry (55)	0.86 ± 0.01	0.08 ± 0.01	0.87 ± 0.04	11.77 ± 1.27	0.74 ± 0.03	0.11 ± 0.01

Table 1: Performance of the various feature sets based on the various metrics described in Eqn. 1. For all the metrics except for FB, higher is better.

How well is the model doing?

Model Maximized TSS = $\frac{TP}{TP+FN} - \frac{FP}{FP+TN}$

$\frac{TP}{TP+FP}$

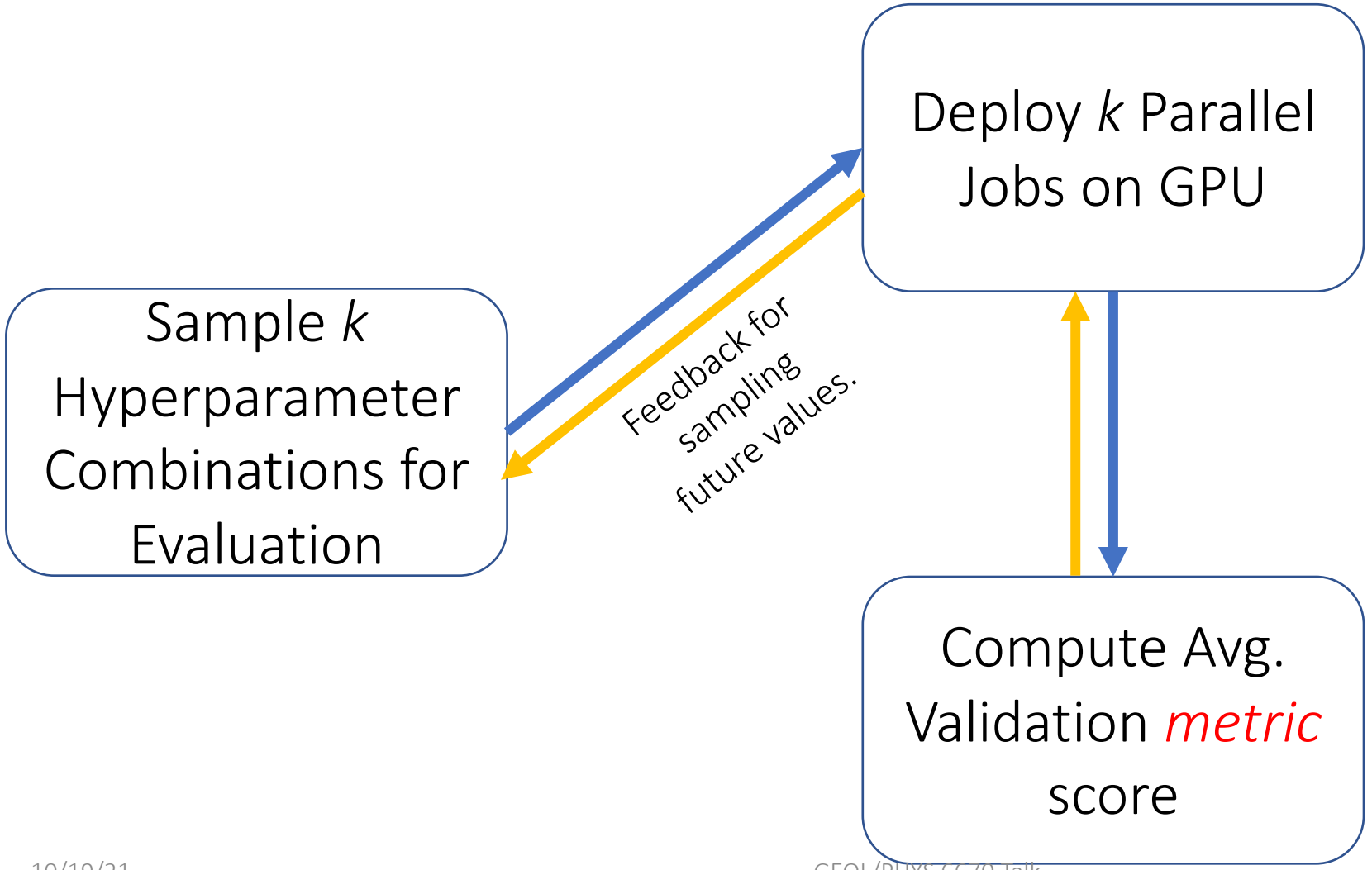
$\frac{TP+FP}{TP+FN}$

	Accuracy	Precision	Recall	FB	TSS	HSS
SHARPs (19)	0.84 ± 0.02	0.06 ± 0.01	0.87 ± 0.05	13.84 ± 1.93	0.70 ± 0.01	0.09 ± 0.02
Geometry (16)	0.82 ± 0.01	0.06 ± 0.01	0.89 ± 0.04	14.89 ± 1.15	0.71 ± 0.04	0.09 ± 0.01
Topology (20)	0.86 ± 0.02	0.08 ± 0.01	0.90 ± 0.02	12.20 ± 1.96	0.75 ± 0.03	0.12 ± 0.02
SHARPs + Geometry (35)	0.84 ± 0.02	0.07 ± 0.01	0.89 ± 0.05	13.24 ± 1.98	0.73 ± 0.03	0.11 ± 0.01
SHARPs + Topology (39)	0.86 ± 0.01	0.08 ± 0.01	0.89 ± 0.03	11.55 ± 1.06	0.75 ± 0.03	0.12 ± 0.01
SHARPs + Topology + Geometry (55)	0.86 ± 0.01	0.08 ± 0.01	0.87 ± 0.04	11.77 ± 1.27	0.74 ± 0.03	0.11 ± 0.01

Table 1: Performance of the various feature sets based on the various metrics described in Eqn. 1. For all the metrics except for FB, higher is better.

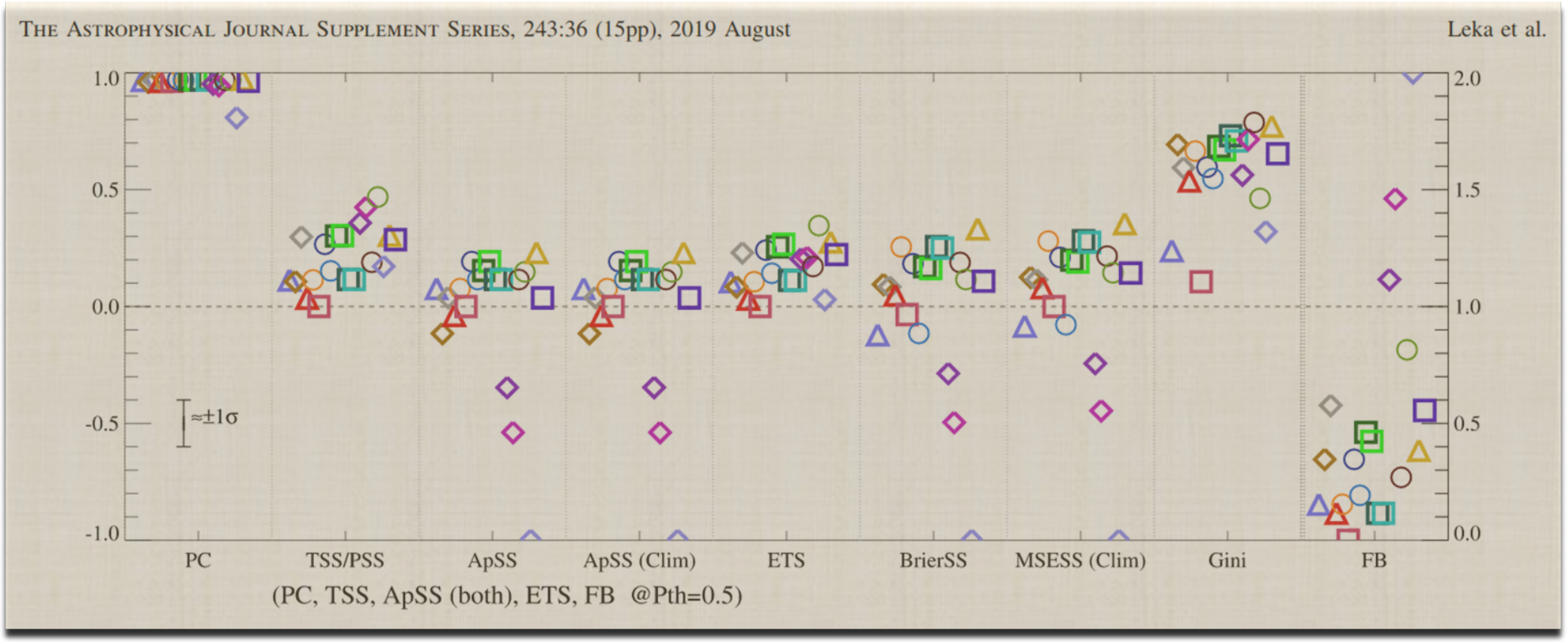
- High values of FB and lower values of Precision indicates our model is overforecasting: **High FP relative to TP and TN.**
- ***Cause: Optimizing on the TSS leads to higher ratio of w_{flare}:$w_{no\text{-}flare}$***

Hyperparameter Tuning on Specific Metrics



Comparison with the Leka et al. 2019 results

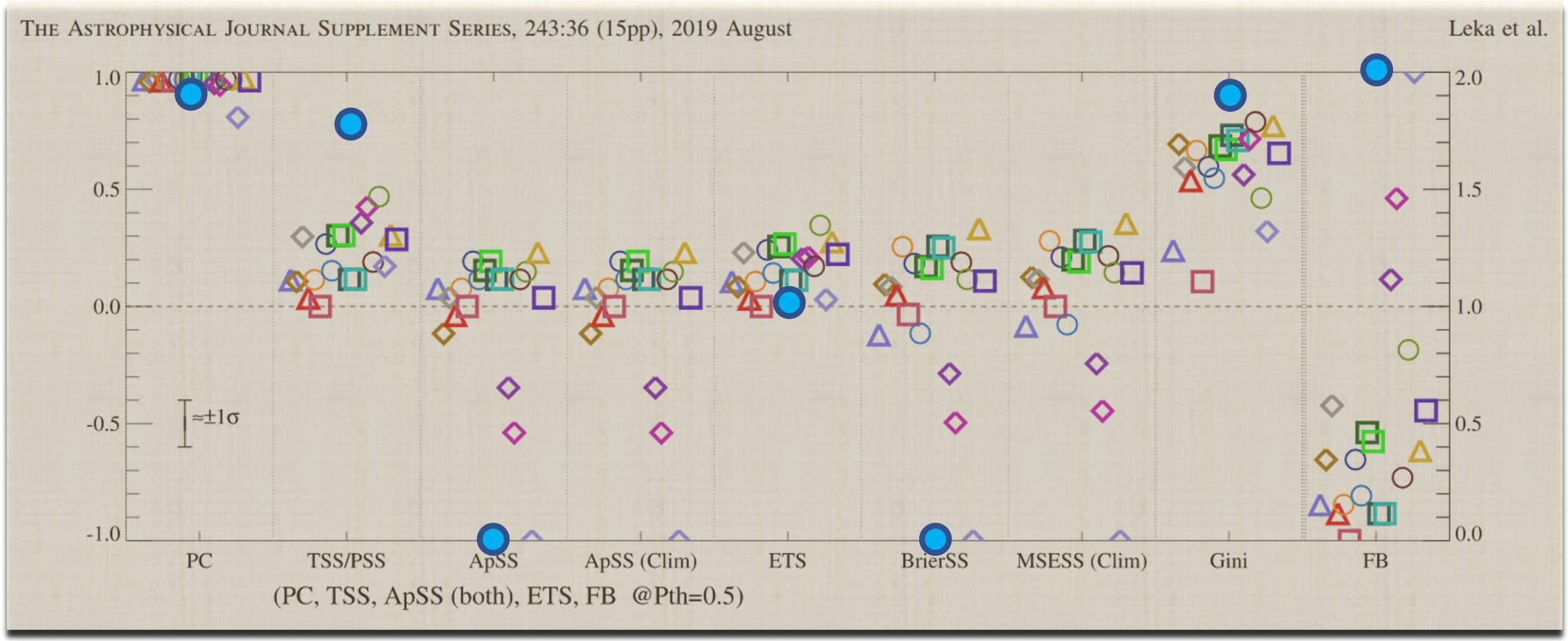
- Training Set: 2010-2015, Testing Set: 2016-2017.
- M1.0+/24 Forecast



Comparison with the Leka et al. 2019 results

- Training Set: 2010-2015, Testing Set: 2016-2017.
- M1.0+/24 Forecast

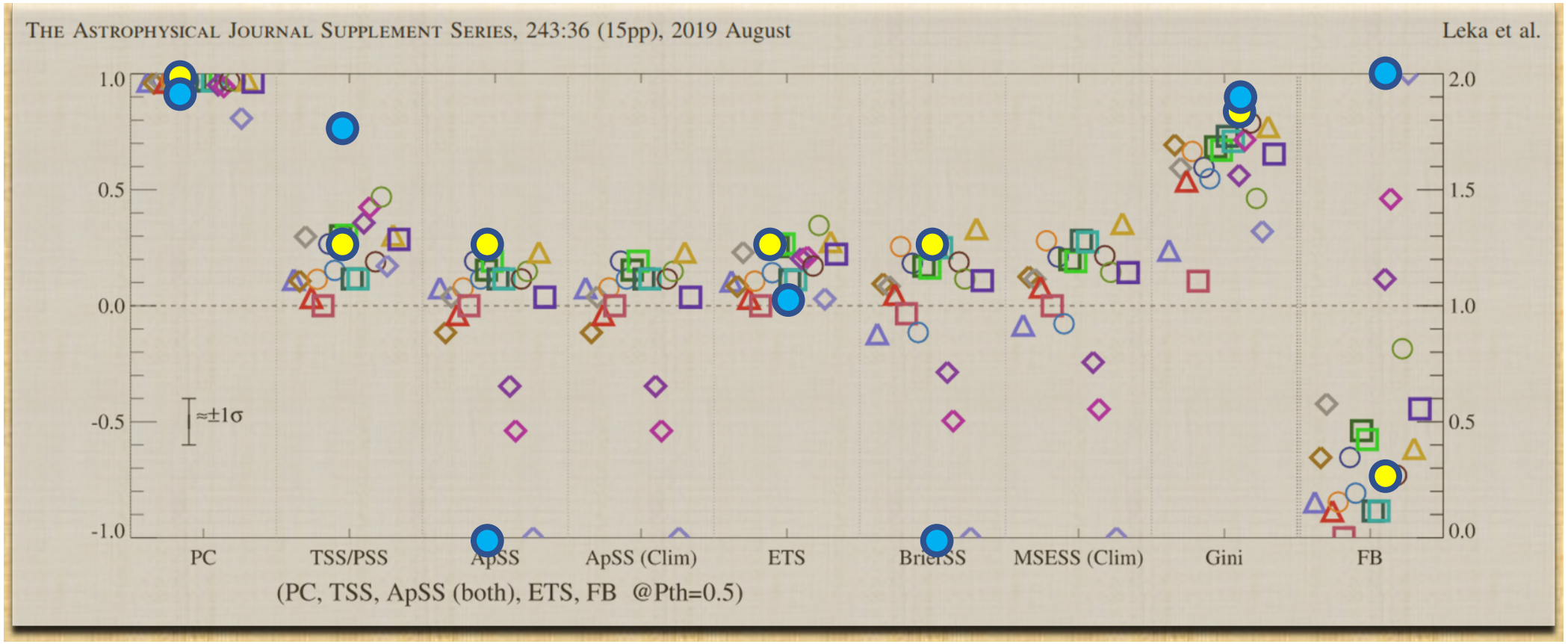
Our model optimized on TSS



Comparison with the Leka et al. 2019 results

- Training Set: 2010-2015, Testing Set: 2016-2017.
- M1.0+/24 Forecast

Our model optimized on TSS
Our model optimized on Precision



Conclusions

- Shape-based features perform better than physics-based features in the context of an MLP model.
  - Topology + SHARPs combination provides the best performance.

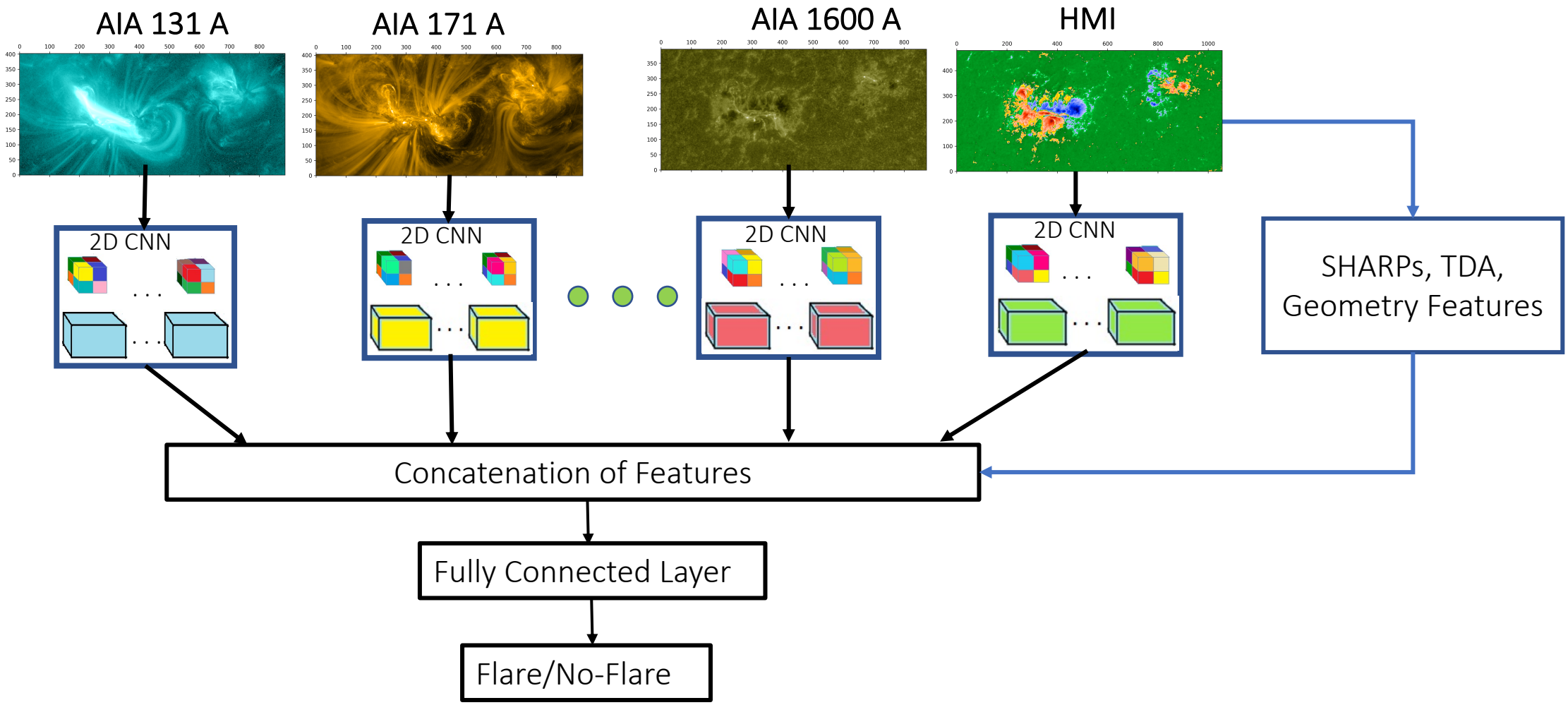
- Optimizing on TSS can lead to overforecasting.
  - Hyperparameter Tuning Framework for tuning specific metrics.

- Comparison with other forecasting methods (Leka et al. 2019).

Future Work

- Improved geometry-based and topology-based featurizaton
- Validation using alternative ML/DL models
- Multi-variate Feature Ranking
- Impact of dataset manipulations (cadence, augmentation) on model performance
- Study trade-offs across different metrics

A Hybrid Flare Forecasting System

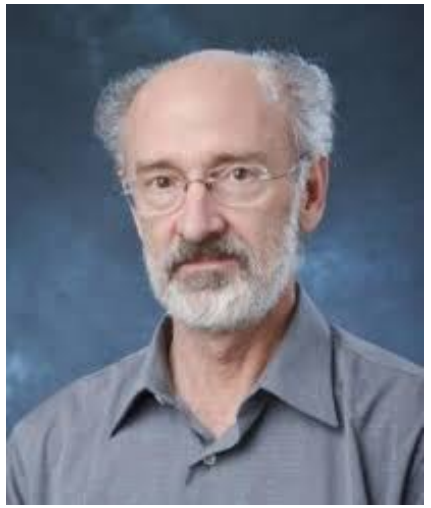


Acknowledgements

Varad Deshmukh,
CU Boulder,
Computer Science

Dr. Elizabeth Bradley,
CU Boulder,
Computer Science

Dr. Thomas Berger,
SWx TREC



Dr. James Meiss,
CU Boulder,
Applied Mathematics

Funding Support

SWx TREC

NSF

NASA

Resources

- Packages
  - Pytorch (deep learning models): https://pytorch.org/
  - Scikit-learn (ML models and metrics): https://scikit-learn.org/stable/
  - Ray.tune (Hyperparameter tuning): https://docs.ray.io/en/latest/tune/index.html

- Textbooks:
  - Machine Learning: A Probabilistic Perspective. *Kevin A. Murphy*
  - Deep Learning. *Ian Goodfellow, Yoshua Bengio, and Aaron Courville*
-

- Pytorch implementation of a simple neural network:
  - https://pytorch.org/tutorials/beginner/basics/intro.html
  - https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

- ML/DL courses:
  - Andrew Ng ML course in MATLAB/Octave: https://www.coursera.org/learn/machine-learning
  - Jeremy Howard's deep learning course: https://www.fast.ai/

Additional Slides

Model Implementation in Pytorch

```
class SimpleDNN(torch.nn.Module):

    #Our batch shape for input x is (3, 256, 256)

    def __init__(self, in_shape):
        super(SimpleDNN, self).__init__()
        self.in_shape = in_shape
        self.fc1 = torch.nn.Linear(in_shape, 36)
        self.fc2 = torch.nn.Linear(36, 24)
        self.fc3 = torch.nn.Linear(24, 16)
        self.fc4 = torch.nn.Linear(16, 2)
        self.softmax = torch.nn.Softmax()
        self.dropout1 = torch.nn.Dropout(0.01)
        self.dropout2 = torch.nn.Dropout(0.001)
        self.dropout3 = torch.nn.Dropout(0.0001)

    def forward(self, x):

        x = x.view(-1, self.in_shape)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)

        return x
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
NN = SimpleDNN(in_shape=len(cols));
NN.to(device);

net = NN
batch_size = 128
n_epochs = 15
learning_rate = 5e-4
k_fold = 5
```

Loss function and Optimizer

```
def createLossAndOptimizer(net, learning_rate, weight, L2_penalty):

    #Loss function
    loss = torch.nn.CrossEntropyLoss(weight=weight)

    #Optimizer
    optimizer = optim.Adagrad(net.parameters(), lr=learning_rate, weight_decay=L2_penalty)

    return(loss, optimizer)
```

Forward pass and backward propagation

```
for i, data in enumerate(train_loader, 0):

    #Get inputs
    inputs, labels = data

    #Wrap them in a Variable object
    inputs, labels = Variable(inputs), Variable(labels)
    inputs, labels = inputs.to(device), labels.to(device)

    #Set the parameter gradients to zero
    optimizer.zero_grad()

    #Forward pass, backward pass, optimize
    outputs = net(inputs)
    loss_size = loss(outputs, labels)
    loss_size.backward()
    optimizer.step()

    #Print statistics
    running_loss += loss_size.data.item()
    total_train_loss += loss_size.data.item()
```

Hyperparameter tuning

```
search_space = {
        "weight": (100,200),
        "L2_penalty": (-3, -1),
        "learning_rate": (-5, -3),
        "decay_factor" : (0.9, 1.0)
        }

search_alg = BayesOptSearch(
        search_space,
        metric="tss",
        mode="max",
        random_search_steps=10,
        max_concurrent=10,
        random_state=1442,
        utility_kwargs={
            "kind": "ucb",
            "kappa": 5,
            "xi": 0.0
        })
```

```
scheduler = AsyncHyperBandScheduler(metric="tss", mode="max")

analysis = tune.run(trainNet,
                    #config=search_space,
                    #scheduler=ASHAScheduler(metric="tss", mode="max", grace_period=1),
                    search_alg=search_alg,
                    scheduler=scheduler,
                    num_samples=40,
                    resources_per_trial={'gpu': 0.1},
                    config={"num_workers": 10,
                            "num_gpus": 0.1})
```

Evaluating on the test set

```
for inputs, labels in test_loader:

    #Wrap tensors in Variables
    inputs, labels = Variable(inputs), Variable(labels)
    inputs, labels = inputs.to(device), labels.to(device)

    #Forward pass
    outputs = model(inputs)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
    for t, p in zip(labels.view(-1), predicted.view(-1)):
        confusion_matrix[t.long(), p.long()] += 1

    test_outputs += softmax(outputs).cpu().detach().numpy()[:,1].tolist()
    test_labels += labels.cpu().detach().numpy().tolist()
    #test_loss_size = loss(test_outputs, labels)
    #total_test_loss += test_loss_size.data.item()

tp = confusion_matrix[1,1]
tn = confusion_matrix[0,0]
fp = confusion_matrix[0,1]
fn = confusion_matrix[1,0]
```