TREE BOOSTING FOR LEARNING EFT PARAMETERS

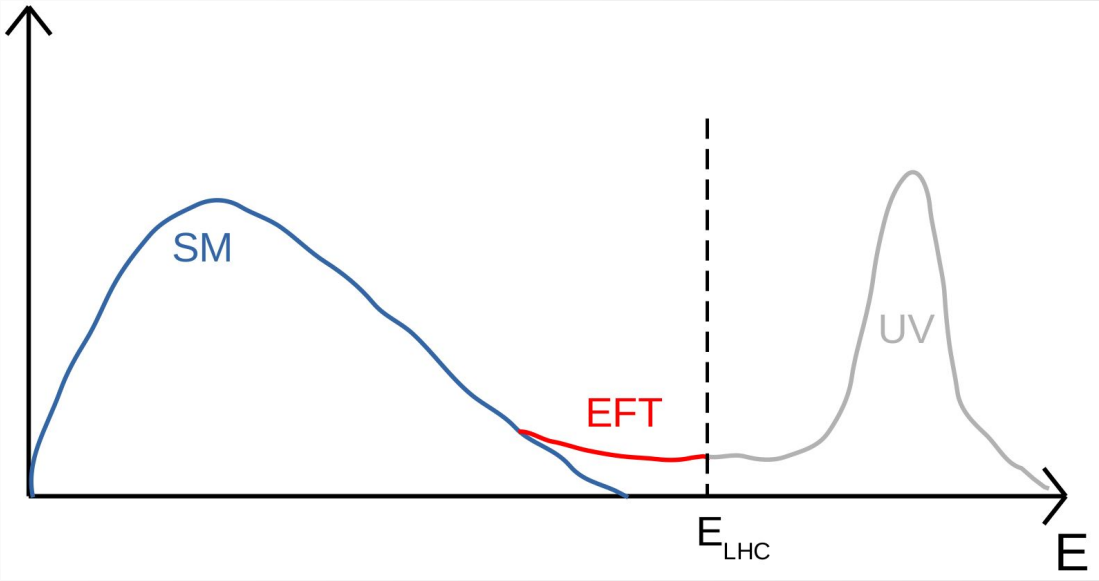
S. Chatterjee, R. Schöfbeck, **D. Schwarz** (HEPHY Vienna)

Glühwein Workshop 2.0 - GlühWien
December 13-15, 2023

EFT in a nutshell

- No direct evidence for new physics at the LHC
- Might be out of reach of LHC
- Study small deviations from the SM with effective field theories
- Extend SM Lagrangian by dimension 6 operators:

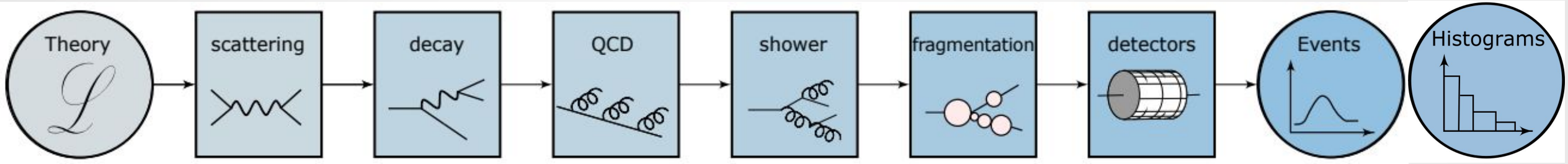
$$\mathcal{L}_{\text{eff}} = \mathcal{L}_{\text{SM}} + \sum_i \frac{c_i}{\Lambda^2} \mathcal{O}_i$$



- Wilson coefficients: coupling strength

Simulation sequence

image from arXiv:2211.01421



theory $\xrightarrow{p(z_p|\theta)}$ parton level $\xrightarrow{p(z_{ptl}|z_p)}$ particle level $\xrightarrow{p(x_{det}|z_{ptl})}$ analysis level

- In this picture, θ is an EFT parameter (Wilson coefficient)
- pdf $p(z_p|\theta)$ can be interpreted as diff. cross section $1/\sigma\, d\sigma(\theta)/dz_p$
- Now particle level with non-perturbative effects
- Reconstruction level depends on θ: $p(x_{det}|\theta) = \int dz_{ptl} \int dz_p [\cdots]$
- Integrated over large latent space! (observables we don't observe)

Likelihood ratio is the optimal statistic (Neyman-Pearson Lemma) for testing two hypotheses:

$$\mathrm{LR}(x_{det}|H_1, H_2) \equiv \frac{p(x_{det}|\theta = H_1)}{p(x_{det}|\theta = H_2)}$$

Would like to evaluate for varying θ
Note: latent space integrated in num&den!

Situation in classification

- ML helps in parameterizing the likelihood ratio using "simulation based inference"
- Possible because we have a history of the generation sequence!
- In classification (signal vs background) θ -> isSig ∈ {0,1}
- The loss function

$$L = \langle f(x)^2 \rangle_{\text{isSig}=1} + \langle (1 - f(x))^2 \rangle_{\text{isSig}=0} = \sum_{\text{isSig}} \int \mathrm{d}x \cdots$$

  can be evaluated using generator level information (i.e. "isSignal")

- With this loss, we converge to:

$$f^*(x) = \frac{1}{1 + \frac{p(x|\text{isSig}=1)}{p(x|\text{isSig}=0)}}$$

(here z is already integrated out)

- Latent space integrated in numerator and denominator
- Neyman-Pearson: optimal test statistic for a cross section measurement

-> Exactly what we need!

How we do the same for EFTs?

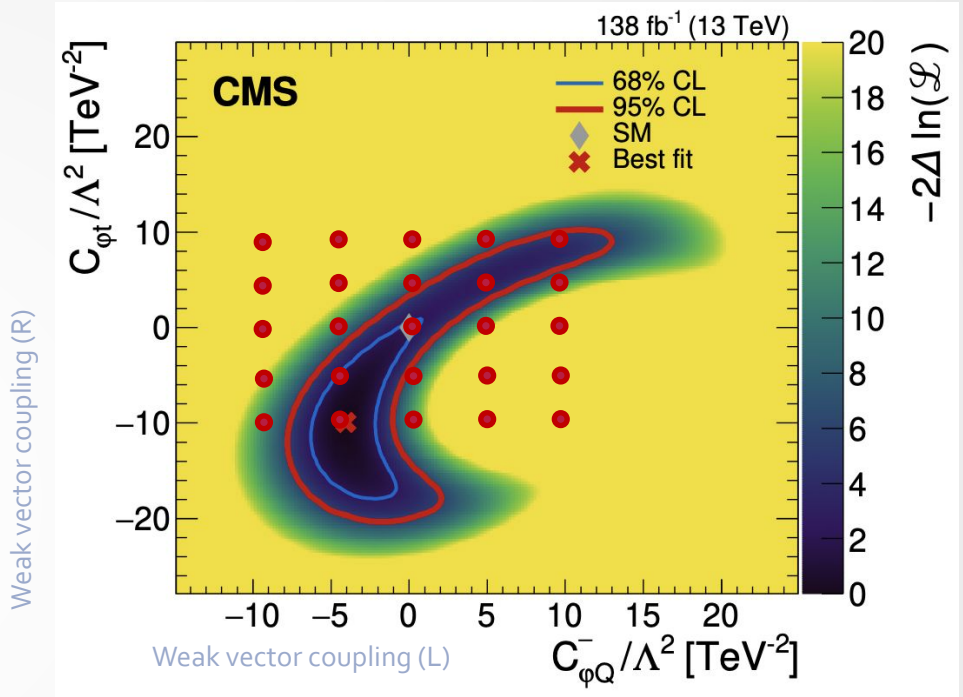
Can we learn EFT effects "on average"?

$$L = \sum_{\theta \in \mathcal{B}} \int \mathrm{d}\boldsymbol{x} \left(p(\boldsymbol{x}|\boldsymbol{\theta}) \hat{f}(\boldsymbol{x})^2 + p(\boldsymbol{x}|\mathrm{SM})(1 - \hat{f}(\boldsymbol{x}))^2 \right)$$

$\boldsymbol{\theta}$ - *ignorant*

mixing signals &
case dependent mixes

$$f^*(\boldsymbol{x}) = \frac{1}{1 + r_{\mathcal{B}}(\boldsymbol{x})} \qquad r_{\mathcal{B}}(\boldsymbol{x}) = \frac{\frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{\theta} \in \mathcal{B}} p(\boldsymbol{x}|\boldsymbol{\theta})}{p(\boldsymbol{x}|\mathrm{SM})}$$



- We can try to learn EFT effects "on average"
- Sending 'mixed signals' to the loss function
  - Averages the training data set -> suboptimal when linear effects dominate -> positive and negative can cancel
  - Classifier does not reflect knowledge on the $\boldsymbol{\theta}$-dependence, which we do know!

1. Go back and inject $\boldsymbol{\theta}$ polynomial SMEFT dependence in estimator
2. Exploit the fact that SMEFT predictions can be weighted: Only need one training data set

[JHEP 12 (2021) 083]

SMEFT reweighting

- Matrix element depends on Wilson coefficients:

$$|\mathcal{M}_{\mathrm{SM}} + \theta_i \mathcal{M}^i_{\mathrm{BSM}}|^2 = |\mathcal{M}_{\mathrm{SM}}|^2 + \underbrace{\theta_i \mathrm{Re}\mathcal{M}^*_{\mathrm{SM}}\mathcal{M}^i_{\mathrm{BSM}}}_{\text{interference, linear}} + \underbrace{\theta_i\theta_j \mathcal{M}^{i*}_{\mathrm{BSM}}\mathcal{M}^j_{\mathrm{BSM}}}_{\text{BSM, quadratic}}$$

- In simulated samples we store the ME for each event at various EFT points as weights
- Strictly polynomial structure allows to get function w(θ) for each event
- Sufficient EFT points needed: (N + 2)(N + 1)/2 for N different θ

-> Only one sample needed!

Exploiting the SMEFT reweighting

Start with classifier but now it is aware of θ:
(details later)

$$L = \sum_{\theta \in \mathcal{B}} \left(\langle \hat{f}(x;\theta)^2 \rangle_\theta + \langle (1 - \hat{f}(x;\theta))^2 \rangle_{\mathrm{SM}} \right)$$

Exploiting the SMEFT reweighting

Start with classifier but now it is aware of θ: (details later)

$$L = \sum_{\theta \in \mathcal{B}} \left(\langle \hat{f}(x;\theta)^2 \rangle_\theta + \langle (1 - \hat{f}(x;\theta))^2 \rangle_{\mathrm{SM}} \right)$$

Write as integral over joint space:

$$= \sum_{\theta \in \mathcal{B}} \int \mathrm{d}x \, \mathrm{dz} \, \left(p(x,z|\theta) \hat{f}(x;\theta)^2 + p(x,z|\mathrm{SM})(1 - \hat{f}(x;\theta))^2 \right)$$

Exploiting the SMEFT reweighting

Start with classifier but now it is aware of θ: (details later)

$$L = \sum_{\theta \in \mathcal{B}} \left(\langle \hat{f}(x;\theta)^2 \rangle_\theta + \langle (1 - \hat{f}(x;\theta))^2 \rangle_{\mathrm{SM}} \right)$$

Write as integral over joint space:

$$= \sum_{\theta \in \mathcal{B}} \int \mathrm{d}x\, \mathrm{dz} \, \left(p(x,z|\theta) \hat{f}(x;\theta)^2 + p(x,z|\mathrm{SM})(1 - \hat{f}(x;\theta))^2 \right)$$

Now pull SM pdf in front:

$$= \sum_{\theta \in \mathcal{B}} \int \mathrm{d}x\, \mathrm{dz}\, p(x,z|\mathrm{SM}) \left(r(x,z|\theta) \hat{f}(x;\theta)^2 + (1 - \hat{f}(x;\theta))^2 \right)$$

"joint" likelihood ratio r(x,z|θ) appears in the estimator

-> we need to know r(x,z|θ) for each event

Exploiting the SMEFT reweighting

Start with classifier but now it is aware of θ:
(details later)

$$L = \sum_{\theta \in \mathcal{B}} \left(\langle \hat{f}(x;\theta)^2 \rangle_\theta + \langle (1 - \hat{f}(x;\theta))^2 \rangle_{\mathrm{SM}} \right)$$

Write as integral over joint space:

$$= \sum_{\theta \in \mathcal{B}} \int \mathrm{d}x\, \mathrm{dz} \, \left(p(x,z|\theta) \hat{f}(x;\theta)^2 + p(x,z|\mathrm{SM})(1 - \hat{f}(x;\theta))^2 \right)$$

Now pull SM pdf in front:

$$= \sum_{\theta \in \mathcal{B}} \int \mathrm{d}x\, \mathrm{dz}\, p(x,z|\mathrm{SM}) \left(r(x,z|\theta) \hat{f}(x;\theta)^2 + (1 - \hat{f}(x;\theta))^2 \right)$$

"joint" likelihood ratio r(x,z|θ) appears in the estimator
-> we need to know r(x,z|θ) for each event

Let's write it down, assume θ only affects parton level:
Factorizes and PDFs cancel [well established, references in backup]

$$r = \frac{p(x_{\mathrm{det}}, \cdots, z_{\mathrm{ptl}}, \cdots, z_{\mathrm{p}}|\theta)}{p(x_{\mathrm{det}}, \cdots, z_{\mathrm{ptl}}, \cdots, z_{\mathrm{p}}|\mathrm{SM})} = \frac{p(x_{\mathrm{det}}|z_{\mathrm{ptl}}) \cdots p(z_{\mathrm{ptl}}|z_{\mathrm{p}}) \cdots p(z_{\mathrm{p}}|\theta)}{p(x_{\mathrm{det}}|z_{\mathrm{ptl}}) \cdots p(z_{\mathrm{ptl}}|z_{\mathrm{p}}) \cdots p(z_{\mathrm{p}}|\mathrm{SM})} = \frac{p(z_{\mathrm{p}}|\theta)}{p(z_{\mathrm{p}}|\mathrm{SM})} \sim \frac{|\mathcal{M}(z_{\mathrm{p}},\theta)|^2}{|\mathcal{M}(z_{\mathrm{p}},\mathrm{SM})|^2}$$

This is exactly our weight function w(θ)!

Parametrized Classifiers

K. Cranmer , J. Pavez , and G. Louppe [1506.02169]
J. Brehmer, K. Cranmer, G. Louppe, J. Pavez [1805.00013]
[1805.00020]
[1805.12244]
J. Brehmer, F. Kling, I. Espejo, K. Cranmer [1907.10621]

Loss: $L = \sum_{\theta \in \mathcal{B}} \int \mathrm{d}x\, \mathrm{d}z\, p(x, z|\mathrm{SM}) \left(r(x, z|\theta) \hat{f}(x; \theta)^2 + (1 - \hat{f}(x; \theta))^2 \right)$ here: MSE, but cross entropy also works

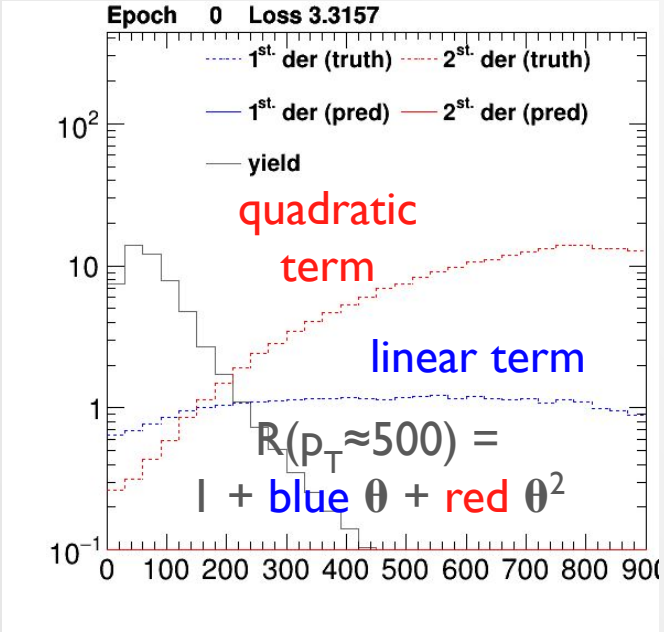
$$\hat{f}(x; \theta) = \frac{1}{1 + \hat{R}(x; \theta)}$$ likelihood trick, we need to know the likelihood ratio R

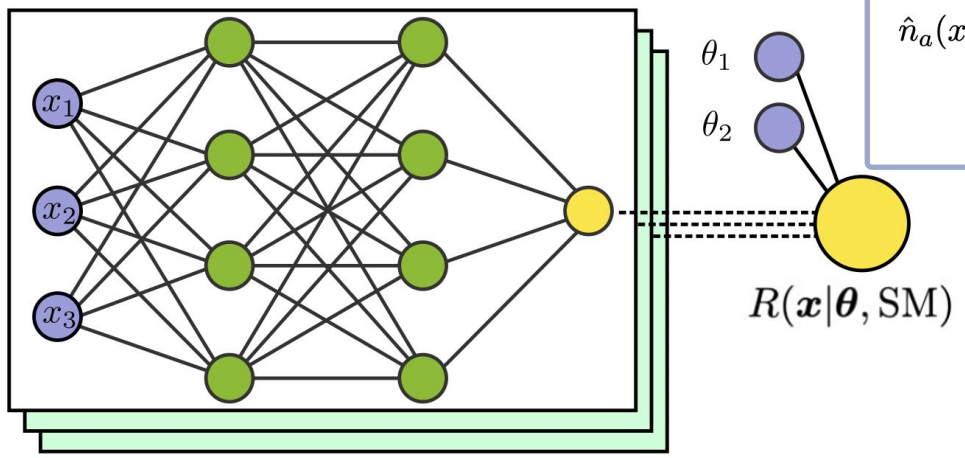
Similar to
S. Chen, A. Glioti,
G. Panico, A. Wulzer
JHEP 05 (2021) 247

arXiv:2308.05704

Insert knowledge of θ-dependence, assume general polynomial:

$$\hat{R}(x; \theta) = \left(1 + \sum_a \theta_a \hat{n}_a(x)\right)^2 + \sum_a \left(\sum_{b \geq a} \theta_b \hat{n}_{ab}(x) \right)^2$$





$$\hat{n}_a(x) \to \left. \frac{\partial_a p(x|\ \theta\ )}{p(x|\mathrm{SM})} \right|_{\theta=\mathrm{SM}} = \left. \frac{\partial_a \int \mathrm{d}z\, p(x, z|\theta)}{\int \mathrm{d}z\, p(x, z|\mathrm{SM})} \right|_{\theta=\mathrm{SM}}$$

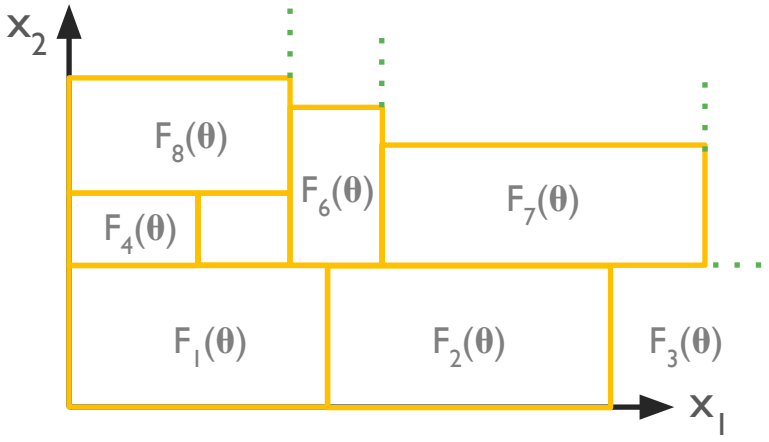
Integrates latent space!

Fit coefficient functions n with NN. This is the current status - Why would you want to use trees instead?

The tree algorithm

[arXiv:2107.10859, arXiv:2205:12976]

phase-space partitioning



A simple tree

index-function (non-linearity)

$$\hat{F}(\boldsymbol{x},\boldsymbol{\theta}) = \sum_{j\in\mathcal{J}} \mathbb{1}_j(\boldsymbol{x}) F_j(\boldsymbol{\theta})$$

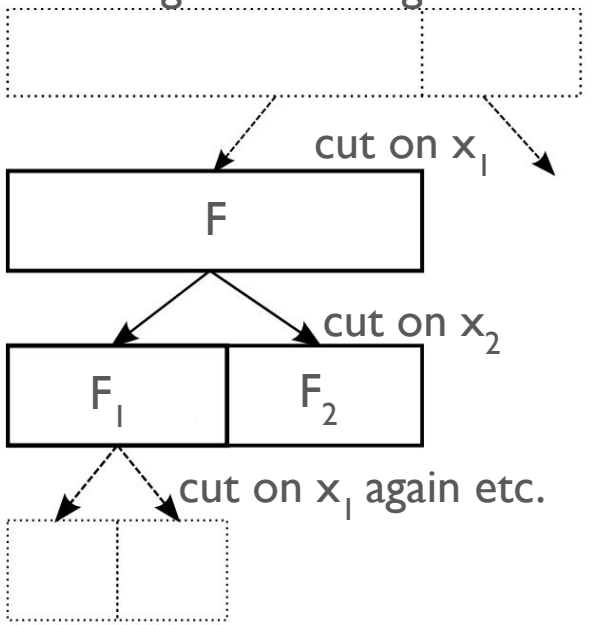
phase space partitioning J

prediction F_j

need to solve for partitioning J and $\{F_j\}$

training phase:

e.g. "CART" algo



- A tree is a hierarchical phase-space partitioning ($\mathscr{J}$)
- Novelty in the Boosted Information Tree is that we associate each region j with a polynomial $F_j(\boldsymbol{\theta})$
- Fitting tree means optimizing "node split positions" on some loss.
- Training on **ENSEMBLE**!

Trees for SMEFT

[arXiv:2107.10859, arXiv:2205:12976]

Want to regress in r, making use of $\boldsymbol{\theta}$-dependence:

$$r(x, z|\theta) = \frac{\mathrm{d}\sigma(\boldsymbol{x}, \boldsymbol{\theta})/\mathrm{d}\boldsymbol{x}}{\mathrm{d}\sigma(\boldsymbol{x}, \mathrm{SM})/\mathrm{d}\boldsymbol{x}}$$

→ will allow to compute the optimal LLR test statistic q($\mathscr{D}$)

Trees for SMEFT

[arXiv:2107.10859, arXiv:2205:12976]

Want to regress in r, making use of $\boldsymbol{\theta}$-dependence:

$$r(x, z|\theta) = \frac{\mathrm{d}\sigma(\boldsymbol{x}, \boldsymbol{\theta})/\mathrm{d}\boldsymbol{x}}{\mathrm{d}\sigma(\boldsymbol{x}, \mathrm{SM})/\mathrm{d}\boldsymbol{x}}$$

→ will allow to compute the optimal LLR test statistic q($\mathscr{D}$)

$$L = \sum_{\theta \in \mathcal{B}} \int \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{z}\, p(\boldsymbol{x}, \boldsymbol{z}|\mathrm{SM}) \left(r(x, z|\theta) - \hat{F}(\boldsymbol{x}, \boldsymbol{\theta}) \right)^2$$

Tree ansatz

$$\hat{F}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j \in \mathcal{J}} \mathbb{1}_j(\boldsymbol{x}) F_j(\boldsymbol{\theta})$$

$F_j(\boldsymbol{\theta})$ polynomial with const. coeff. (per node)

find optimal partitioning

find optimal predictor

Trees for SMEFT

[arXiv:2107.10859, arXiv:2205:12976]

Want to regress in r, making use of $\boldsymbol{\theta}$-dependence:

$$r(x, z|\theta) = \frac{\mathrm{d}\sigma(\boldsymbol{x}, \boldsymbol{\theta})/\mathrm{d}\boldsymbol{x}}{\mathrm{d}\sigma(\boldsymbol{x}, \mathrm{SM})/\mathrm{d}\boldsymbol{x}}$$

→ will allow to compute the optimal LLR test statistic q($\mathscr{D}$)

$$L = \sum_{\theta \in \mathcal{B}} \int \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{z}\, p(\boldsymbol{x}, \boldsymbol{z}|\mathrm{SM}) \left(r(x, z|\theta) - \hat{F}(\boldsymbol{x}, \boldsymbol{\theta}) \right)^2$$

Tree ansatz

$$\hat{F}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j \in \mathcal{J}} \mathbb{1}_j(\boldsymbol{x}) F_j(\boldsymbol{\theta})$$

find optimal partitioning

find optimal predictor

$F_j(\boldsymbol{\theta})$ polynomial with const. coeff. (per node)

Injecting the ansatz, this minimizes the loss:

$$F_j(\boldsymbol{\theta}) = \frac{\sum_{i \in j} w_i(\boldsymbol{\theta})}{\sum_{i \in j} w_i(\boldsymbol{\theta}_0)}$$

sum up event weights within node j & divide → very fast!

No trainable parameters in the predictor (Regression by means of classification)

Trees for SMEFT

[arXiv:2107.10859, arXiv:2205:12976]

Want to regress in r, making use of $\boldsymbol{\theta}$-dependence:

$$r(x, z|\theta) = \frac{\mathrm{d}\sigma(\boldsymbol{x}, \boldsymbol{\theta})/\mathrm{d}\boldsymbol{x}}{\mathrm{d}\sigma(\boldsymbol{x}, \mathrm{SM})/\mathrm{d}\boldsymbol{x}}$$

→ will allow to compute the optimal LLR test statistic q($\mathscr{D}$)

$$L = \sum_{\theta\in\mathcal{B}} \int \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{z}\, p(\boldsymbol{x}, \boldsymbol{z}|\mathrm{SM}) \left(r(x, z|\theta) - \hat{F}(\boldsymbol{x}, \boldsymbol{\theta})\right)^2$$

Tree ansatz

$$\hat{F}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j\in\mathcal{J}} \mathbb{1}_j(\boldsymbol{x}) F_j(\boldsymbol{\theta})$$

$F_j(\boldsymbol{\theta})$ polynomial with const. coeff. (per node)

find optimal partitioning

find optimal predictor

Injecting the ansatz, this minimizes the loss:

$$F_j(\boldsymbol{\theta}) = \frac{\sum_{i\in j} w_i(\boldsymbol{\theta})}{\sum_{i\in j} w_i(\boldsymbol{\theta}_0)}$$

sum up event weights within node j & divide → very fast!

No trainable parameters in the predictor (Regression by means of classification)

Leading to this loss (minimizing partitioning!):

$$L = -\sum_{\boldsymbol{\theta}\in\mathcal{B}} \sum_{j\in\mathcal{J}} \frac{w_j^2(\boldsymbol{\theta})}{w_j(\boldsymbol{\theta}_0)} = -\sum_{j\in\mathcal{J}} \sum_{\boldsymbol{\theta}\in\mathcal{B}} \theta^a \theta^b I_{ab}^{(j)} + \mathcal{O}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^3$$

Taylor expansion shows: We're optimizing the Fisher information!

This is only one tree → boost!

Tree boosting

[arXiv:2107.10859, arXiv:2205:12976]

- Boosting: Fit model iteratively to pseudo-residuals of the preceding iteration with learning rate η
- Ansatz :

$$\hat{F}^{(b)}(\boldsymbol{x}, \boldsymbol{\theta}) = \hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) + \eta \hat{F}^{(b-1)}(\boldsymbol{x}, \boldsymbol{\theta})$$

current iteration

previous iteration

1. Fit first tree
2. Multiply output with η
3. Fit next tree to residual

Insert into the loss function:

$$L[\hat{f}^{(b)}] = \sum_{\boldsymbol{\theta} \in \mathcal{B}} \int \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{z}\, p(\boldsymbol{x}, \boldsymbol{z}|\mathrm{SM}) \left(r(x, z|\theta) - \eta \hat{F}^{(b-1)}(\boldsymbol{x}, \boldsymbol{\theta}) - \hat{f}^{(b)}(\boldsymbol{x}, \boldsymbol{\theta}) \right)^2$$

current iteration

previous iteration

current iteration

pseudo-residual, expressed with weights:

$$w_i^{(b)}(\boldsymbol{\theta}) \quad \rightarrow \quad w_i^{(b-1)}(\boldsymbol{\theta}) - \eta\, w_i^{(b-1)}(\boldsymbol{\theta}_0) \hat{F}^{(b-1)}(\boldsymbol{x}_i, \boldsymbol{\theta})$$

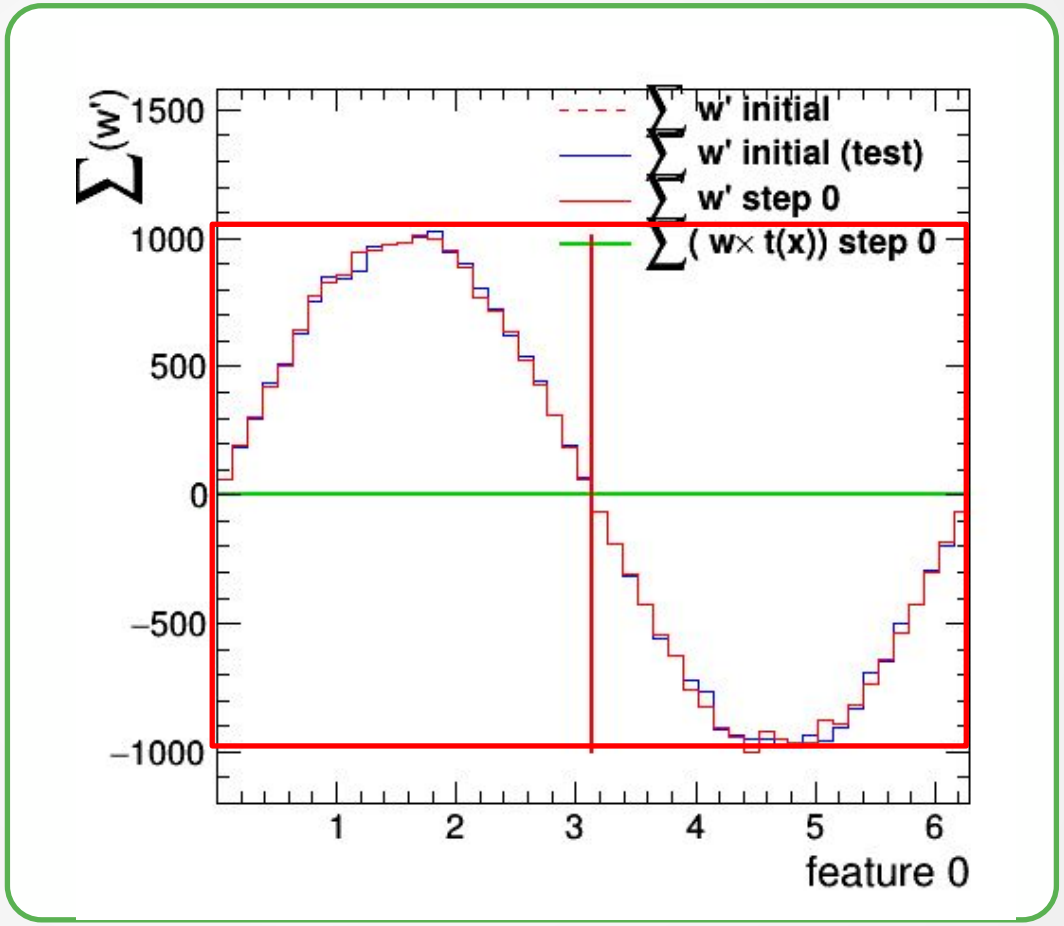
-> Iterative fitting and re-weighting (similar to AdaBoost)

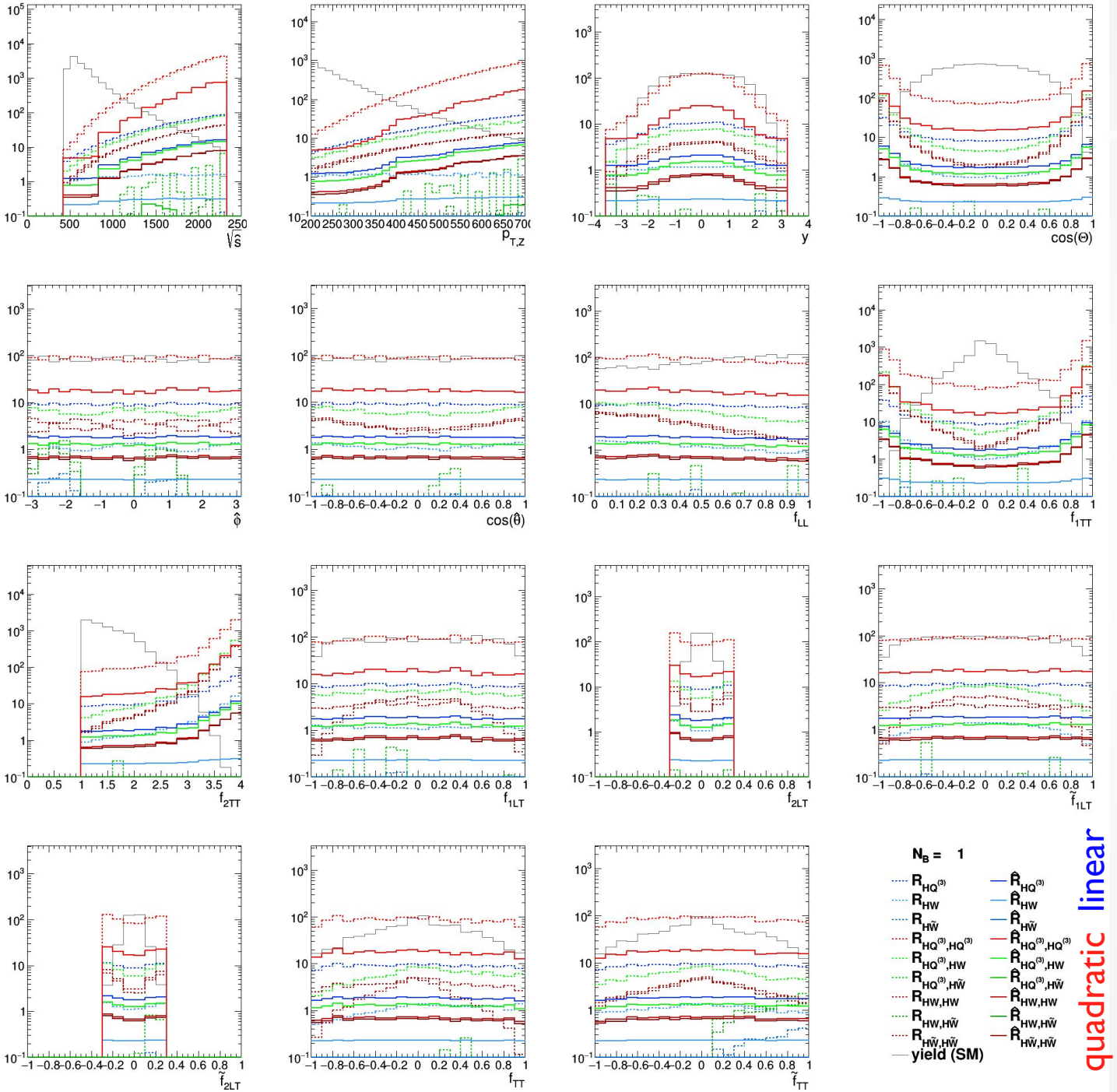
.... perform this iteratively
"Boosted Information Tree"

1D TOY EXAMPLE

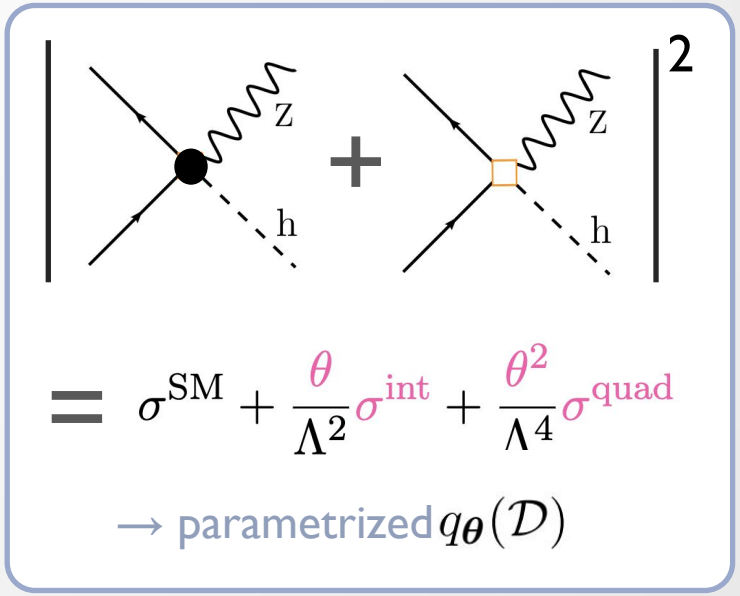
- pdf(x|$\boldsymbol{\theta}$) = 1/(2$\boldsymbol{\pi}$) - $\boldsymbol{\theta}$ Sin(x)
- We predict the first derivative of the pdf wrt. to the parameter

GIF animation not showing in pdf



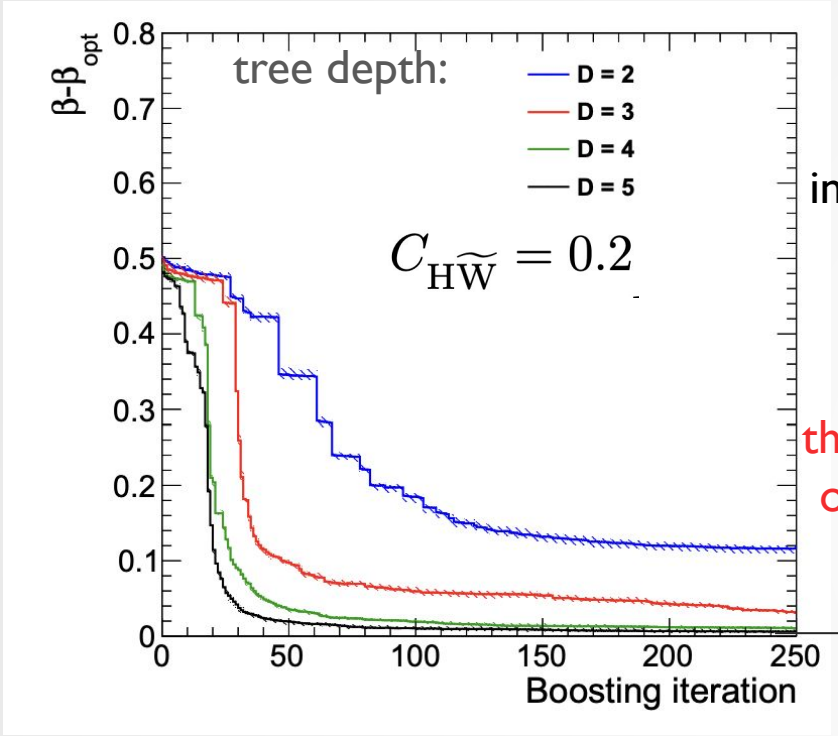


- Realistic case: model of the ZH process
- "Boosted Information Tree (BIT)"
  - 3 WC, 9 DOF, 500k events, ZH
  - 200 trees, D=5, 9 minutes of training
  - also more realistic study, including backgrounds [2107.10859], [2205.12976]
- Learning coefficient functions to compute parametrized optimal oberables



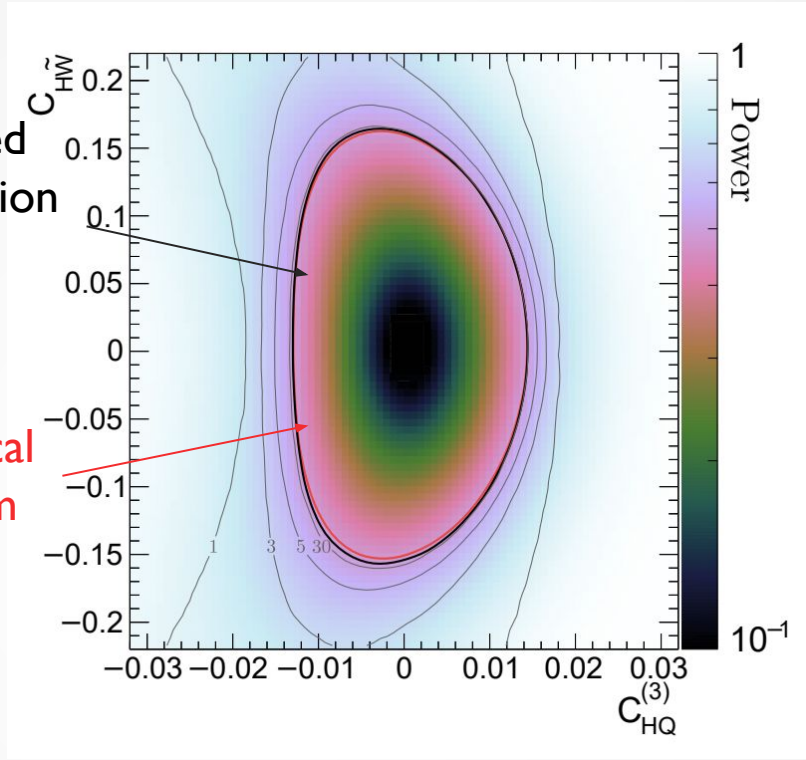
OPTIMALITY IN TEST CASES

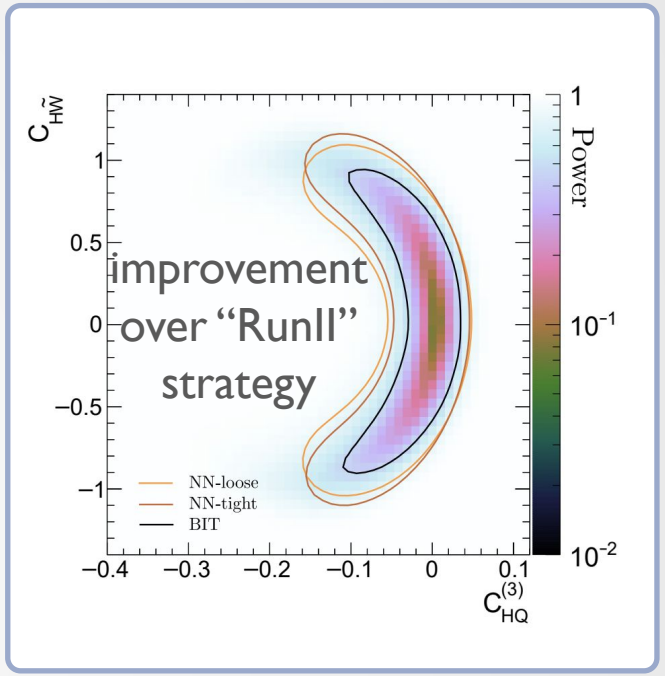
[arXiv:2107.10859, arXiv:2205:12976]



Boosted information tree (BIT)

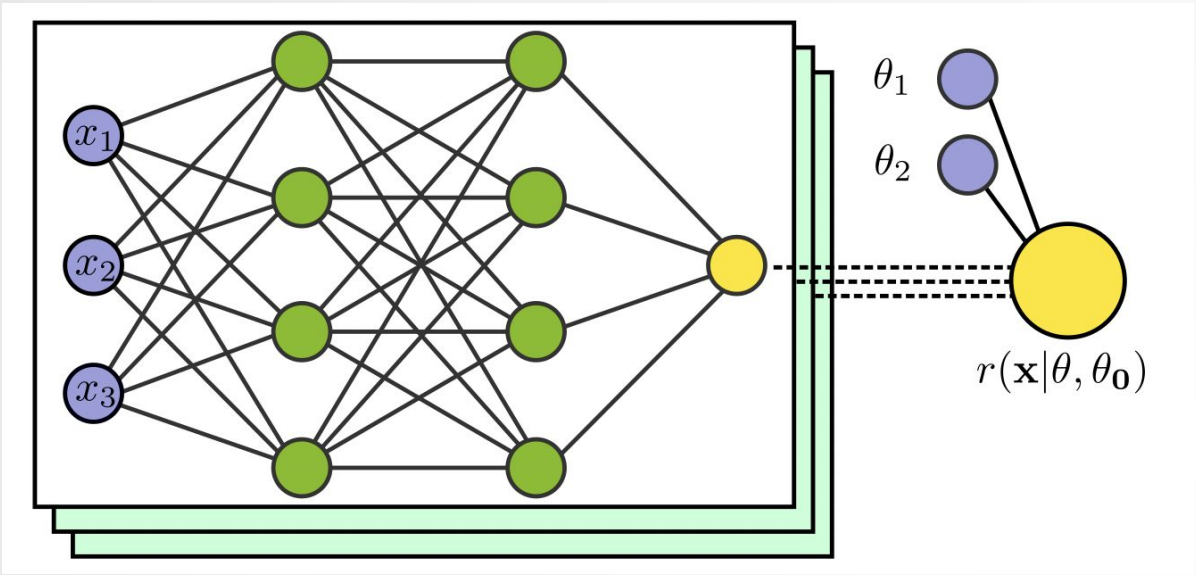
theoretical optimum

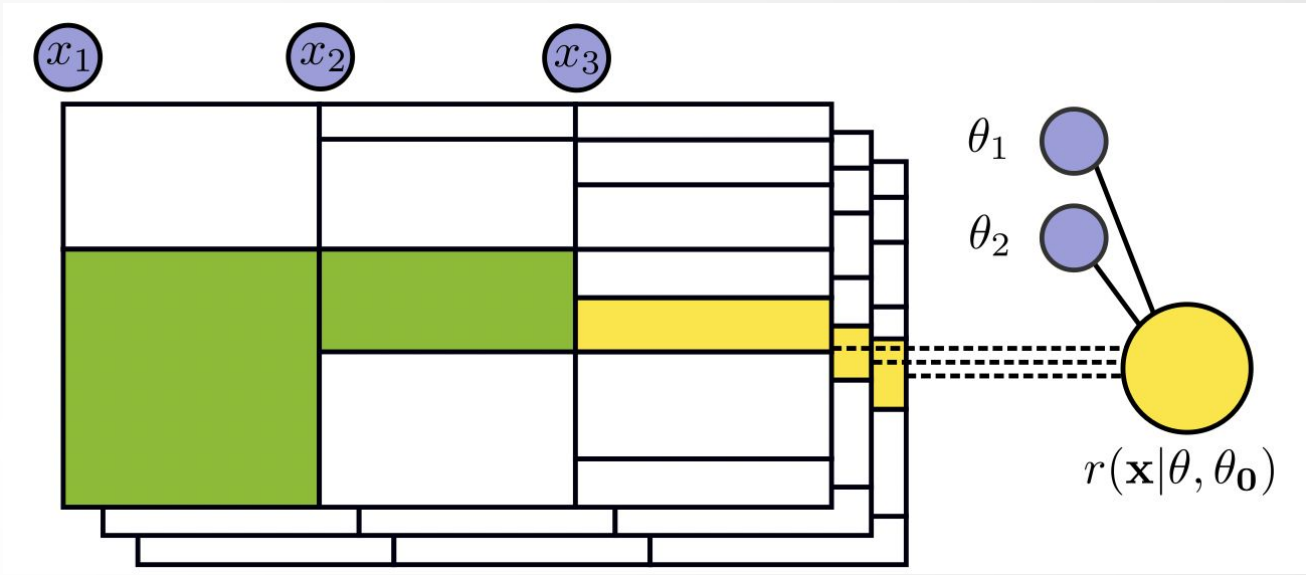




- Obtain parametrized classifiers with 20-40% improvements in 2D toy cases (NOT marginalized!)
- Easy to train with backgrounds
- Most gain when performing unbinned analysis (otherwise choices in binning must be made)
- Strong performance for marginalized limits in high dimensions

Network vs Trees





- Given a phase space region with EFT dependence: NN must select & predict
- In the Boosted Information Tree, the weak learner only selects
  - The prediction (F_j) is computed from the boxed events → integrates latent space
  - The regression problem is solved with computational complexity of classification
    - Speed advantage at high operator dimensions!

Summary

- Various algorithms predict SMEFT dependence, mostly capitalizing on weighted simulation
  - Provide NP-optimal observables for hypothesis tests
- Trees eliminate the latent space integration and are suitable to weight-based SMEFT predictions
- Parametrized classifiers "ask" for unbinned analysis -> Needs development in experiments

Get in touch with us for an in-person walkthrough

```
from MultiBoostedInformationTree import MultiBoostedInformationTree

bit = MultiBoostedInformationTree(
        training_features     = training_features,
        training_weights      = training_weights,
        base_points           = base_points,
        feature_names         = model.feature_names,
        **model.multi_bit_cfg
            )
bit.boost()
test_predictions = bit.vectorized_predict(test_features)
```

REFERENCES

- Madminer: Neural networks based likelihood-free inference & related techniques
  - K. Cranmer , J. Pavez , and G. Louppe [1506.02169]
    J. Brehmer, K. Cranmer, G. Louppe, J. Pavez [1805.00013] [1805.00020] [1805.12244]
    J. Brehmer, F. Kling, I. Espejo, K. Cranmer [1907.10621]
  - J. Brehmer, S. Dawson, S. Homiller, F. Kling, T. Plehn [1908.06980]
  - A. Butter, T. Plehn, N. Soybelman, J. Brehmer [2109.10414]
  - established many of the *main ideas* & *statistical interpretation* in various *NN applications*
- Weight derivative regression (A.Valassi) [2003.12853]
- Parametrized classifiers for SM-EFT: NN with quadratic structure
- S. Chen, A. Glioti, G. Panico, A. Wulzer [*JHEP* 05 (2021) 247] [arXiv:2308.05704]
- Boosted Information Trees: Tree algorithms & boosting
  - S. Chatterjee, S. Rohshap, N. Frohner, R. Schöfbeck, D. Schwarz [2107.10859], [2205.12976]
- ML4EFT R. Ambrosio, J. Hoeve, M. Madigan, J. Rojo, V. Sanz [2211.02058]

- All approaches are "SMEFT-specific ML" with differences mostly on the practical side

WH with Bkgs