Machine learning in HEP

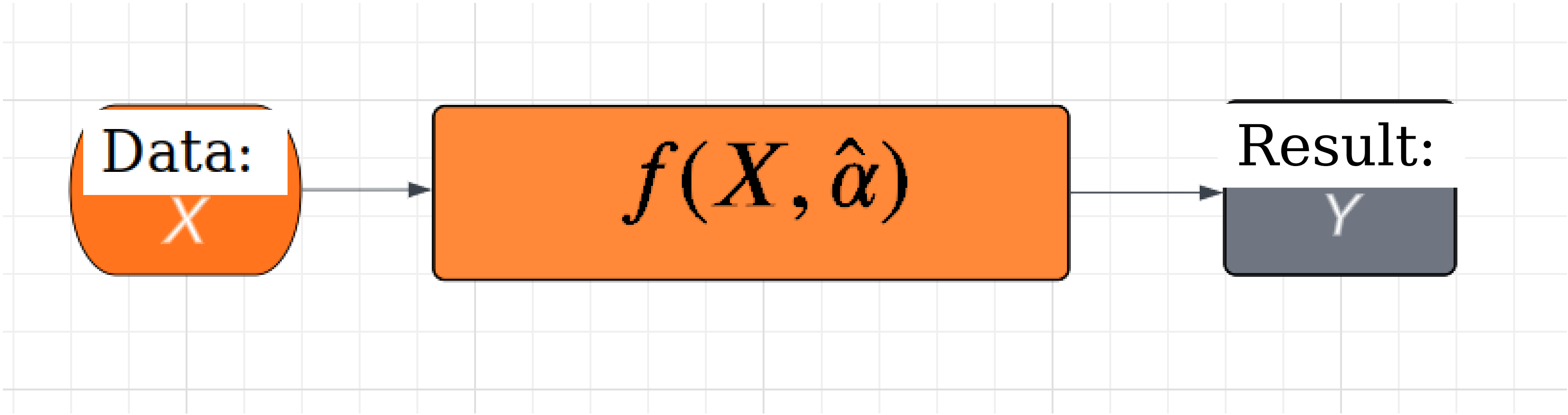
Artur Kalinowski

University of Warsaw

Function fitting

X: n dimensional input data

Y: k dimensional output data



Function fitting:

- **best case: N$\leq$2, k=1**
- **basis functions given *explicite***
- **expansion coefficients *could* be interpretable**

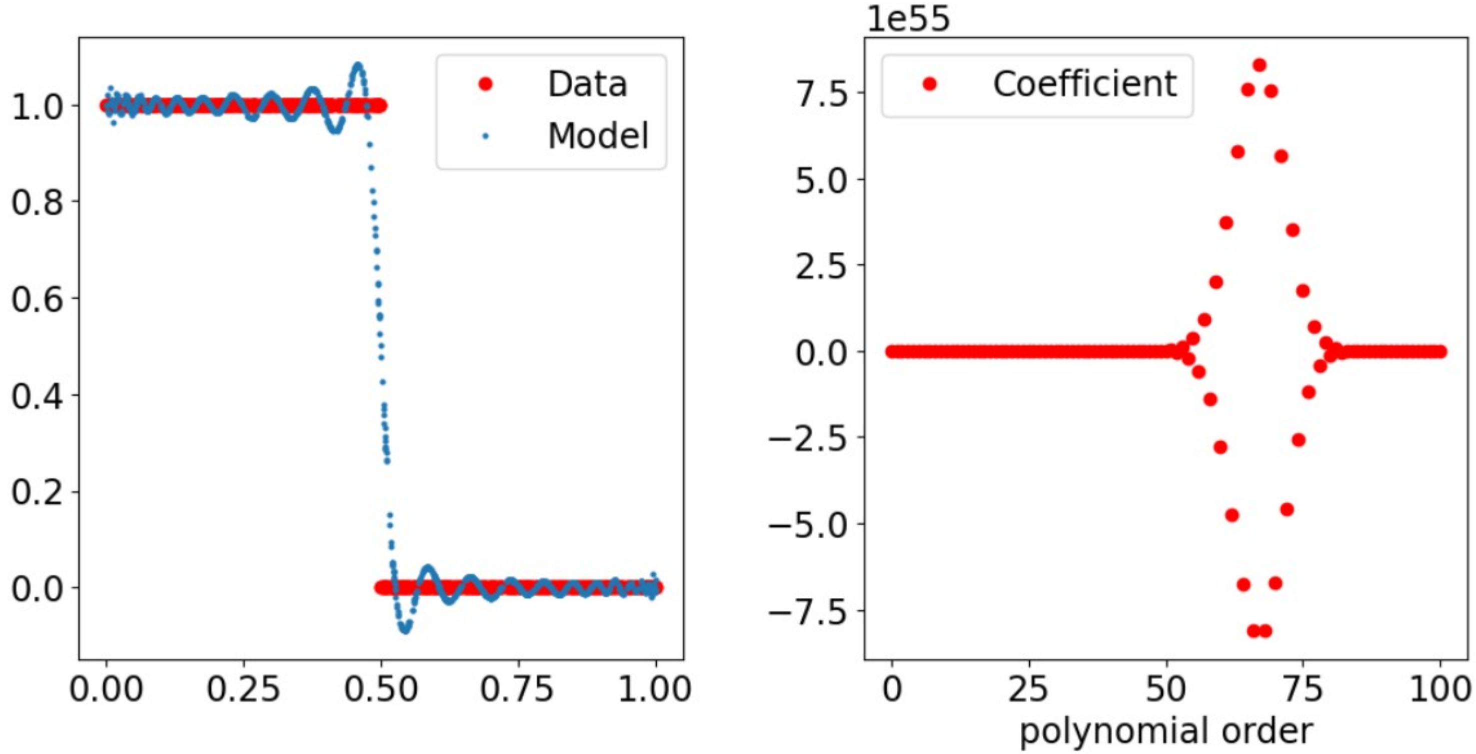
Universal approximator

Taylor theorem (J. Gregory, 1671):

Every, continuous, differentiable, function f(x): R → R can be approximated by a polynomial:

$$f(x,\theta) = \sum_{n=0}^{\infty} \theta_n x^n \simeq \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$$

Coefficients θ_i are derivatives of f(x).
In the case of unknown function ("data") coefficients can be found by a numerical procedure.
Usually...



Universal approximator

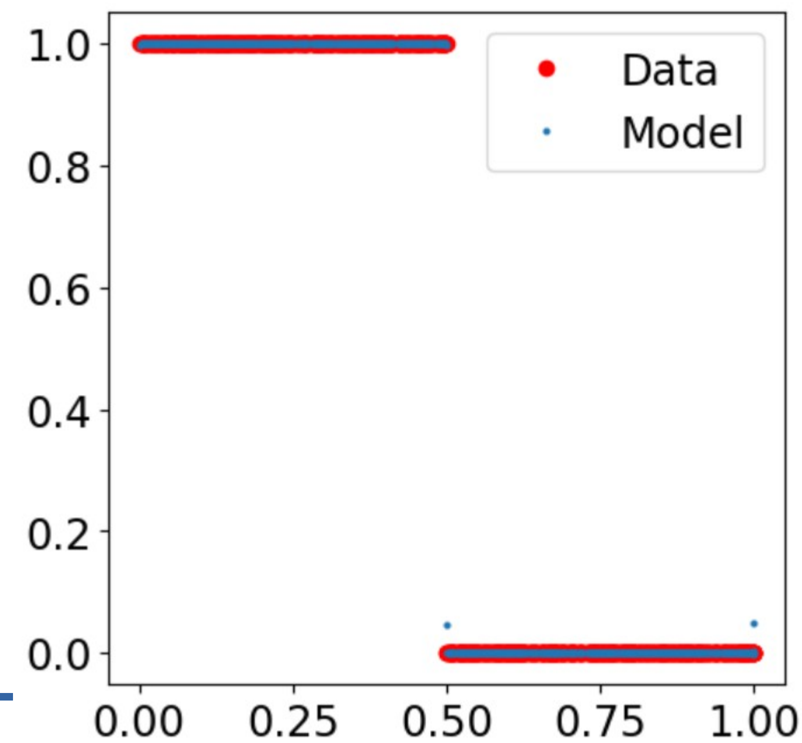
Fourier theorem (1807) :

Every continuous, differentiable, and periodic function f(x): R → R can be approximated in a basis of sines i cosines:

$$f(x,\theta) = \frac{\theta_{0,0}}{2} + \sum_{n=1}^{\infty} \theta_{0,n} \cos(n\omega x) + \theta_{1,n} \sin(n\omega x),\ \omega = \frac{2\pi}{T}$$

Coefficients θ_i can be found analytically.
In the case of unknown function (“data”) coefficients can be found by a numerical procedure.

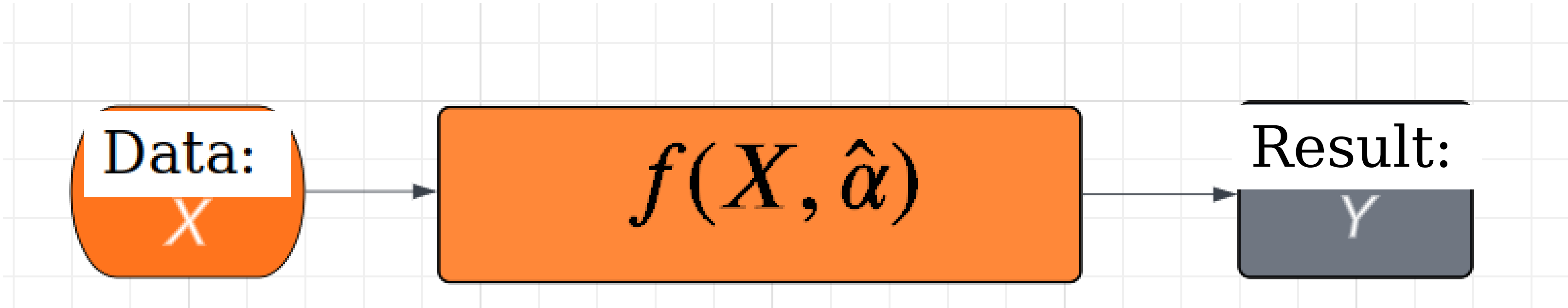
```
nMax = 25000
omega = 2*np.pi/1.0

Y_model = np.full_like(X,0.5)
for n in range(1,nMax+1, 2):
    Y_model += 2.0/np.pi*1/n*np.sin(omega*n*X)
```



Machine learning

X: n dimensional input space

Y: k dimensional output space



- **n~10^{l}, k~10^{m}, l,m~6**
- **basis functions defined *implicite* - through data flow – the network architecture**
- **expansion coefficients are uninterpretable**

Universal approximator

A sigmoid function: any non polynomial function fulfilling conditions:

$$A(\theta, x) = A(\sum_{i=1}^{n} \theta_i x_i + b) \qquad \lim_{x_i \to -\infty} A(x) \to 0 \; \lim_{x_i \to +\infty} A(x) \to 1$$

Universal approximator theorem (Cybenko, 1989):

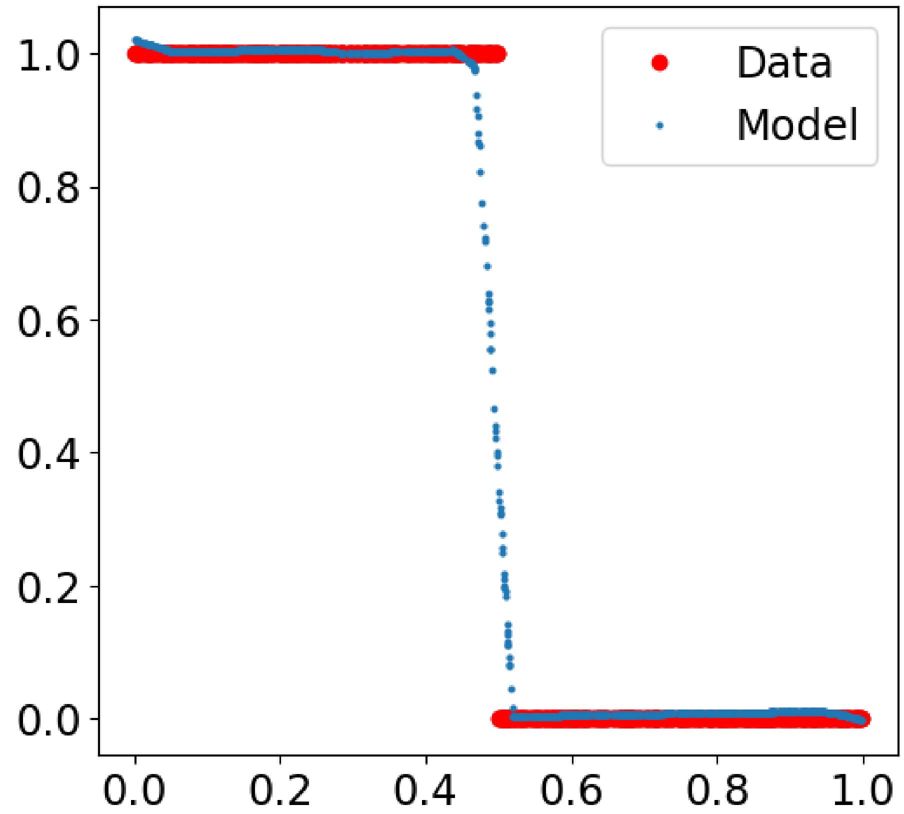
Every continuous function f(x) $R^n \to R$ can be approximated in basis of sigmoidal functions:

$$f(x, \theta) \simeq \sum_n w_n A(\theta_n, x)$$

Coefficients θ_i, w_i do not have in general an analytic form, but can be found using a numerical procedure

Universal approximator

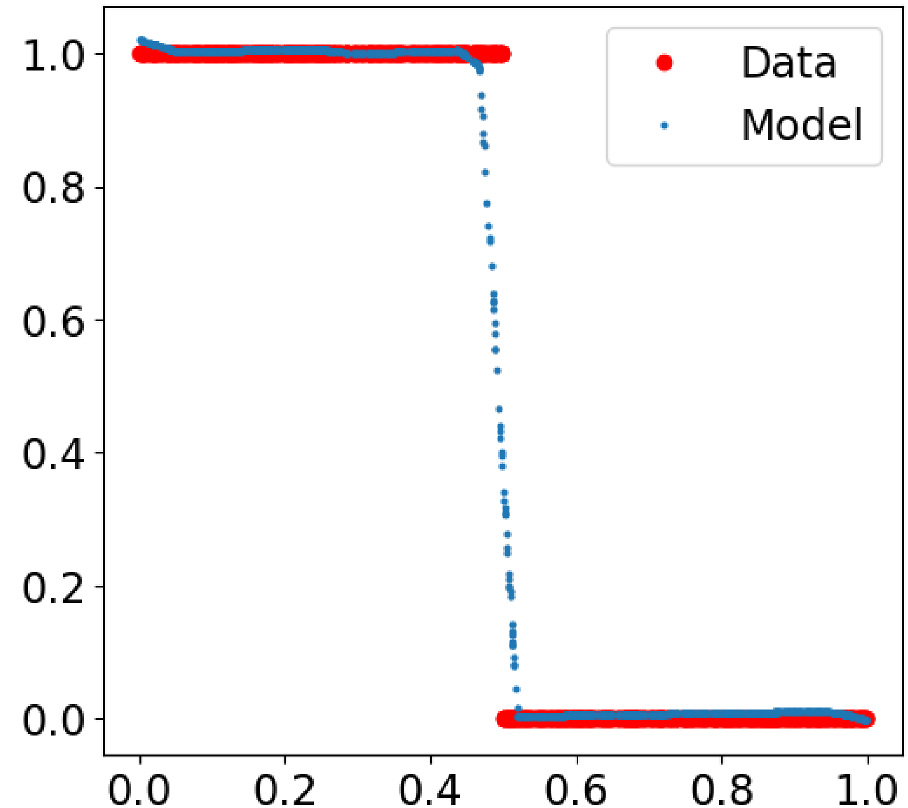
```python
nUnits = 32
inputs = tf.keras.Input(shape=(1,))
layer1 = tf.keras.layers.Dense(nUnits, activation='relu')(inputs)
layer2 = tf.keras.layers.Dense(nUnits, activation='relu')(layer1)

outputs = tf.keras.layers.Dense(1, activation='linear')(layer2)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss = 'mse')
###
history = model.fit(X,Y,epochs=150)
Y_model = model.predict(X)
```



Universal approximator

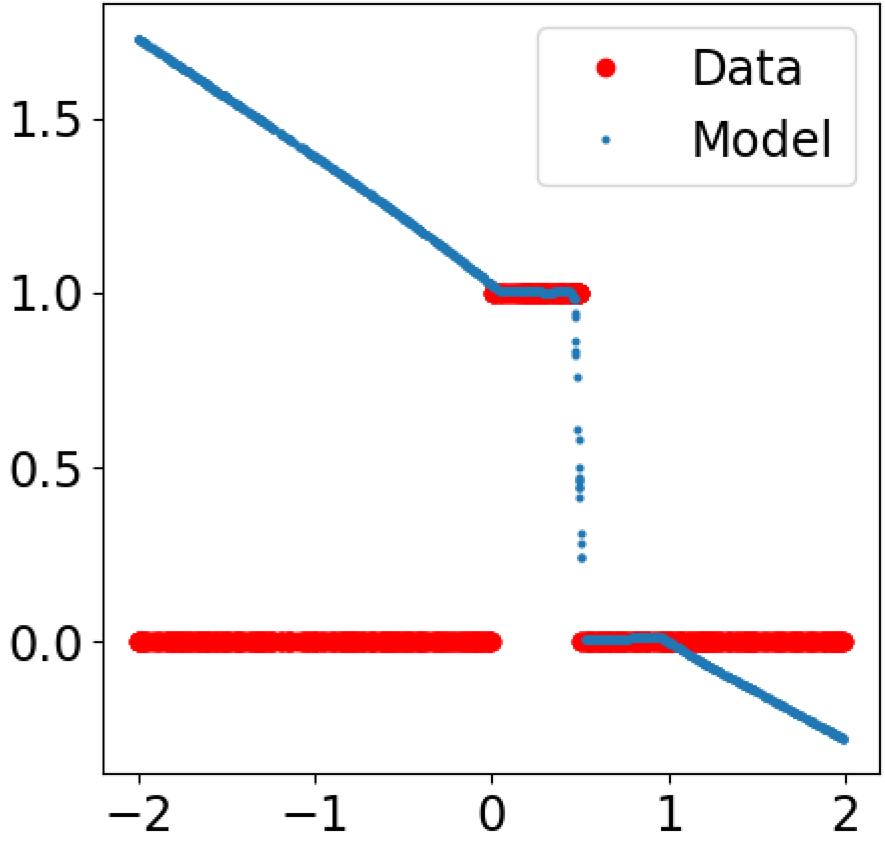
```python
nUnits = 32
inputs = tf.keras.Input(shape=(1,))
layer1 = tf.keras.layers.Dense(nUnits, activation='relu')(inputs)
layer2 = tf.keras.layers.Dense(nUnits, activation='relu')(layer1)

outputs = tf.keras.layers.Dense(1, activation='linear')(layer2)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss = 'mse')
###
history = model.fit(X,Y,epochs=150)
Y_model = model.predict(X)
```

An issue 0: it is quite hard to get an extremely precise predictions from ML.



Universal approximator


```python
nUnits = 32
inputs = tf.keras.Input(shape=(1,))
layer1 = tf.keras.layers.Dense(nUnits, activation='relu')(inputs)
layer2 = tf.keras.layers.Dense(nUnits, activation='relu')(layer1)

outputs = tf.keras.layers.Dense(1, activation='linear')(layer2)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss = 'mse')
###
history = model.fit(X,Y,epochs=150)
Y_model = model.predict(X)
```

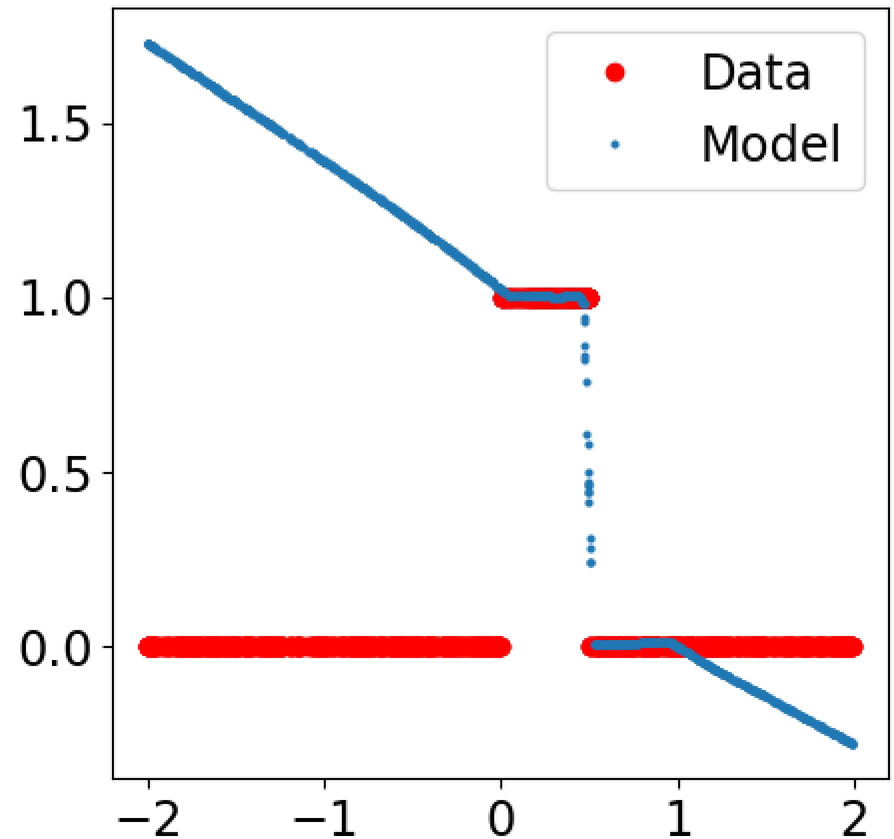


Universal approximator

```
nUnits = 32
inputs = tf.keras.Input(shape=(1,))
layer1 = tf.keras.layers.Dense(nUnits, activation='relu')(inputs)
layer2 = tf.keras.layers.Dense(nUnits, activation='relu')(layer1)

outputs = tf.keras.layers.Dense(1, activation='linear')(layer2)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss = 'mse')
###
history = model.fit(X,Y,epochs=150)
Y_model = model.predict(X)
```

An issue 1: behavior for unseen data is hard (impossible?) to predict.

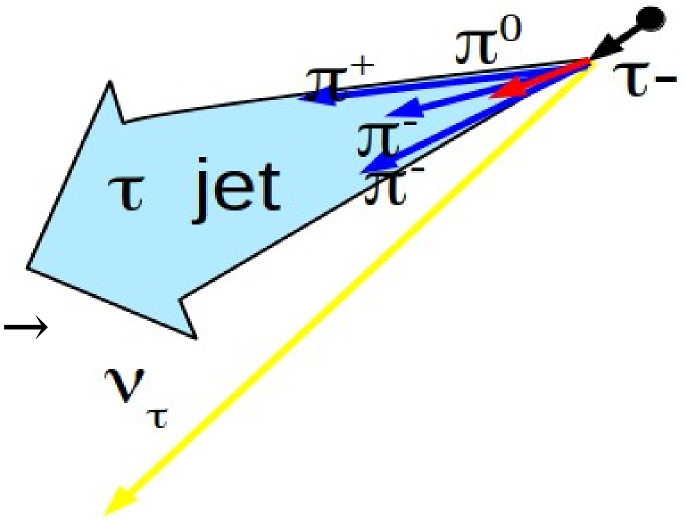
If the model is trained on simulated input the real data might contain unseen parts of the input space



Example: hadronic τ decay identification

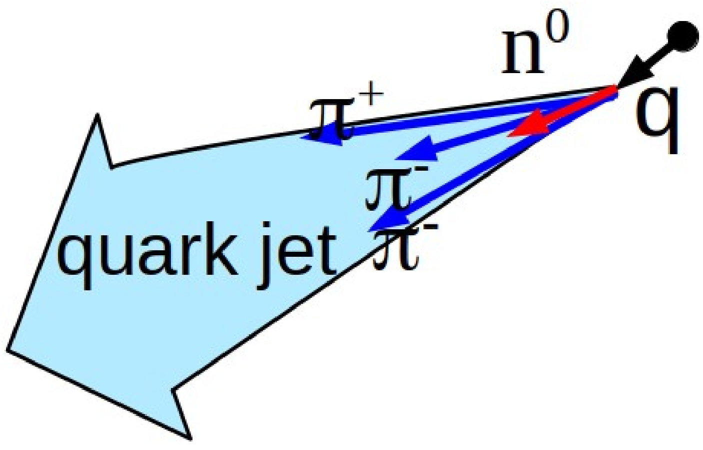
CMS Collaboration,
JINST 17 (2022) P07023

The task: jet categorisation:
hadronic τ decay/electron/muon/jet



Context:

- τ is the heaviest lepton, with short life time → only decay products are observed in the detector
- τ decays to 1/3 hadrons + neutrinos in 65% cases
- τ decays look very similar to small jets originating from quarks and gluons



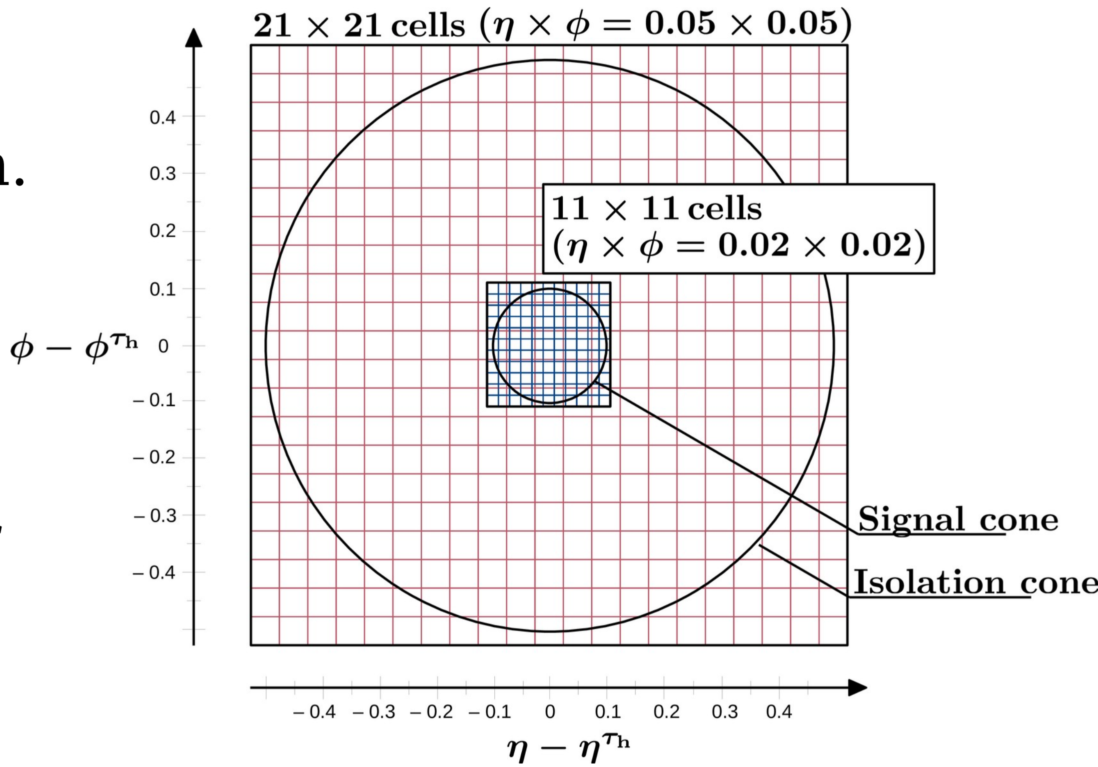
Example: hadronic τ decay identification

CMS Collaboration, JINST 17 (2022) P07023

Input data:

- *low level:*
  highest p_T particle of given type:
  e/μ/charged hadron.neutral hadron
  properties from each of cells
  around the τ candidate direction.
  About **20** features per particle

- *high level:*
  47 human invented features
  derived from particle properties



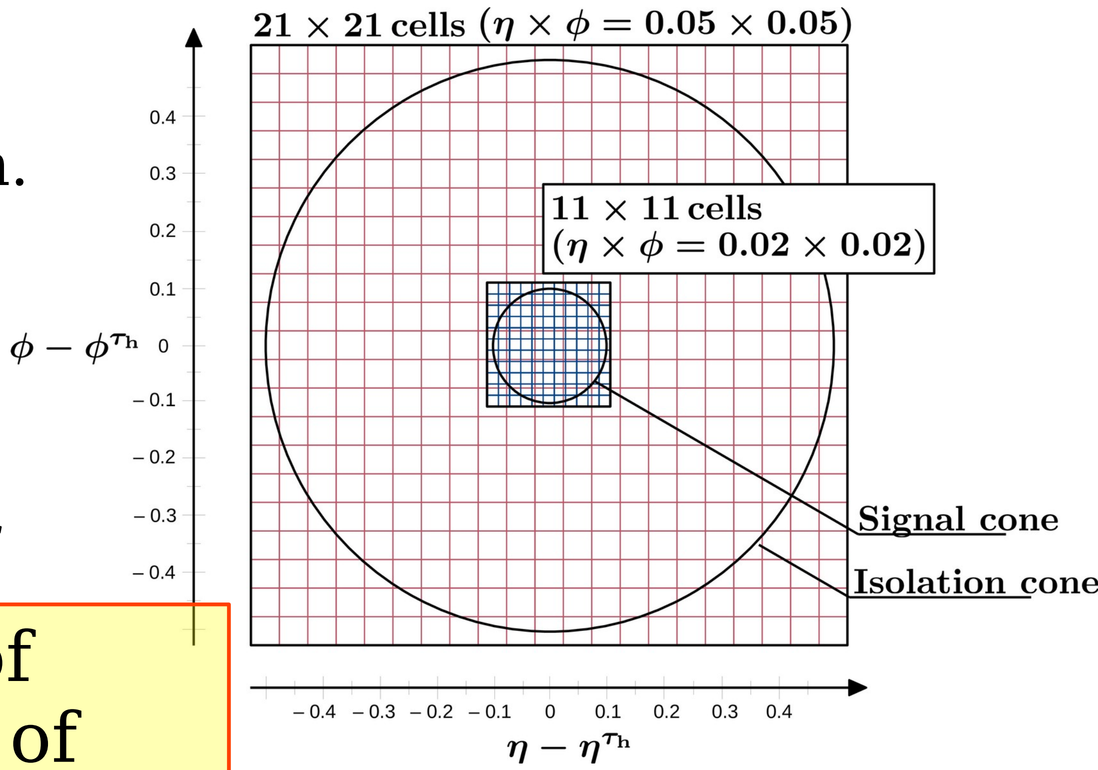
Example: hadronic τ decay identification

CMS Collaboration, JINST 17 (2022) P07023

Input data:

- *low level:*
  highest p_T particle of given type:
  e/μ/charged hadron.neutral hadron
  properties from each of cells
  around the τ candidate direction.
  About **20** features per particle

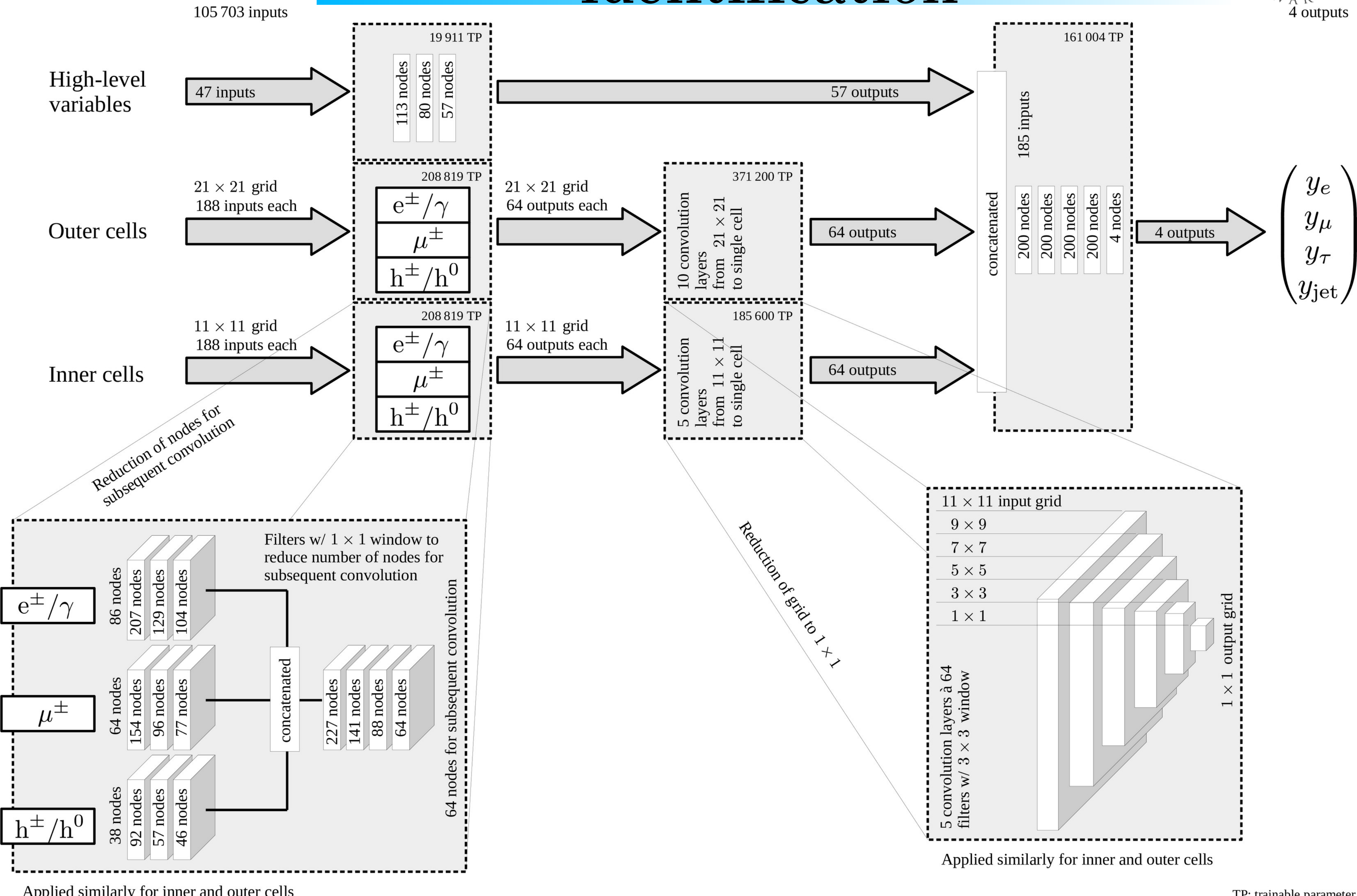
- *high level:*
  47 human invented features
  derived from particle properties



Observation 0: injection of domain knowledge in form of hand crafted features training improves stability and convergence.

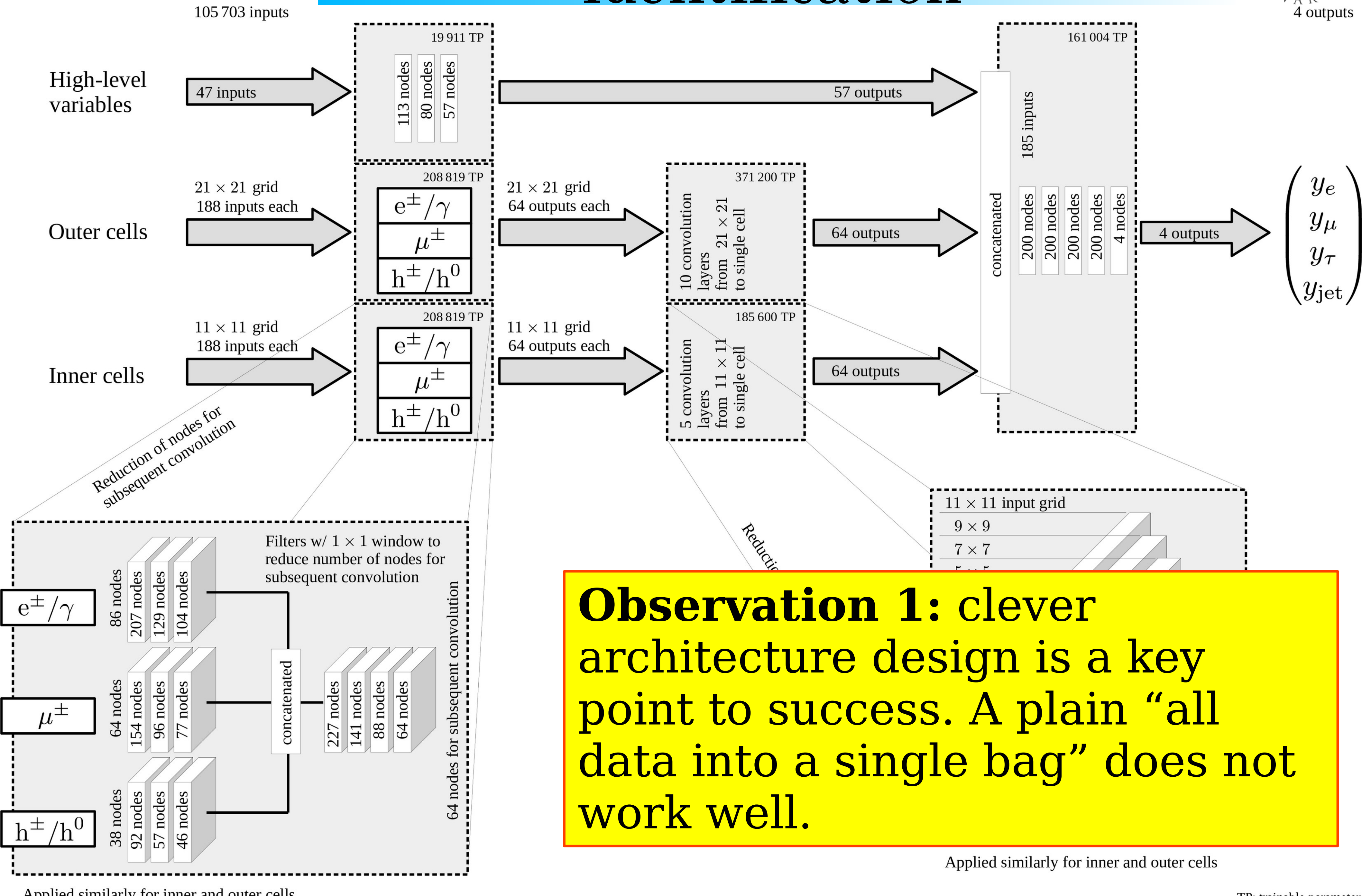
Example: hadronic τ decay identification

CMS Collaboration,
JINST 17 (2022) P07023



Example: hadronic τ decay identification

CMS Collaboration, JINST 17 (2022) P07023



Observation 1: clever architecture design is a key point to success. A plain "all data into a single bag" does not work well.

Example: hadronic τ decay identification

CMS Collaboration, JINST 17 (2022) P07023

$$L(\mathbf{y}^{\mathrm{true}}, \mathbf{y}; \boldsymbol{\kappa}, \gamma, \boldsymbol{\omega}) = \underbrace{\kappa_\tau \, H_\tau(\mathbf{y}^{\mathrm{true}}, \mathbf{y}; \boldsymbol{\omega})}_{\text{(a) Separation of all } \alpha} + \underbrace{\left(\kappa_{\mathrm{e}} + \kappa_\mu + \kappa_{\mathrm{jet}}\right) \overline{F}_{\mathrm{cmb}}(1 - y_\tau^{\mathrm{true}}, 1 - y_\tau; \gamma_{\mathrm{cmb}})}_{\text{(b) Focused separation of e, } \mu \text{, jet from } \tau_{\mathrm{h}}}$$

$$+ \underbrace{\kappa_F \sum_{i \in \{\mathrm{e}, \mu, \mathrm{jet}\}} \kappa_i \, \hat{\theta}(y_\tau - 0.1) \, \overline{F}_i(y_i^{\mathrm{true}}, y_i; \gamma_i)}_{\text{(c) Focused separation of } \tau_{\mathrm{h}} \text{ from e, } \mu \text{, jet for } y_\tau > 0.1}.$$

Focal loss function:

$$F(y^{\mathrm{true}}, y; \gamma) = -y^{\mathrm{true}} \, (1 - y)^\gamma \log(y) \qquad \overline{F}(y^{\mathrm{true}}, y; \gamma) = \mathcal{N} \, F(y^{\mathrm{true}}, y; \gamma),$$

Example: hadronic τ decay identification

CMS Collaboration,
JINST 17 (2022) P07023

$$L(\mathbf{y}^{\mathrm{true}},\mathbf{y};\boldsymbol{\kappa},\boldsymbol{\gamma},\boldsymbol{\omega}) = \underbrace{\kappa_\tau\, H_\tau(\mathbf{y}^{\mathrm{true}},\mathbf{y};\boldsymbol{\omega})}_{\text{(a) Separation of all } \alpha} + \underbrace{\left(\kappa_{\mathrm{e}} + \kappa_\mu + \kappa_{\mathrm{jet}}\right)\overline{F}_{\mathrm{cmb}}(1-y_\tau^{\mathrm{true}}, 1-y_\tau;\gamma_{\mathrm{cmb}})}_{\text{(b) Focused separation of e, } \mu \text{, jet from } \tau_{\mathrm{h}}}$$

$$+ \underbrace{\kappa_F \sum_{i\in\{\mathrm{e},\mu,\mathrm{jet}\}} \kappa_i\, \hat{\theta}(y_\tau - 0.1)\, \overline{F}_i(y_i^{\mathrm{true}}, y_i;\gamma_i)}_{\text{(c) Focused separation of } \tau_{\mathrm{h}} \text{ from e, } \mu \text{, jet for } y_\tau > 0.1}.$$

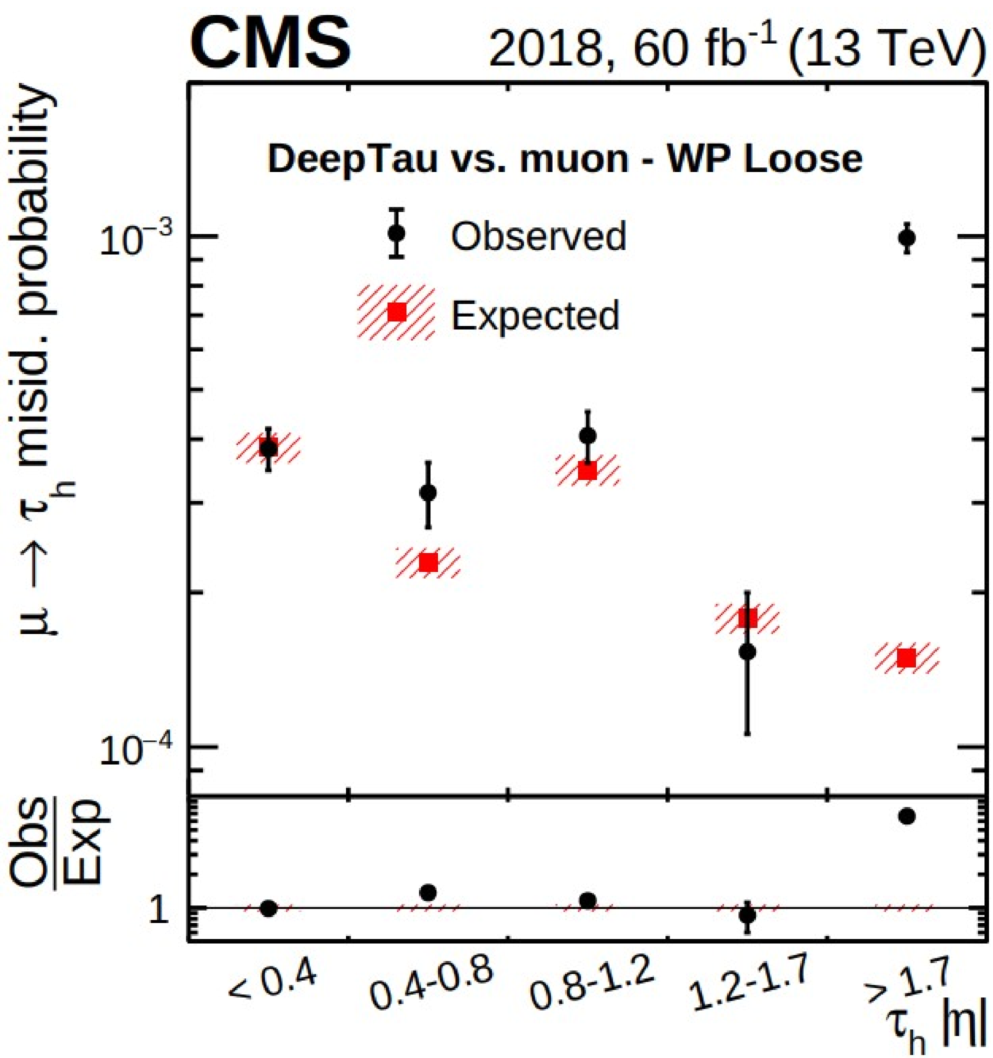
Observation 2: clever loss function improves model performance in intermediate region of efficiency (aka. recall.)

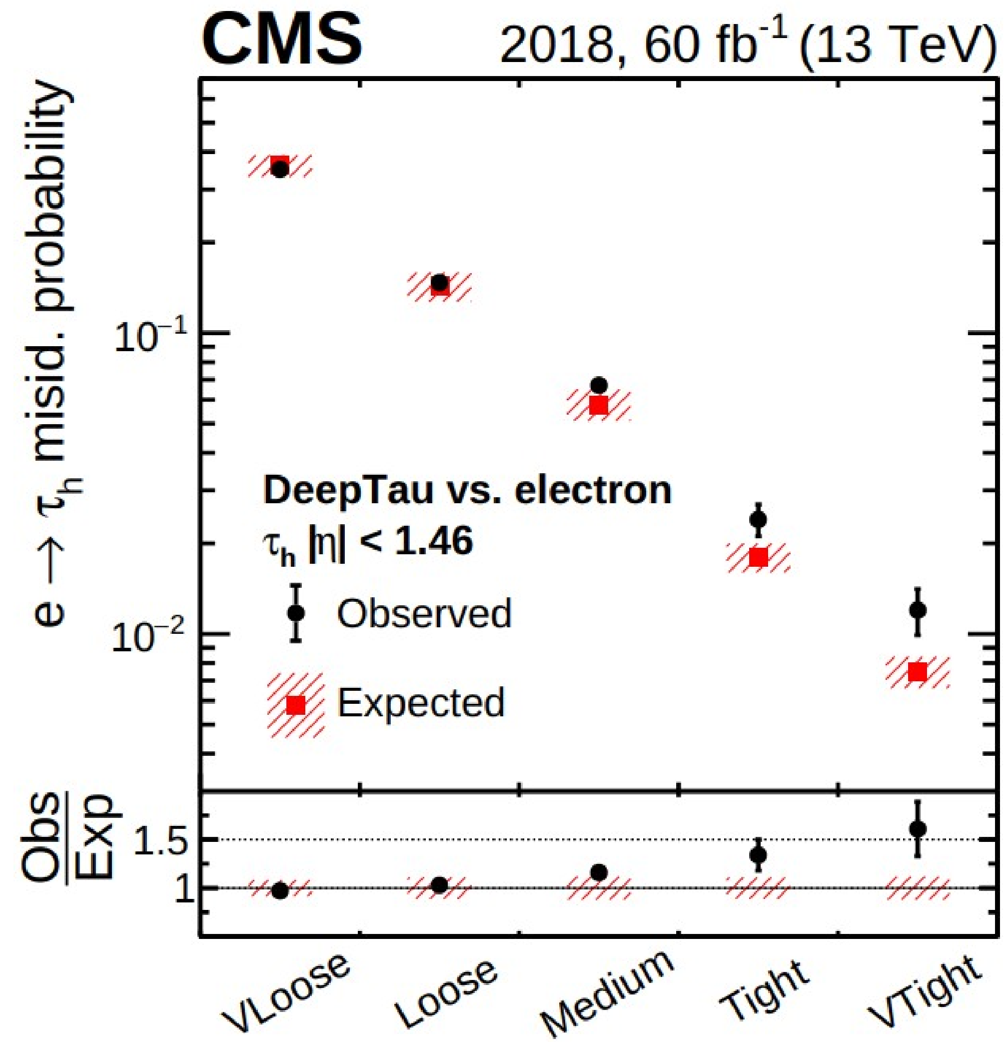
Focal loss function:

$$F(y^{\mathrm{true}},y;\gamma) = -y^{\mathrm{true}}\,(1-y)^\gamma \log(y) \qquad \overline{F}(y^{\mathrm{true}},y;\gamma) = \mathcal{N}\, F(y^{\mathrm{true}},y;\gamma),$$

Example: hadronic τ decay identification

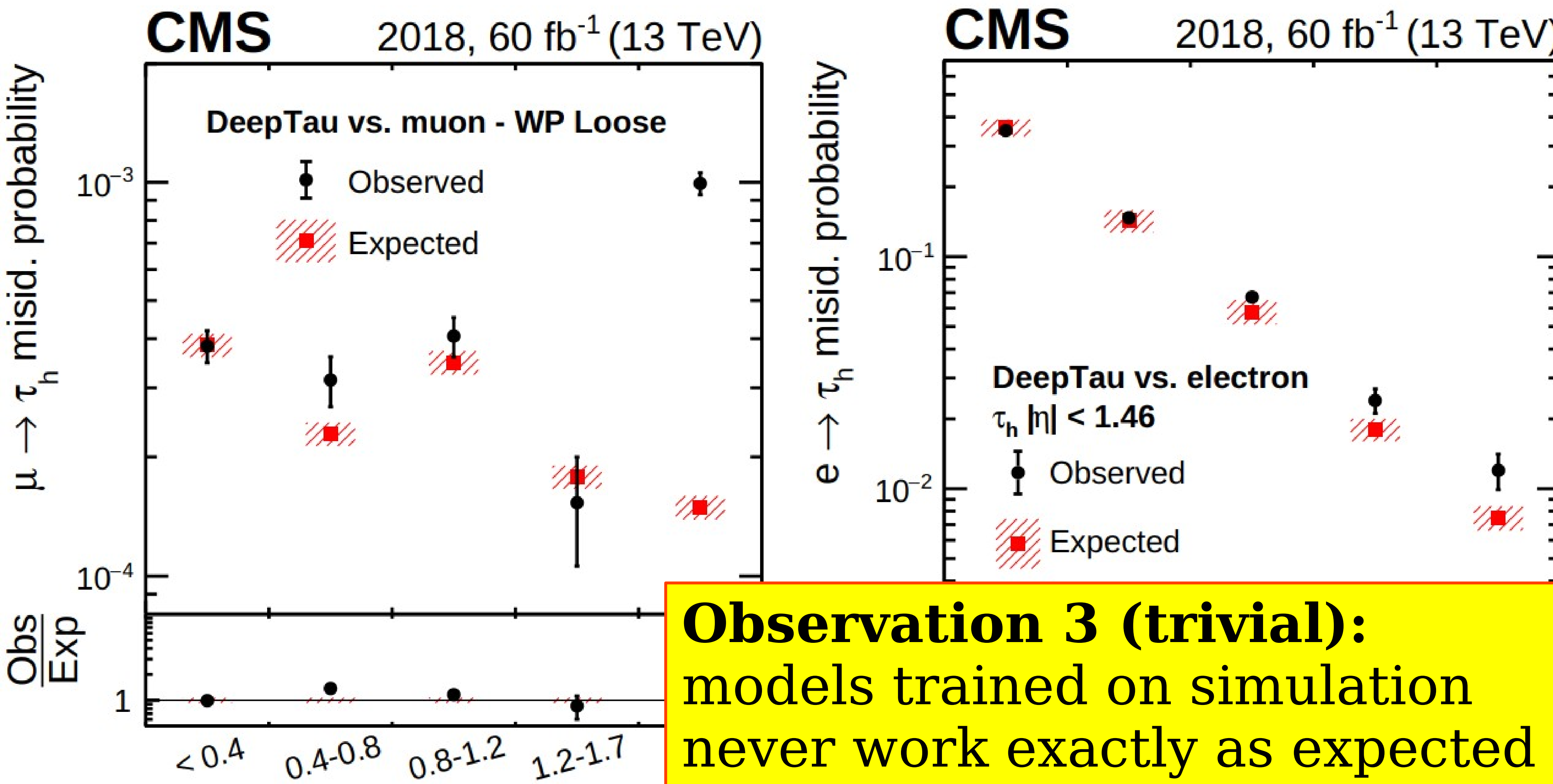
CMS Collaboration,
JINST 17 (2022) P07023





Example: hadronic τ decay identification

CMS Collaboration, JINST 17 (2022) P07023



Observation 3 (trivial): models trained on simulation never work exactly as expected on a real data. The more rare cases the harder to follow the expectations.

Adversarial Attack

Explaining and Harnessing Adversarial Examples
arXiv:1412.6572

$\boldsymbol{x}$

"panda"
57.7% confidence

$+$? $=$

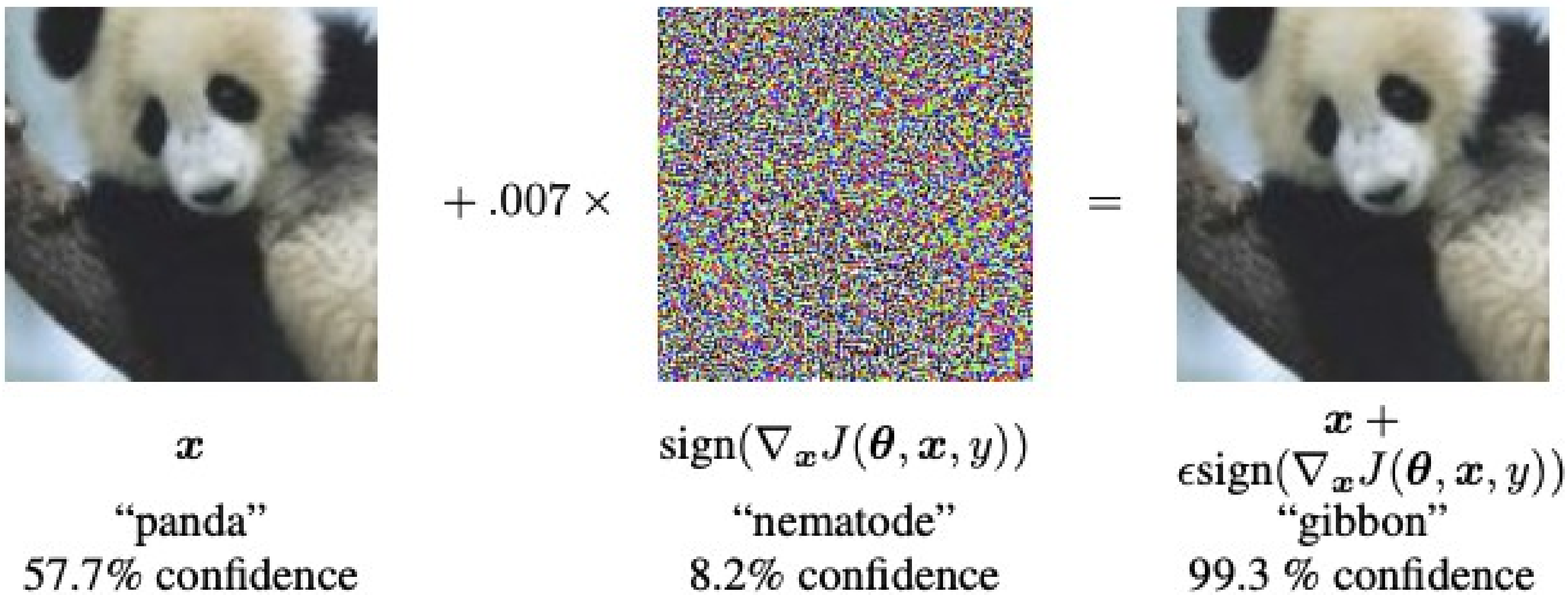
$\boldsymbol{x} +$?

"gibbon"
99.3 % confidence

Small and smart distortion to input data can lest to dramatic change in model response. This effect is exploited by **bad people** to perform Adversarial Attacks.

Adversarial Attack

Explaining and Harnessing Adversarial Examples
arXiv:1412.6572



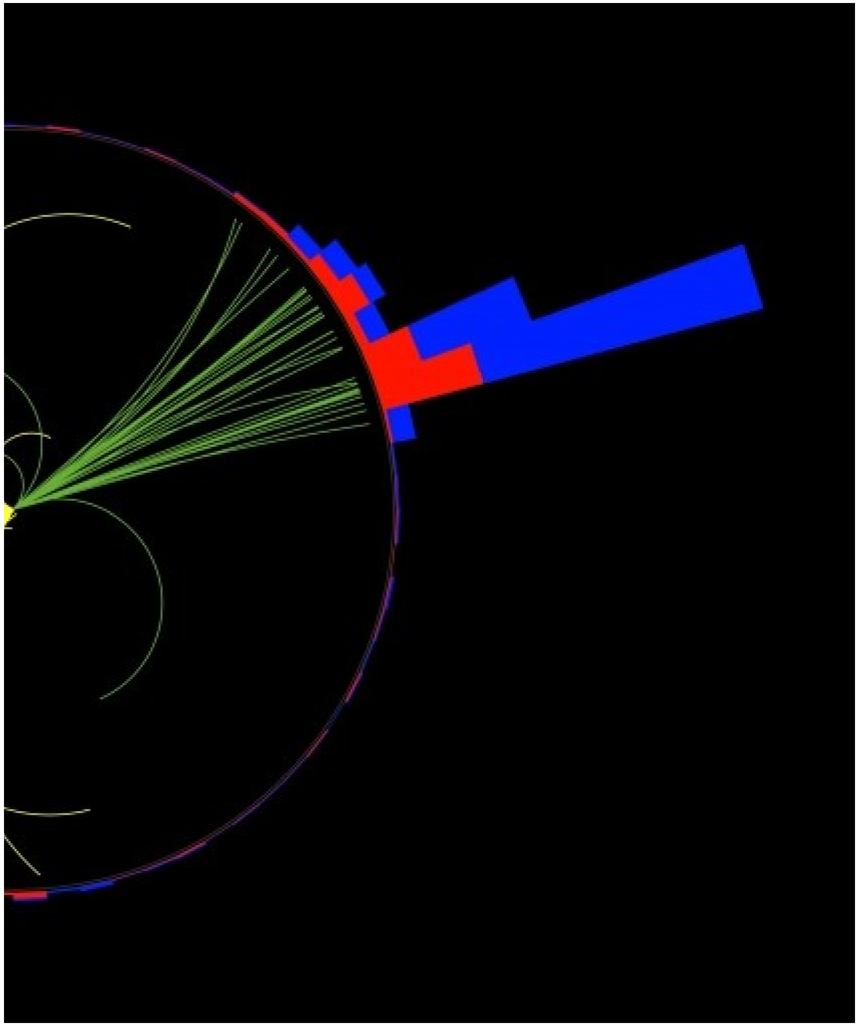
Small and smart distortion to input data can lest to dramatic change in model response.This effect is exploited by **bad people** to make an Adversarial Attacks.

Good people can use is to increase robustness of a model to features distortions

Example: jet identification

Comput Softw Big Sci 6 (2022) 15

The task: improve network robustness against Data – Monte Carlo differences.



https://cms.cern/news/machining-jets

Context:

- jet classification:
  - light quark or gluon jets
  - c-quark jets
  - b-quark jets

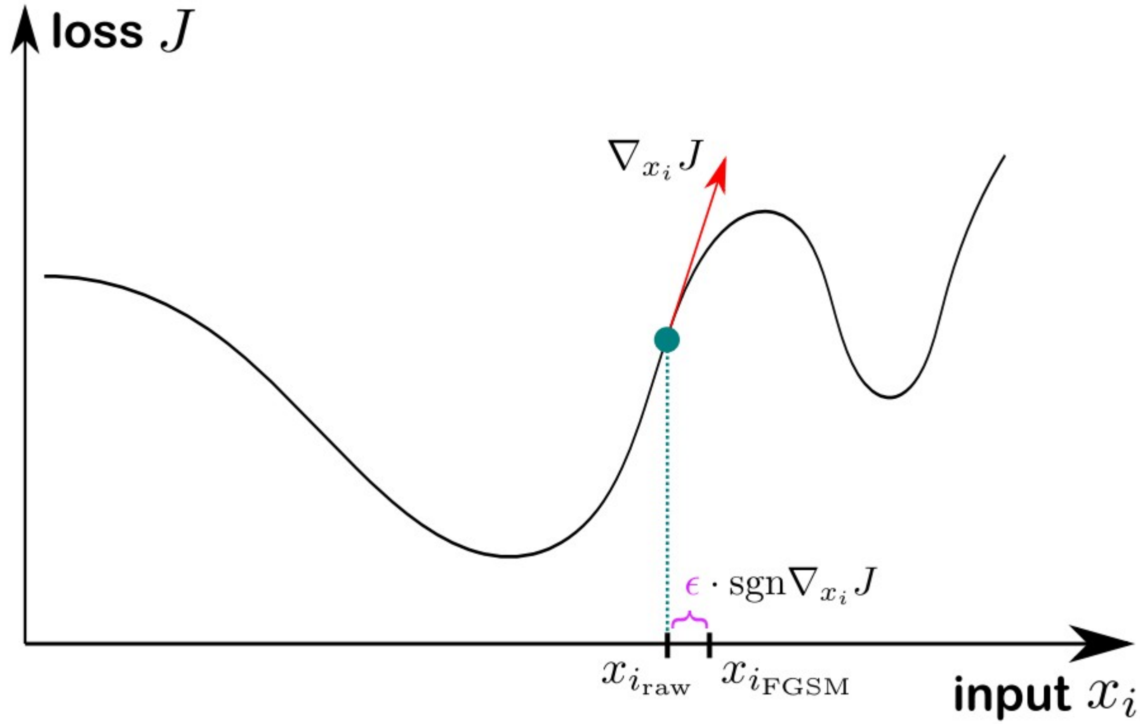
- features:
  - *low level:* 21 parameters of 6 tracks within a jet
  - *high level:* 14 jet properties

Fast Gradient Sign Adversarial input Modification (FGSM)

Comput Softw Big Sci 6 (2022) 15

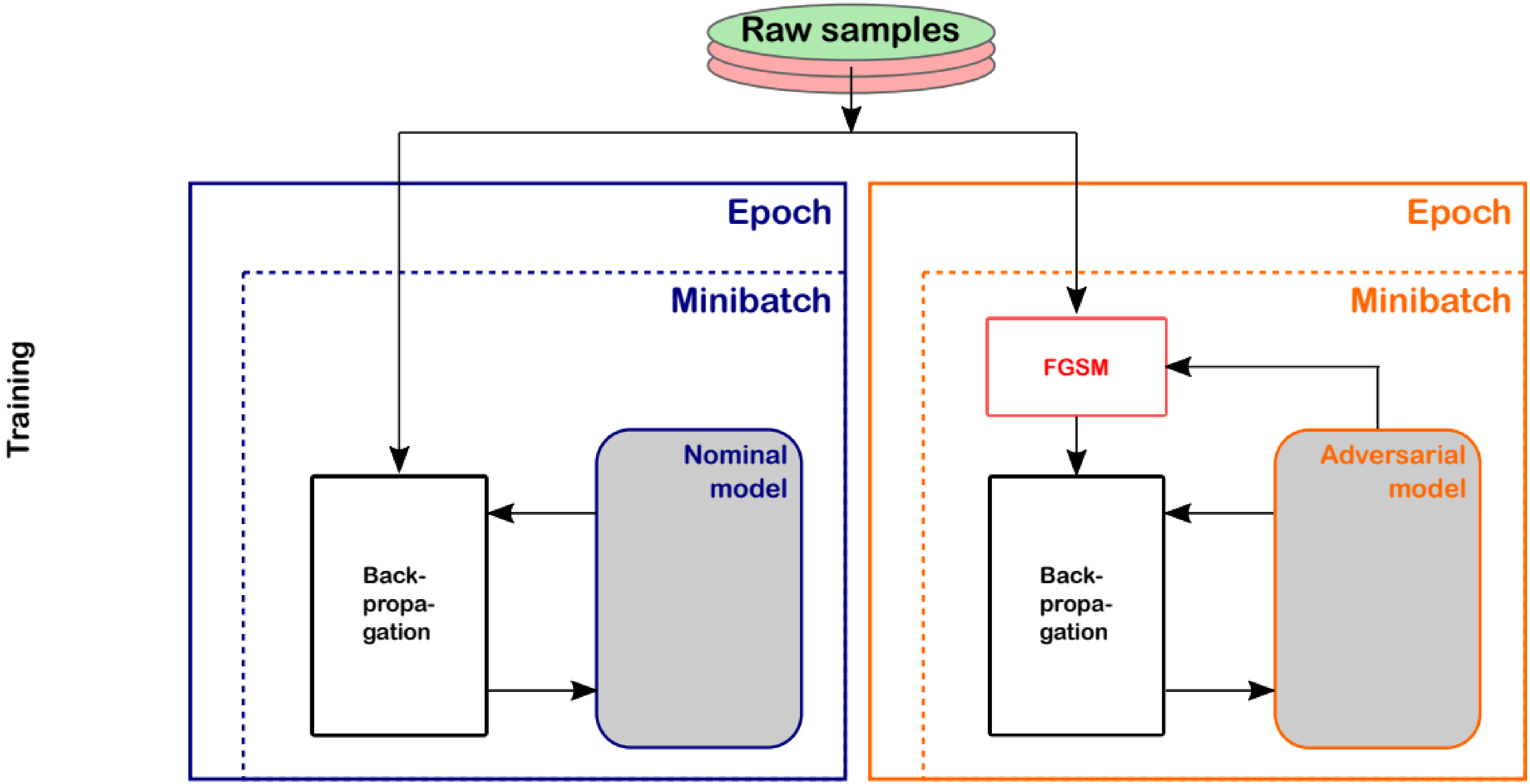
$$x_{\text{FGSM}} = x_{\text{raw}} + \epsilon \cdot \text{sgn}\left(\nabla_{x_{\text{raw}}} J(x_{\text{raw}}, y)\right)$$

$\varepsilon = 0.01$ used during the training phase



Fast Gradient Sign Adversarial input Modification (FGSM)

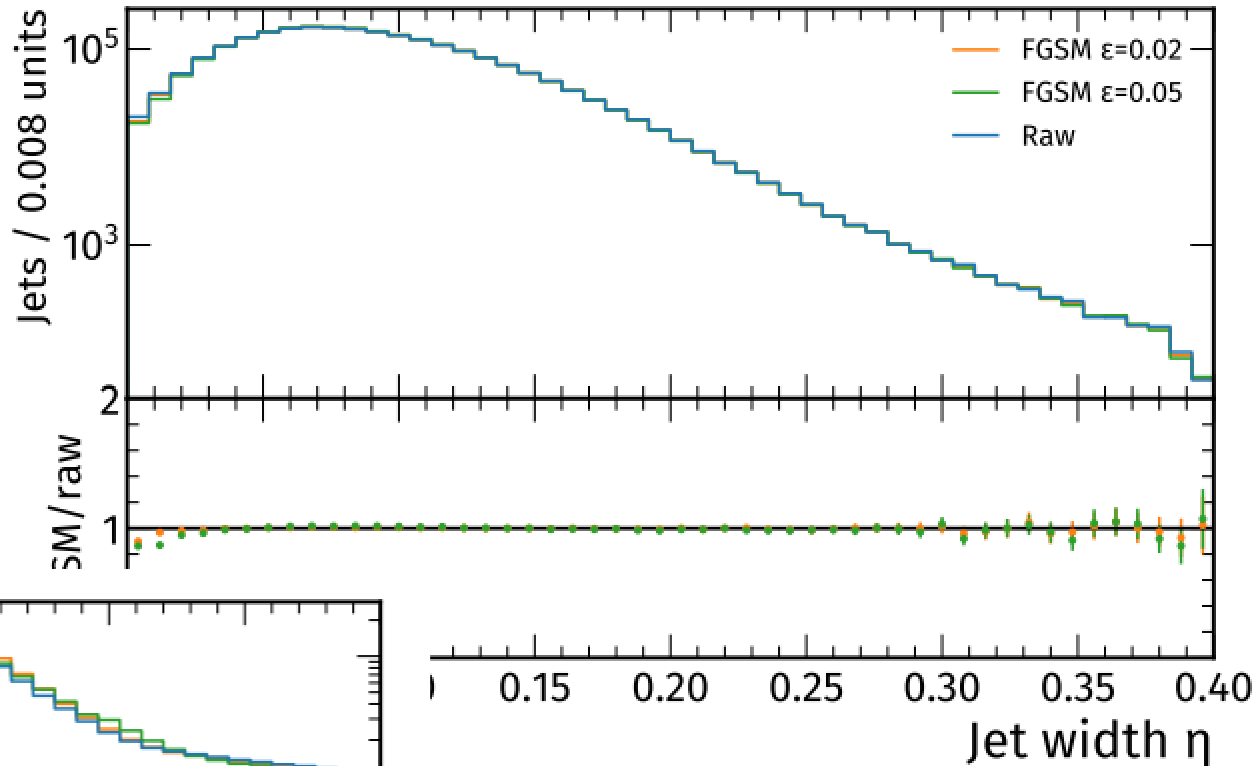
Comput Softw Big Sci 6 (2022) 15

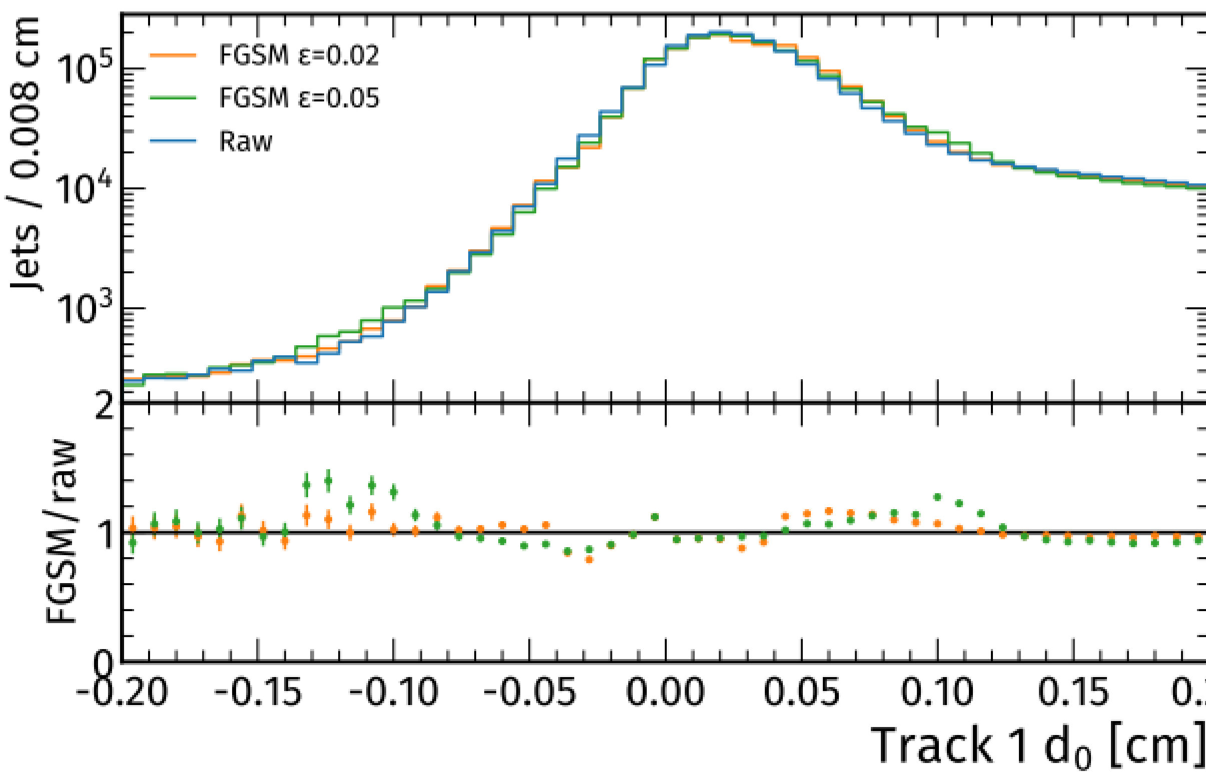


Fast Gradient Sign Adversarial input Modification (FGSM)

Comput Softw Big Sci 6 (2022) 15

The variable distribution distortion from FGSM is quite small

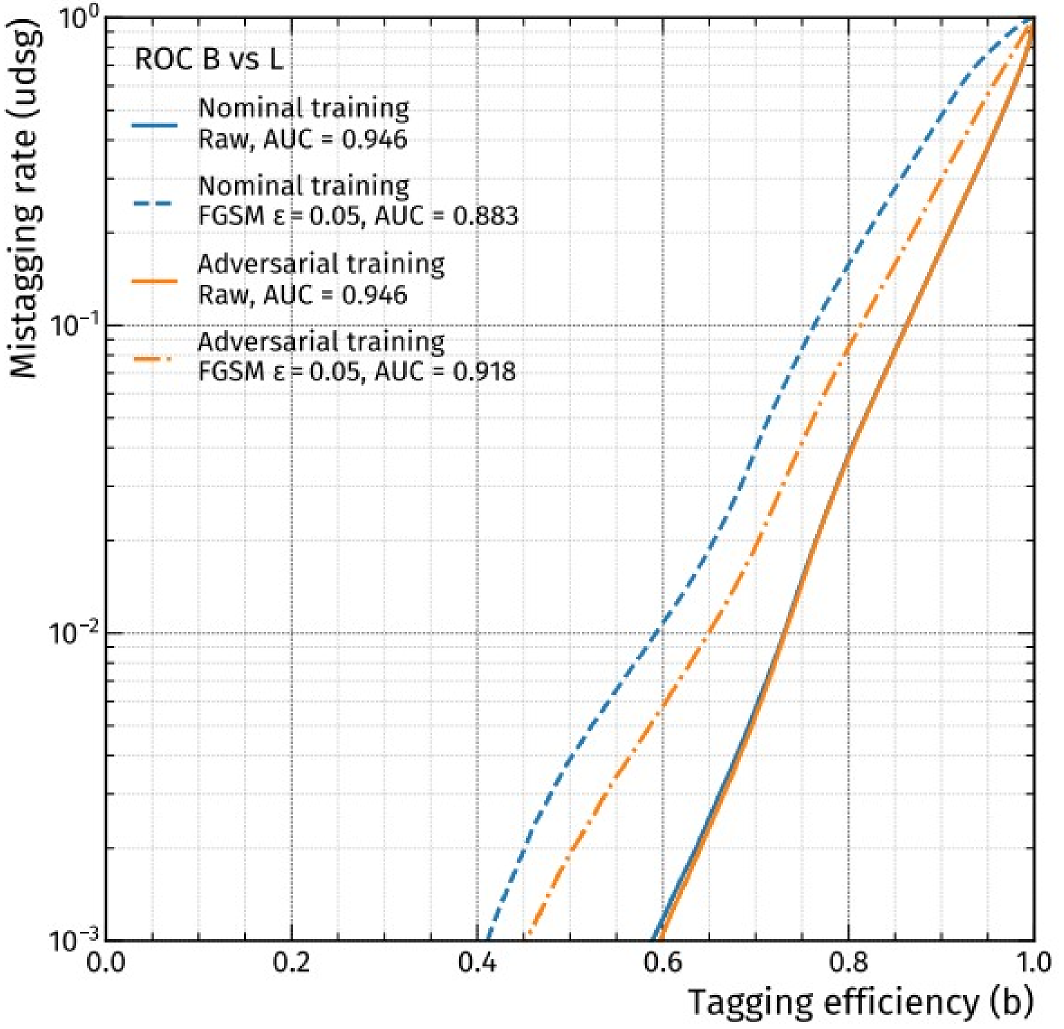




Fast Gradient Sign Adversarial input Modification (FGSM)

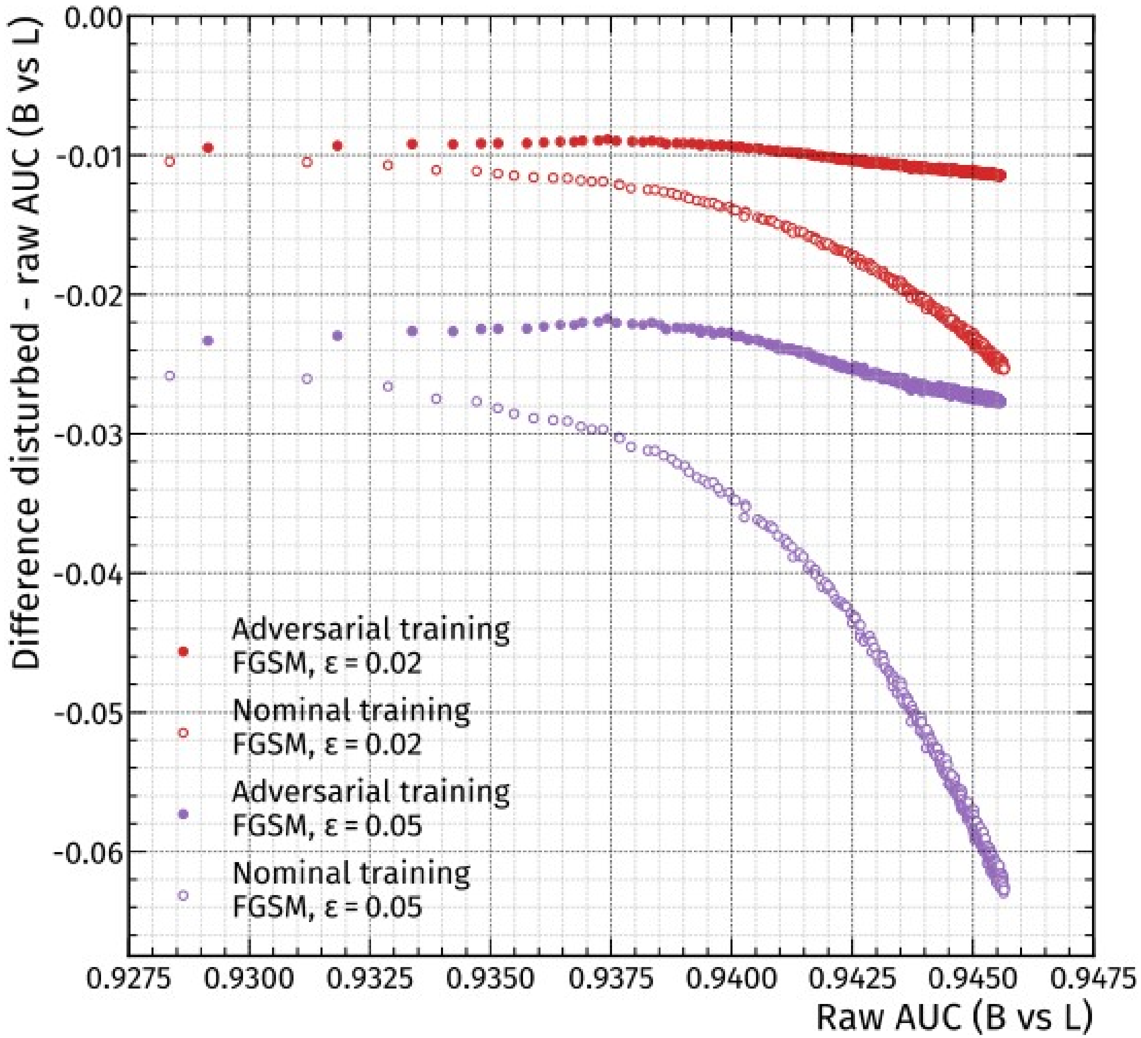
Comput Softw Big Sci 6 (2022) 15

FGSM acts as a regularisation, which does not affect the performance of the model, but improves it generalization capabilities.



Fast Gradient Sign Adversarial input Modification (FGSM)

Comput Softw Big Sci 6 (2022) 15



FGSM regularisation, works better than early stopping refularisation.

Summary

- Machine Learning is not a magic ward – this is yet another technology

but keep in mind what Arthur C. Clare said: “Any sufficiently advanced technology is indistinguishable from magic.”

- „ordinary” ML users should concentrate on creative problem formulation instead of attempting to invent a new, complicated, architecture

- we should remember about unavoidable difference between features distributions used for training (MC) and inference (Data)

FACULTY OF PHYSICS UNIVERSITY OF WARSAW

Living ML review for Particle Physics

HEP ML Living Review

Search

GitHub ☆316 95

Home Recent About Contribute Code of conduct Resources Cite Us

A Living Review of Machine Learning for Particle Physics

Modern machine learning techniques, including deep learning, is rapidly being applied, adapted, and developed for high energy physics. The goal of this document is to provide a nearly comprehensive list of citations for those developing and applying these approaches to experimental, phenomenological, or theoretical analyses. As a living document, it will be updated as often as possible to incorporate the latest developments. A list of proper (unchanging) reviews can be found within. Papers are grouped into a small set of topics to be as useful as possible. Suggestions are most welcome.

Table of contents

Backup slides