universität wien
Fakultät für Physik

VDS
VIENNA - DOCTORAL - SCHOOL - PHYSICS

ESI
Erwin Schrödinger International Institute for Mathematics and Physics

Machine Learning in Particle Physics (II) – Correlations and Systematics

Wolfgang Waltenberger
ÖAW and Uni Wien

ESI Winter School, Vienna,
Feb 10 - 20, 2020

HEPHY
INSTITUT FÜR HOCHENERGIEPHYSIK

Disclaimer

- Gregor, Lisa and I are all (mostly) LHC physicists. Within these lectures we will thus keep a strong focus on the context of the LHC.

- You may find all my material for this winterschool at

  https://github.com/WolfgangWaltenberger/winterschool →

- We will use google's colab for the tutorial
- On the github page given above of the repo you will find a link to our google colab instance

Syllabus

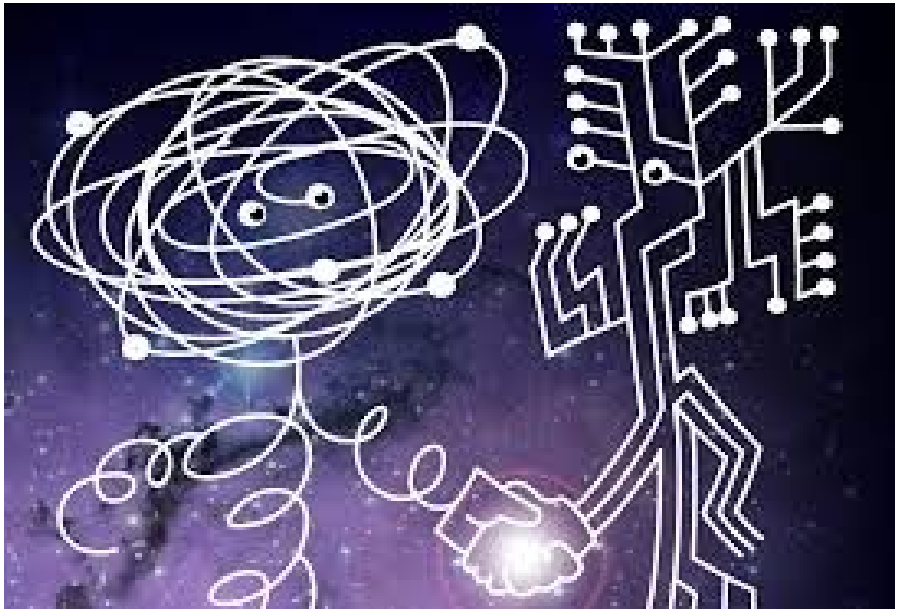
For today and tomorrow, we aim to put our focus on

- uncertainties
- decorrelating variables
- generative models for event simulation
- outlier detection

Wednesday	Thursday
9:00 Introductory Lecture, Detailling our Use Cases and Problems	**9:00** Event Simulation and Outlier Detection (I)
11:00 Uncertainties, Decorrelating Variables	**11:00** Event Simulation and Outlier Detection (II)
14:00 Tutorial, Uncertainties	**14:00** Tutorial, Event Simulation and Outlier Detection

Particle Physics and Machine Learning – Lecture II:

Correlations, and Systematics



Uncertainties: ML jargon / physics jargon

- **Aleatoric uncertainty:** (“alea” in latin is “dice”)

  All uncertainties related to the input data, e.g. sensor noise. In particle physics we often have a good understanding of these.

- **Epistemic uncertainty:** (epistemic = “related to knowledge”)

  Uncertainties on the network weights, i.e. uncertainties that are introduced with the network – and the fact that it is trained only with finite data.

- I propose that for applications we call them “errors on the inputs”, “errors due to the network”, respectively.

- Epistemic uncertainties are systematic uncertainties in our statistical analysis

Uncertainties: ML jargon / physics jargon

- **Aleatoric uncertainty:** (“alea” in latin is “dice”)

  All uncertainties related to the input data, e.g. sensor noise. In particle physics we often have a good understanding of these.

- **Epistemic uncertainty:** (epistemic = “related to knowledge”)

  Uncertainties on the network weights, i.e. uncertainties that are introduced with the network – and the fact that it is trained only with finite data.

- Epistemic uncertainties are systematic uncertainties in our statistical analysis

- Does one of the two systematically dominate? In most of LHC applications, it seems like so far aleatoric uncertainties dominate, but with more and more involved neural networks this might change in the future.

- How can we estimate aleatoric uncertainties? Conceptually simple! At the LHC we usually know the errors on the input variables. So we sample from our statistical model, we run the input data through the network, we get a statistics of our predictor. Done!

Uncertainties: ML jargon / physics jargon

- **Aleatoric uncertainty:** (“alea” in latin is “dice”)

  All uncertainties related to the input data, e.g. sensor noise. In particle physics we often have a good understanding of these.

- **Epistemic uncertainty:** (epistemic = “related to knowledge”)

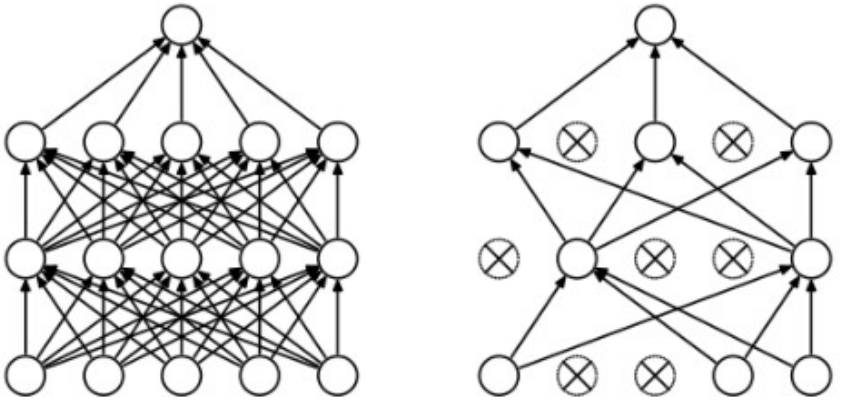
  Uncertainties on the network weights, i.e. uncertainties that are introduced with the network – and the fact that it is trained only with finite data.

- How can we estimate epistemic uncertainties?

Recap: Dropout

Dropout

- Idea: randomly “drop out” units during training
- Different units for each sample and in each iteration
- No unit can rely on the presence of other units for their work.
- Typical dropout rates: 0.5 for hidden and 0.2 for input units
- Implementation note: need to scale weights (or activations) after training



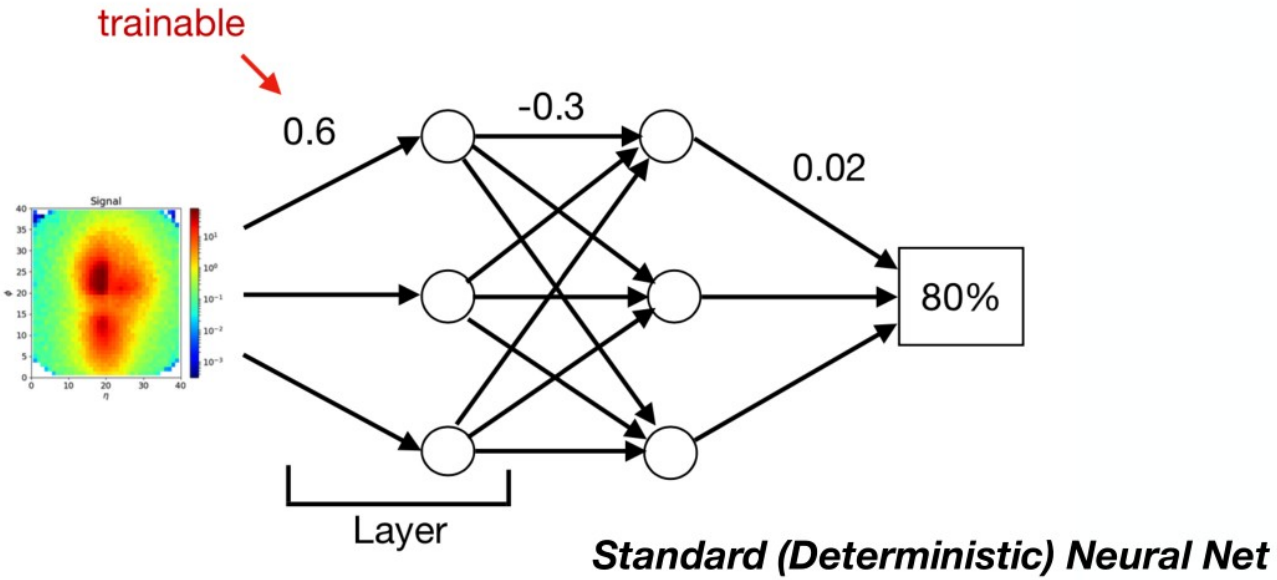
(a) Standard Neural Net (b) After applying dropout.

Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 2014

Monte Carlo Dropout

"Normal" Dropout is applied only during training, not during test time!

At test time, the prediction is deterministic: the same input will always result in the exact same prediction.



Standard (Deterministic) Neural Net

Monte Carlo Dropout

So what if you use dropout also at test time?

In this case you get a stochastic predictor. Running several times gives you an *ensemble* of predictions. It can be shown that this ensemble converges asymptotically against the posterior probability of the predictor.

"Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning"

https://arxiv.org/abs/1506.02142

Recap: Bayesian Statistics

Statistics is a lot about modelling random processes. Think e.g. a measurement of a length “X”. In statistics we typically model the measurement as the *realization of a random variable*, e.g. drawn from a Gaussian distribution with the “theory” parameters μ and σ^2, corresponding to the first two statistical moments:

$$\mathrm{X} \sim \mathrm{No}(\mu, \sigma^2)$$

The probability to measure a certain value “x” is then given by the likelihood

$$p(x) = \mathrm{No}(x; \mu, \sigma^2) = P(x|\mu, \sigma^2)$$

Note here, that it is the probability of observable “x”, given “theory parameters” μ and σ^2. The probability here can be rigorously interpreted as a (relative) frequency of observing “x” (admittedly, for continuous numbers we need to integrate, but still).

Recap: Bayesian Statistics

So frequentist statistics always connects the term “probability” with a (relative) frequency of observing something – hence the name.

The only conditional probabilities that are well defined for a frequentist are of the form:

$$p = \mathrm{p}(\mathrm{data}|\mathrm{theory})$$

Recap: Bayesian Statistics

So frequentist statistics always connects the term “probability” with a (relative) frequency of observing something – hence the name.

The only conditional probabilities that are well defined for a frequentist are of the form:

$$p = p(\text{data}|\text{theory})$$

However, we can employ the law of inverse probability:

$$p(A|B)p(B) = p(B|A)p(A)$$

and spectral decomposition (here for the discrete case):

$$p(B) = \sum_i p(B_i), p(B_i \cap B_j) = 0, \forall i \neq j$$

Recap: Bayesian Statistics

to arrive at Bayes theorem:

$$p(B_i|A) = \frac{p(A|B_i)p(B_i)}{\sum_i p(A|B_i)p(B_i)}$$

or, if we identify “A” as data, and B_i as “theory”:

$$p(\mathrm{theory}|\mathrm{data}) = \frac{p(\mathrm{data}|\mathrm{theory})p(\mathrm{theory})}{\int \mathrm{d}(\mathrm{theory})p(\mathrm{data}|\mathrm{theory})p(\mathrm{theory})}$$

Bayesian Neural Networks

"Weight Uncertainty in Neural Networks", (https://arxiv.org/abs/1505.05424)

$$p(\text{theory}|\text{data}) = \frac{p(\text{data}|\text{theory})p(\text{theory})}{\int \text{d}(\text{theory})p(\text{data}|\text{theory})p(\text{theory})}$$

In Bayesian neural networks, we think of "theory" as the weights of the network, and "data" as the network trained with data:

Weights given model **Model given weights** **Prior weights**

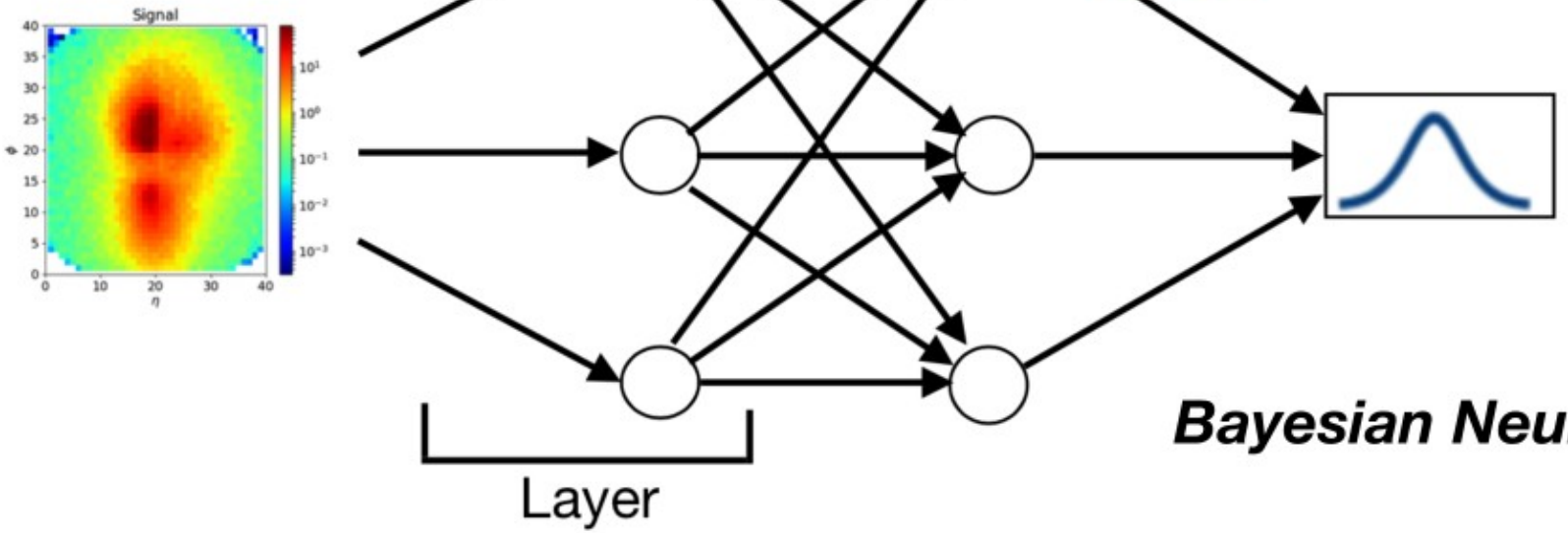
$$p(\omega|C) = \frac{p(C|\omega)\ p(\omega)}{p(C)}$$

Model evidendence

Replace weights (numbers) with probability distributions (Gaussians):

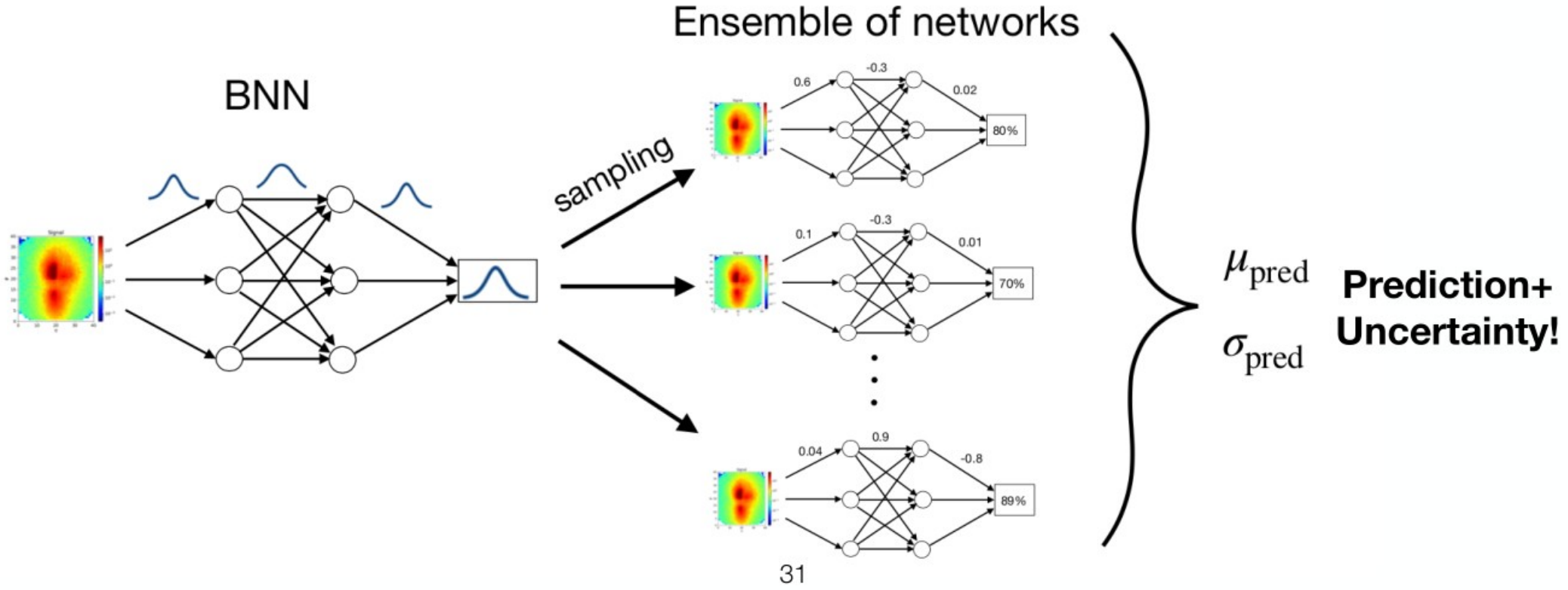
Bayes theorem prior

$$p(\omega \mid C) \propto p(C \mid \omega)\, p(\omega)$$
$$\approx \mathrm{N}(\omega \mid \mu, \sigma)$$



Per weight probability distributions gives us a measure of uncertainty on the network output

So we replace the network with an entire ensemble of networks, we sample from this ensamble and we have a posterior distribution on the prediction!



https://arxiv.org/abs/1505.05424

And to summarize …. uncertainties:

- We physicists obsess over errors and their estimates
- For errors on the input variables we simply perform error propagation
- For errors due to the network itself, we “go Bayesian” and mentally replace the weights of the networks with Bayesian posteriors.
- The very simple Monte Carlo dropout method can be thought of as a simple approximative algorithm.

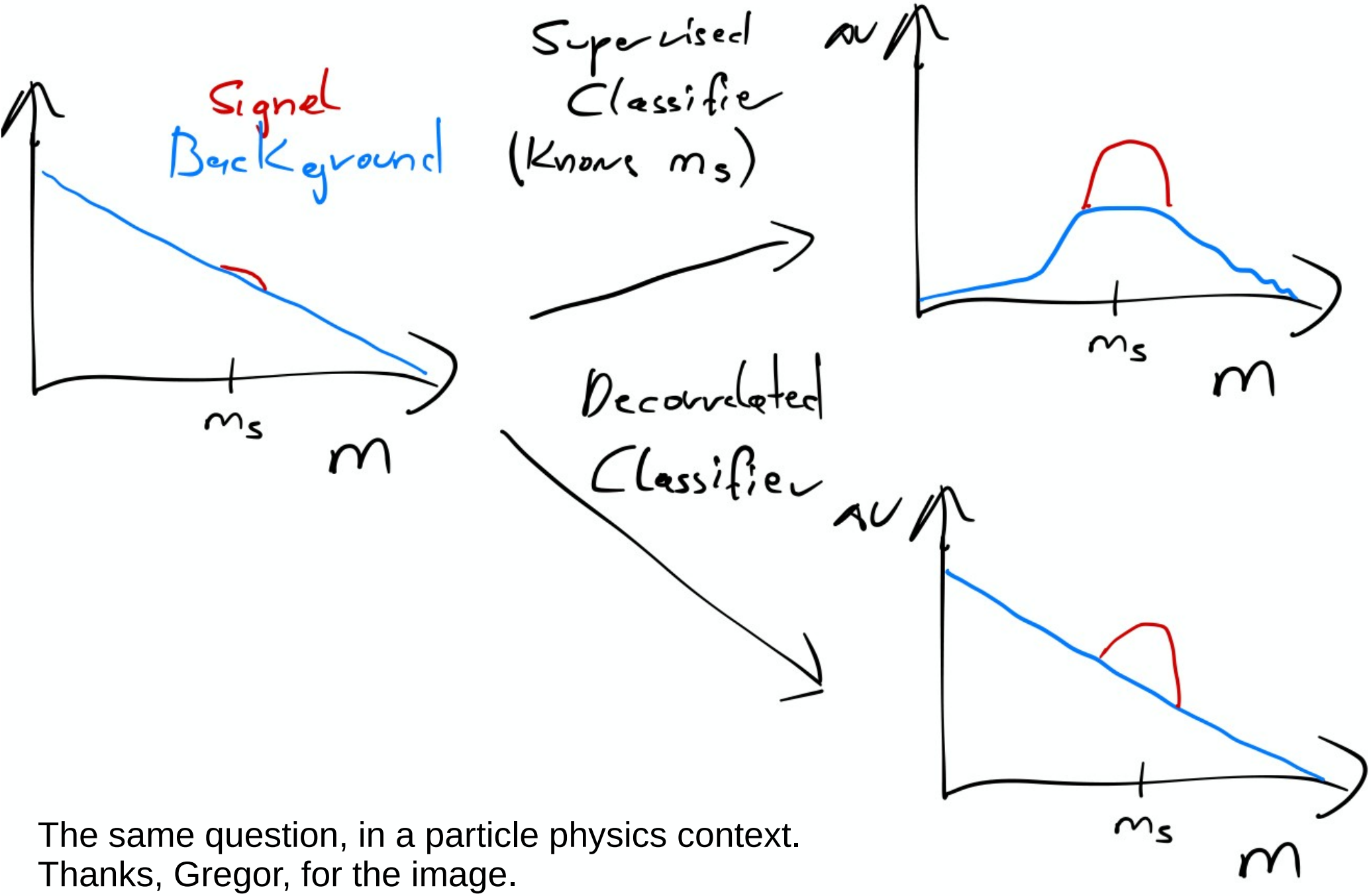
Decorrelating Variables

Decorrelating Variables

Task: Reduce or remove impact of certain variables on final analysis result.

Think e.g. a computational task at the Austrian AMS (Arbeitsmarktservice). Let's say, given all input data, the machine needs to propose persons for specific job openings. But let us further assume that politics dictates that e.g. the gender (or race or any other trait) of the person must not enter – neither explicitly nor implicitly – in the prediction.

How does one do this algorithmically?



The same question, in a particle physics context.
Thanks, Gregor, for the image.

Decorrelating Variables

Task: Reduce or remove impact of certain variables on final analysis result.

Think e.g. a computational task at the Austrian AMS (Arbeitsmarktservice). Let's say, given all input data, the machine needs to propose persons for specific job openings. But let us further assume that politics dictates that e.g. the gender (or race or any other trait) of the person must not enter – neither explicitly nor implicitly – in the prediction.

How does one do this algorithmically?

Well, removing the explicit dependency is easy: just remove the variable from the list of input features.

(And let's for the rest of the lecture call the variable that we want to decorrelate against, the "target feature".)

But how do we correct for the correlation of the input data with the prediction?

Simple solution: data planing

(What is the Machine Learning? arXiv:1709.10106)

If we have a statistically correct distribution of the target feature, we can simply perform “data planing”, i.e. reweight the input data so that the regressor’s prediction is flat, as a function of the target feature.

$$\left[w(\vec{x}_i)\right]^{-1} = C \left.\frac{\mathrm{d}\sigma(\vec{x}_i)}{\mathrm{d}m}\right|_{m=m_i}$$

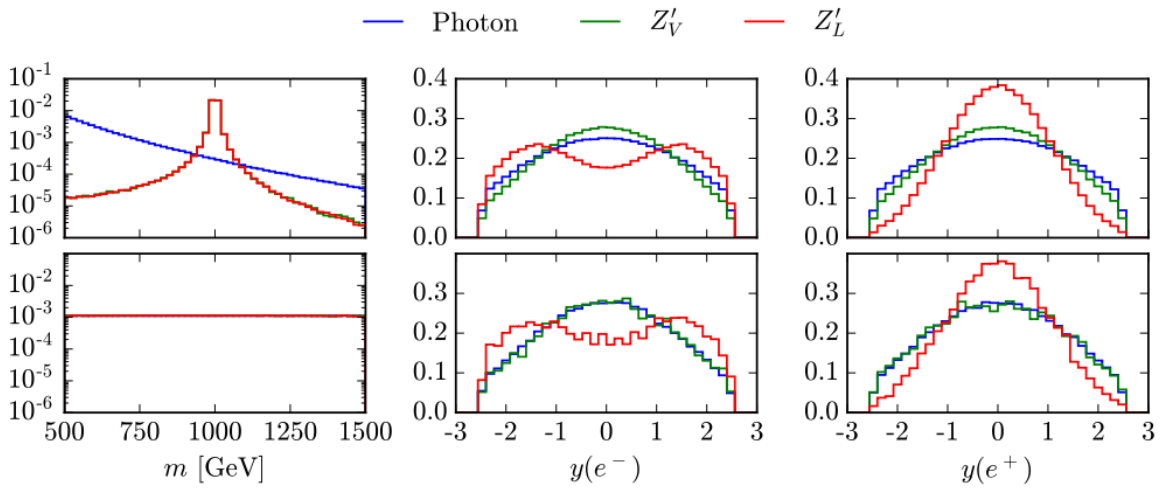
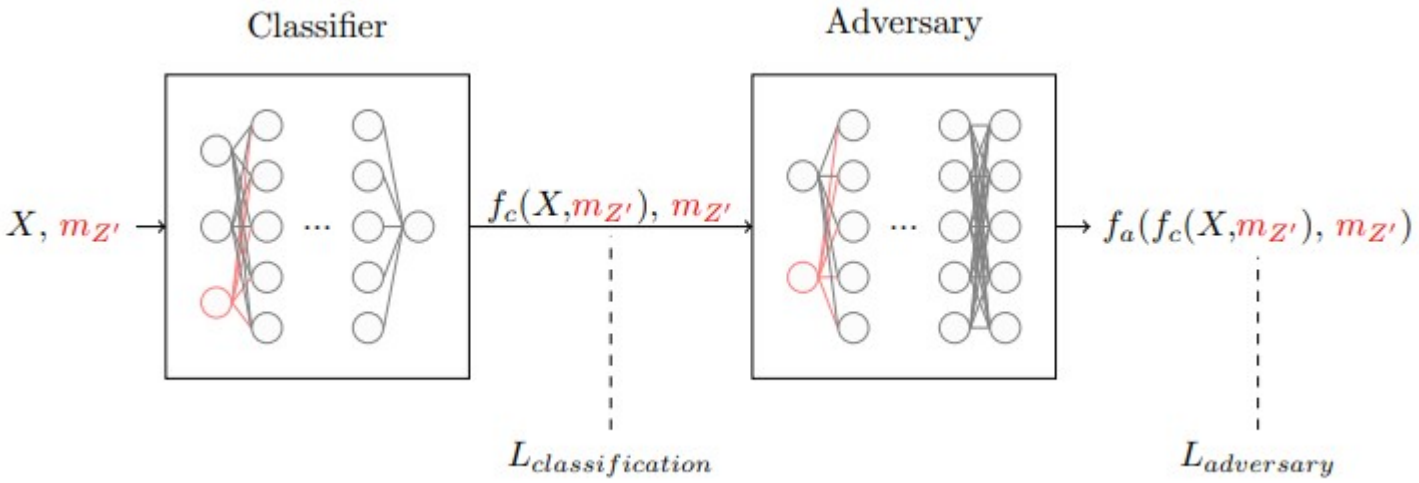


FIG. 2: Histograms of the constructed variables normalized to unity. The top [bottom] panels are before [after] planing the input events using the invariant mass m. The rapidity of the electron (positron) is specified by $y(e^-)$ ($y(e^+)$).

Decorrelating: Complex solution

"Decorrelated Jet Substructure Tagging using Adversarial Neural Networks"



$$L_{\text{tagger}} = L_{\text{classification}} - \lambda L_{\text{adversary}}$$

So the adversary tries to predict the mass from the output of the classifier (if that is possible then clearly the variables are not uncorrelated).
The "success" of this acts as a regularization term for the classifier.
[N.B. Adversarial training is notoriously difficult]

https://arxiv.org/abs/1703.03507

And to summarize …. decorrelating variables:

- Often, the problem arises that a regressor or a classifier should be decorrelated with a certain feature.
- (Think e.g. the political aspect here: the prediction for a job applicant to be a “good match” should perhaps not depend on e.g. the gender or the race of the person)
- The right algorithmic solution very much depends on the exact situation, e.g. an important question is: do we have a known functional dependence of the to-be-decorrelated variable from the input features, etc?
- One very generic solution is adversarial training, i.e. an adversary tries to learn the target feature from the output of the regressor. The regressor gets punished if the adversary succeeds.
- In many simple real-life cases we can simply perform “data-planing”, i.e. reweight our dataset so that the distribution of our target feature is flat.

And now …. exercises!

- I prepared only one fairly simple exercise on the MNIST dataset with ConvNets, implementing Monte Carlo Dropout.

- As mentioned before, you find all material at

  https://github.com/WolfgangWaltenberger/winterschool →

- We will use google's colaboratory for the tutorial
- On the github page given above of the repo you will find a link to google colab.