Particle Identification with Autoencoder Neural Networks

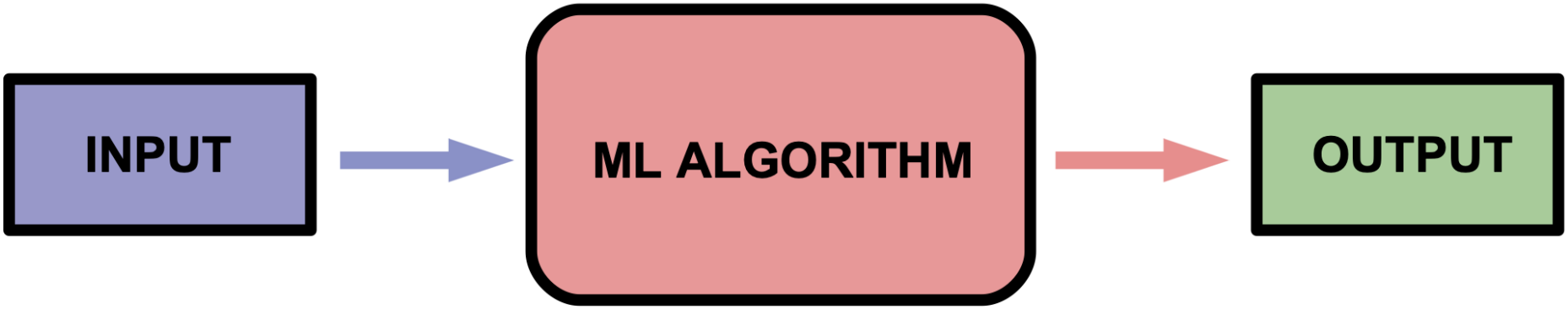
Daniel Lersch & Sean Dobbs

June 19, 2020

Overview

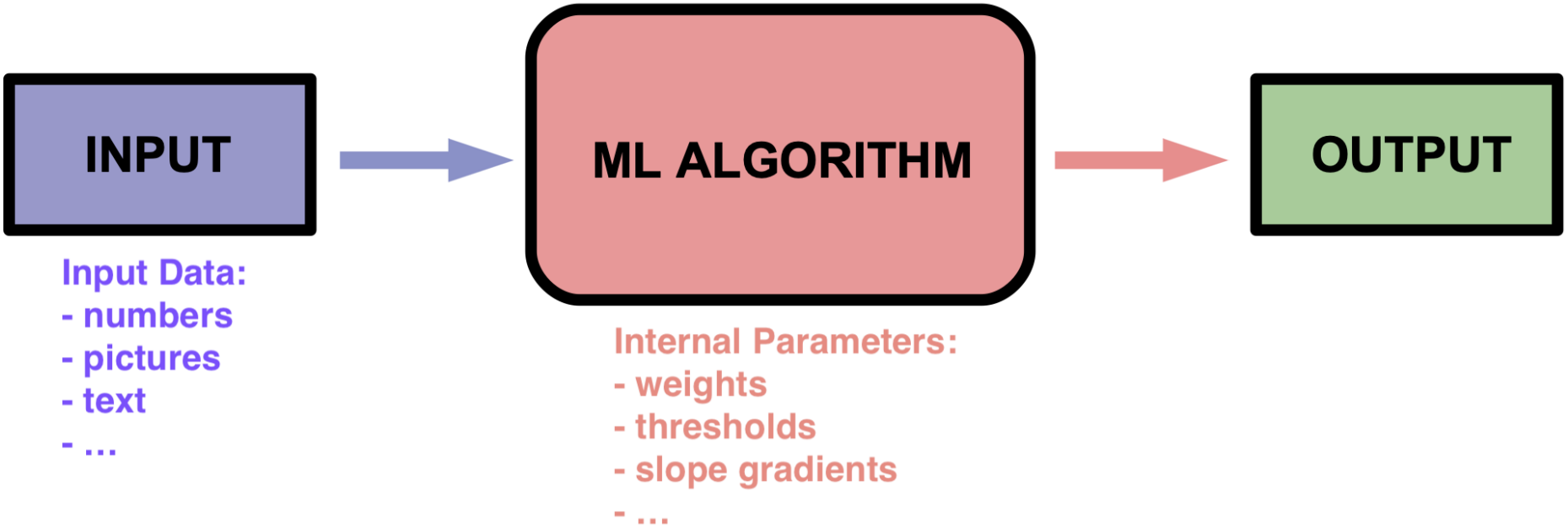
1. Very short introduction to machine learning
2. Definition and properties of (deep) neural networks
3. Application of autoencoders for particle identification at GlueX

Basic Components of Machine Learning

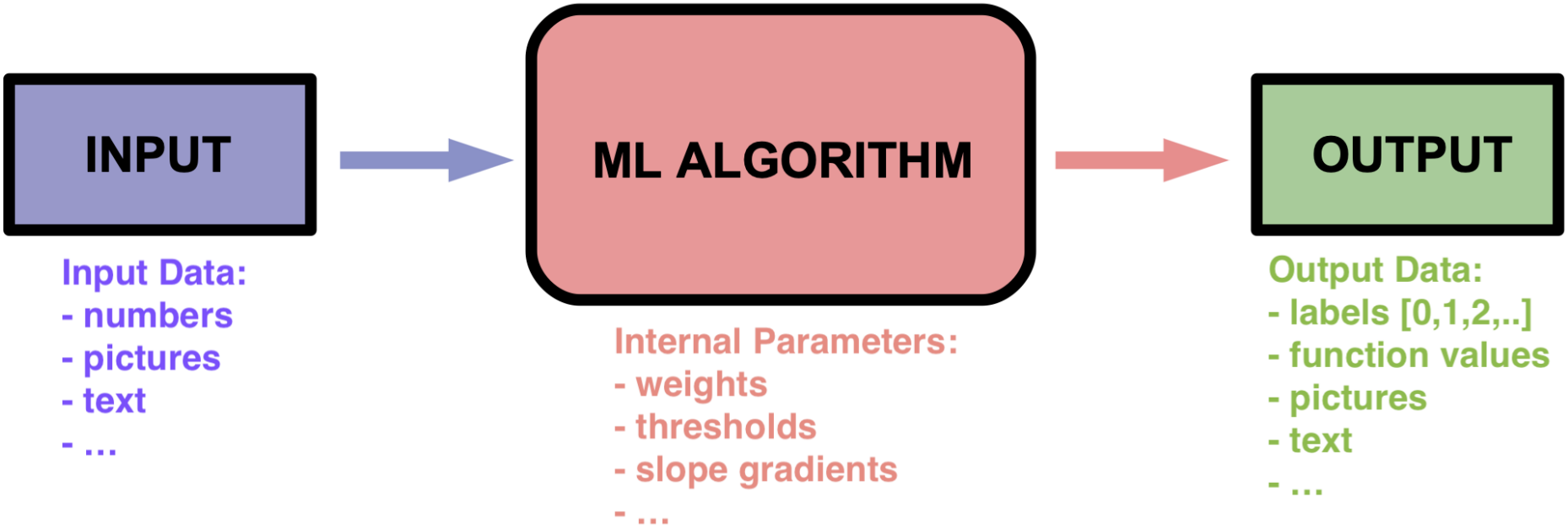


Basic Components of Machine Learning

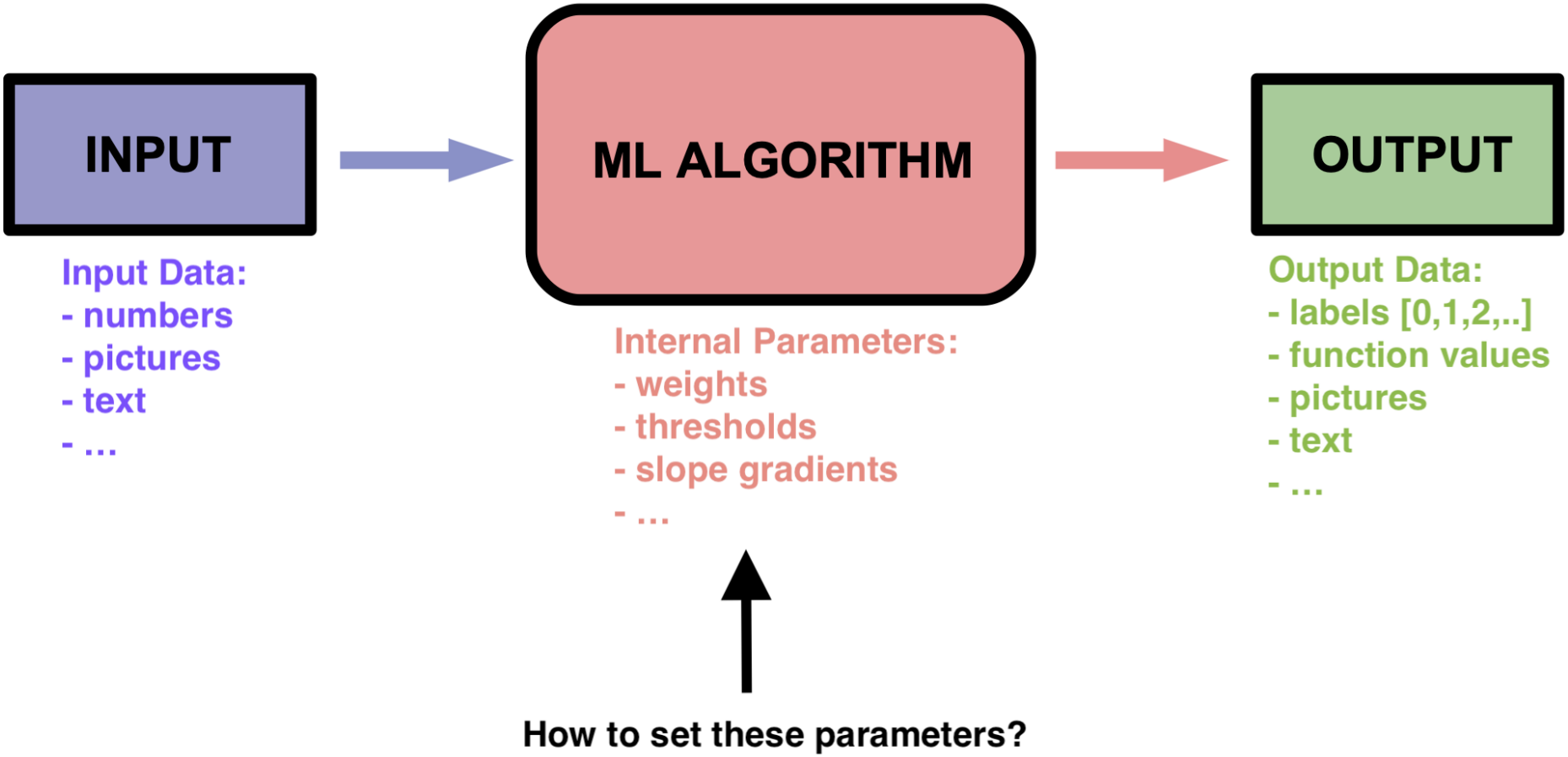
Basic Components of Machine Learning



Basic Components of Machine Learning

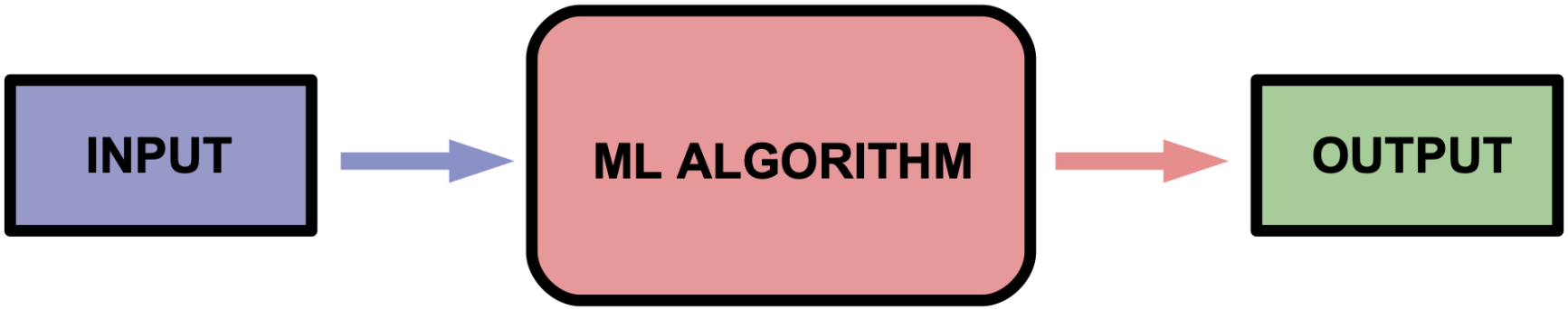


Basic Components of Machine Learning



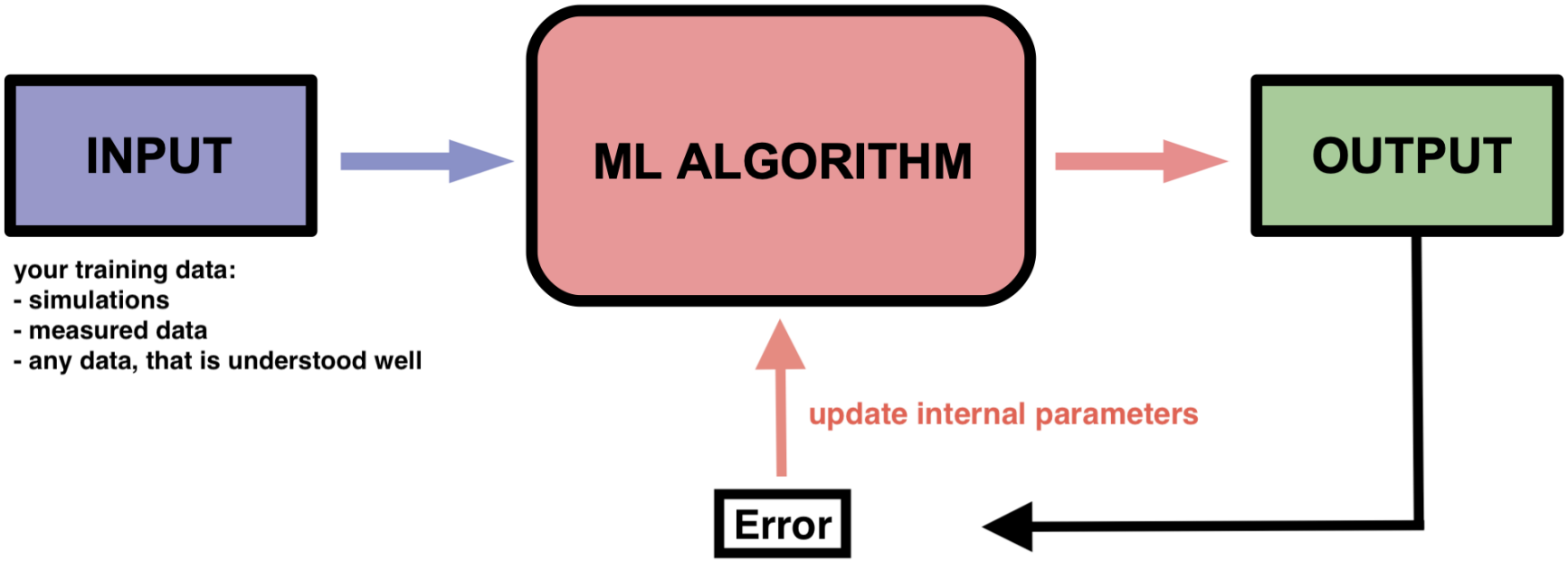
Training of Machine Learning Algorithms

- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training



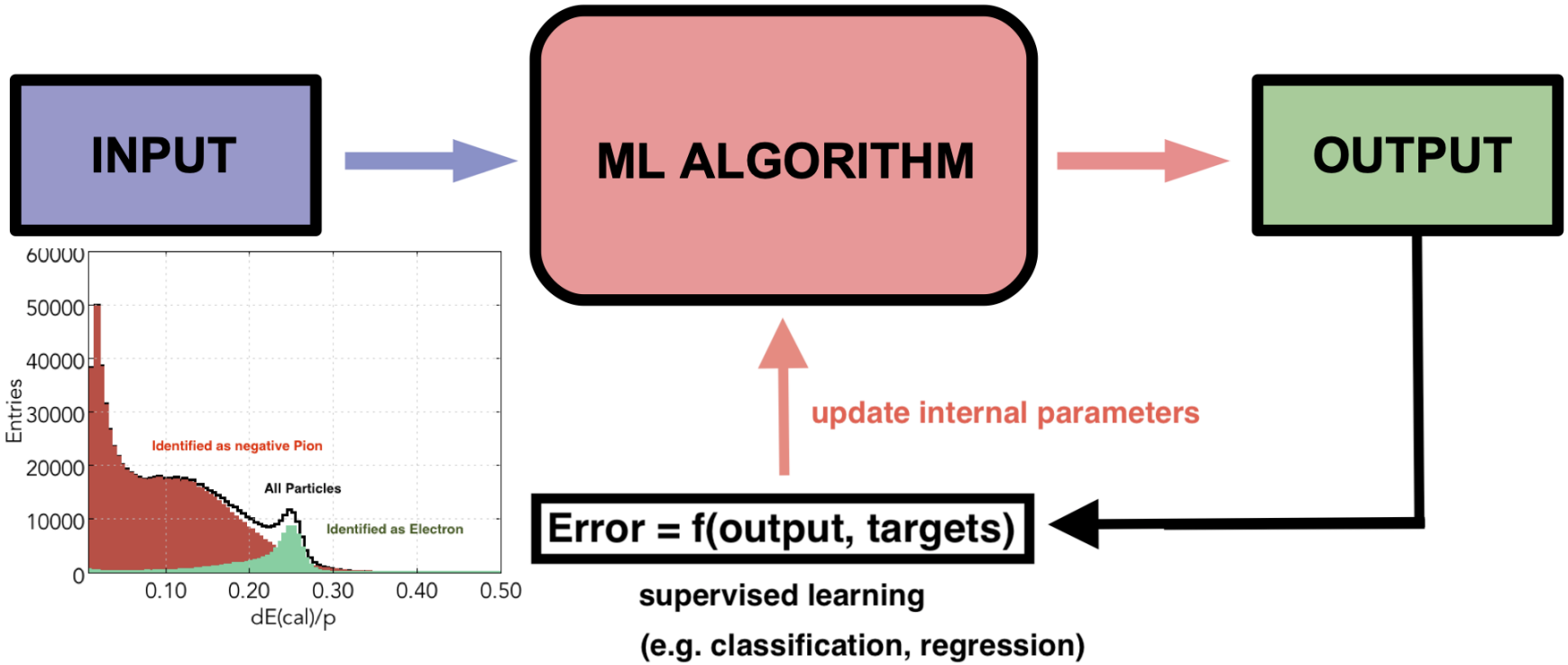
Training of Machine Learning Algorithms

- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training



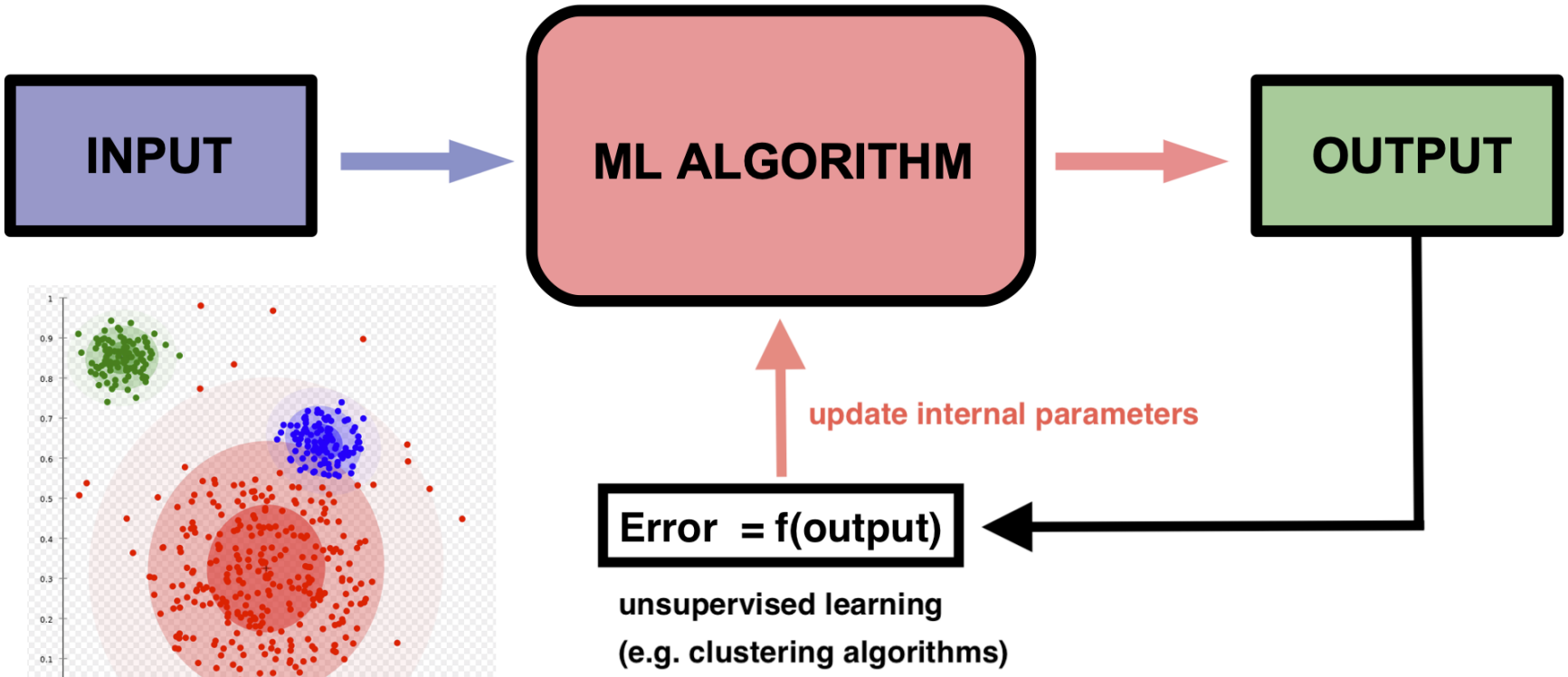
Training of Machine Learning Algorithms

- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training



Training of Machine Learning Algorithms

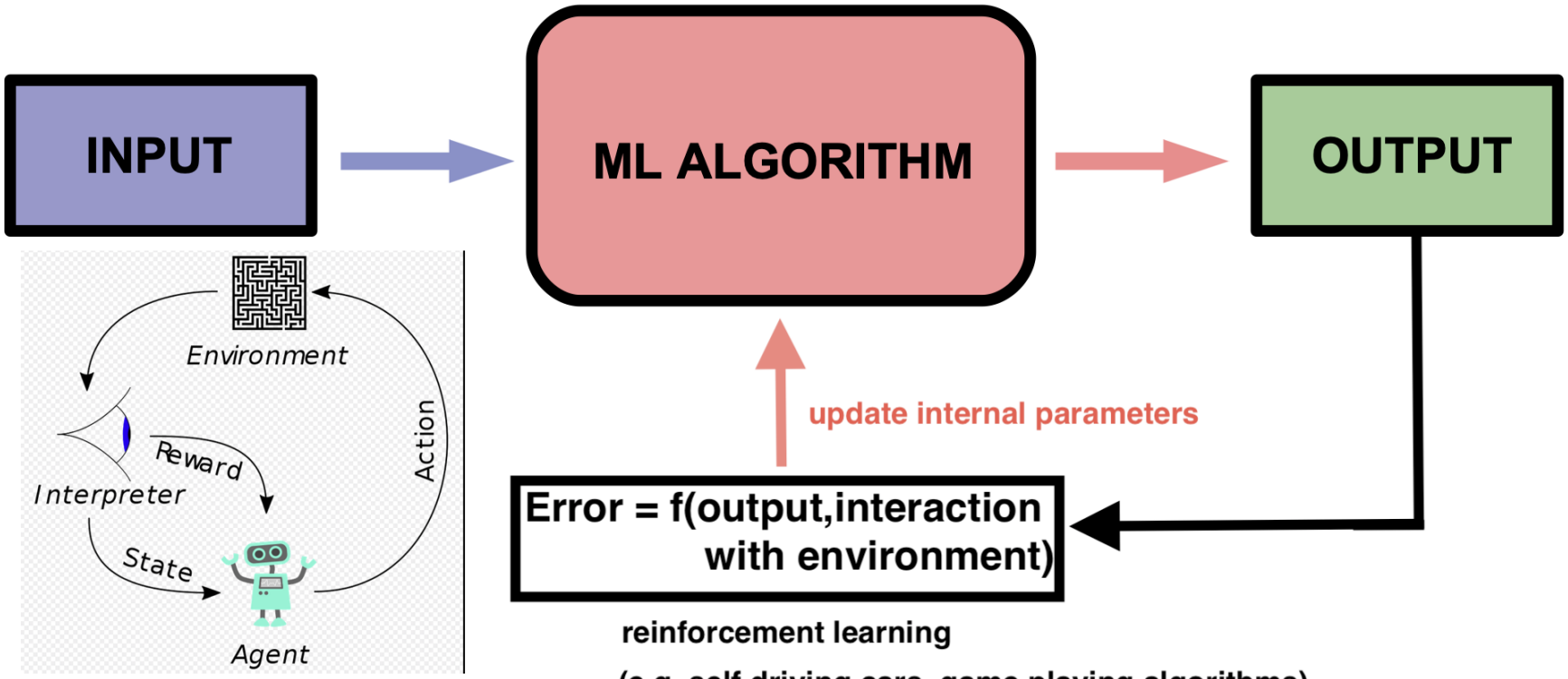
- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training



clustering algorithm
taken from: wikipedia
By Chire - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17087089

Training of Machine Learning Algorithms

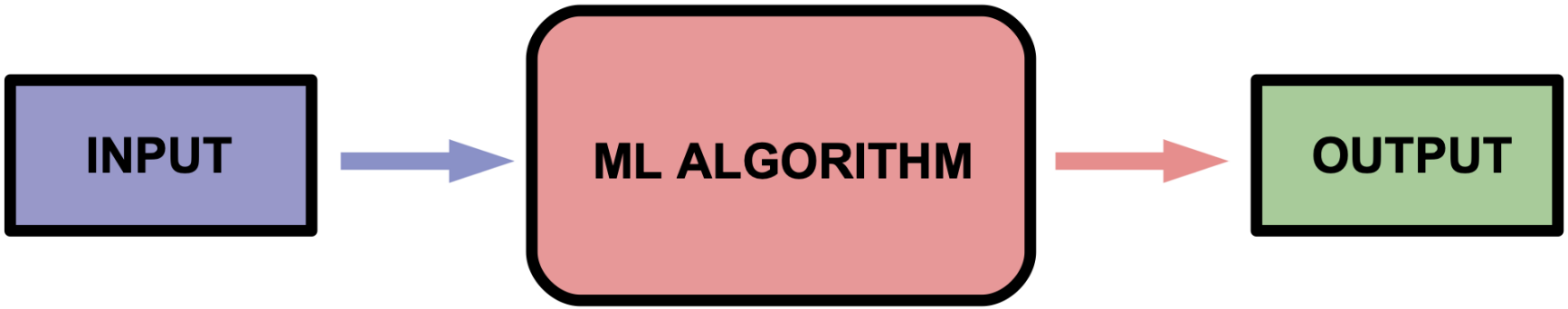
- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training



Taken from: wikipedia
By Megajuice - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741

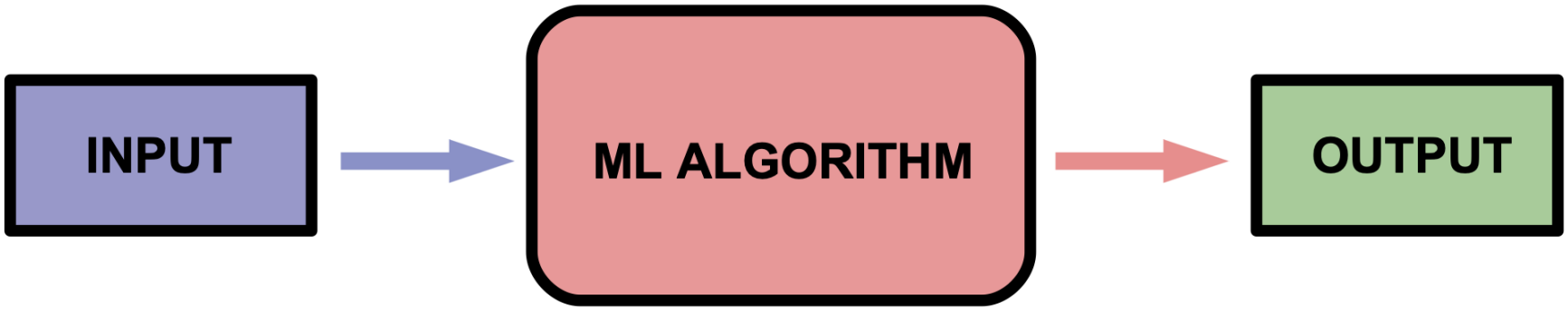
Training of Machine Learning Algorithms

- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training
- **NOTE:** The algorithms behavior / performance highly depends on the provided training data

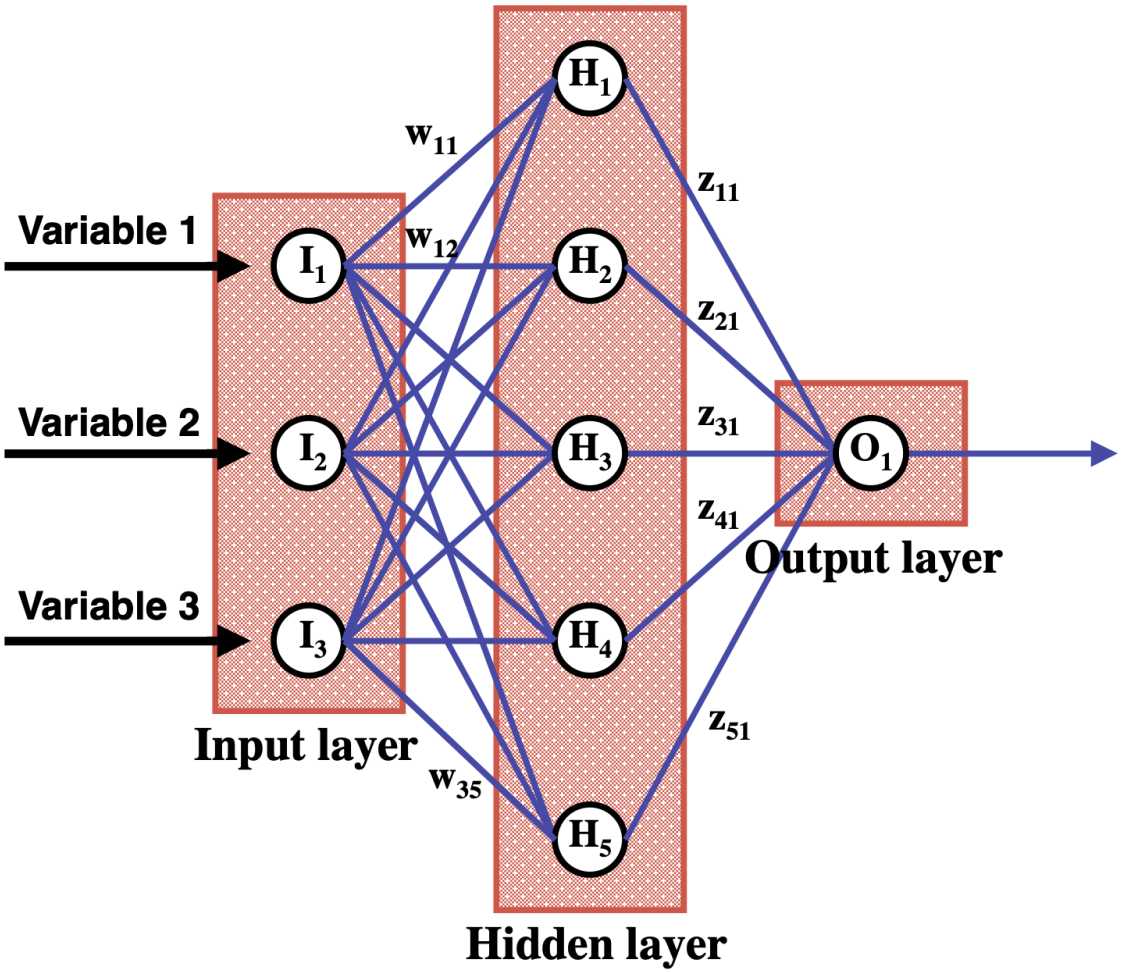


Training of Machine Learning Algorithms

- Any machine learning algorithms "learn" patterns / actions from a given data set by setting its internal parameters appropriately
- The parameter adjustment is done during training
- **NOTE:** The algorithms behavior / performance highly depends on the provided training data
- Your algorithm is useless if the training data significantly differs from the data you are trying to analyze (However, there is some room for variations...)

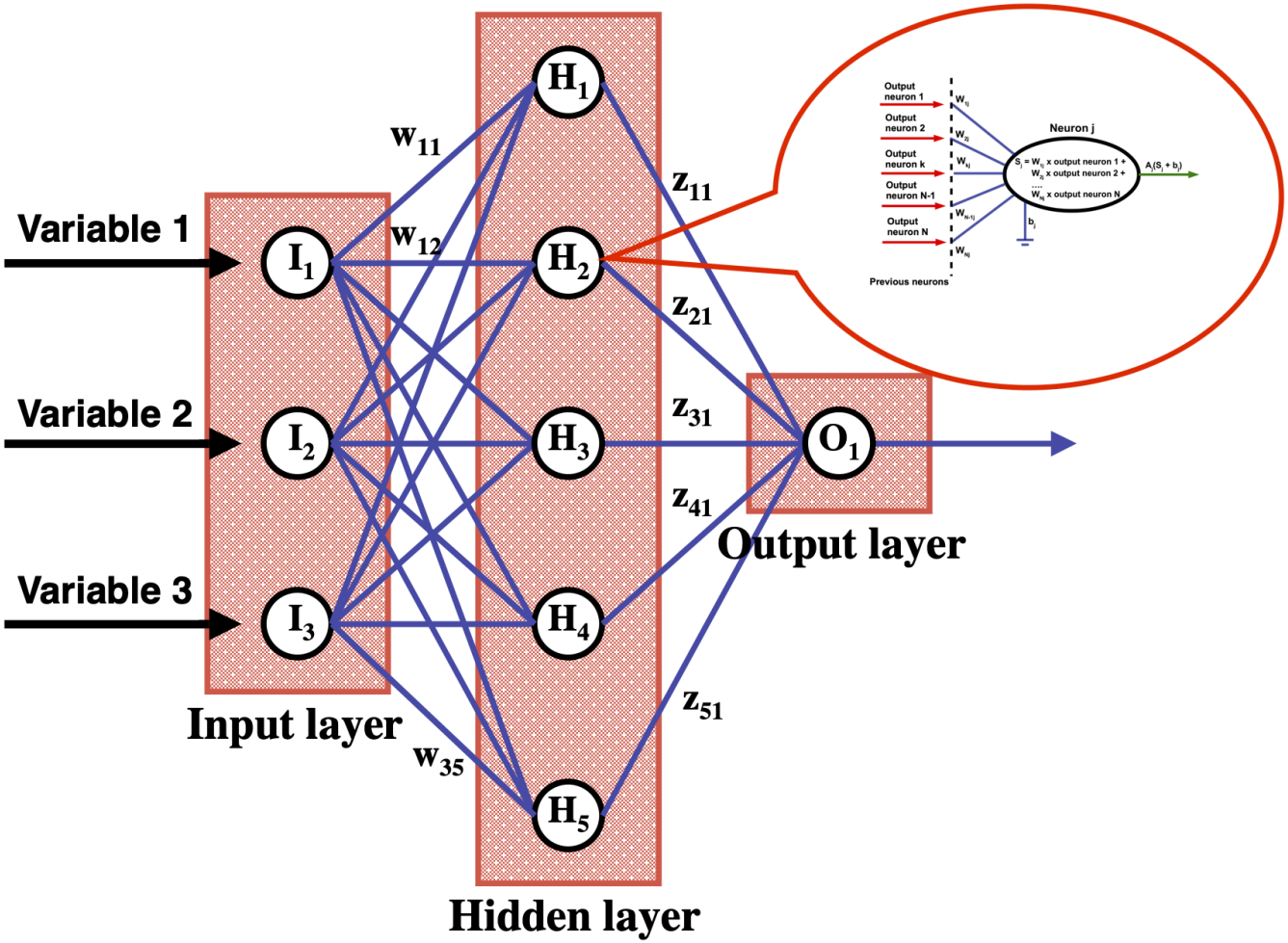


The Multilayer Perceptron



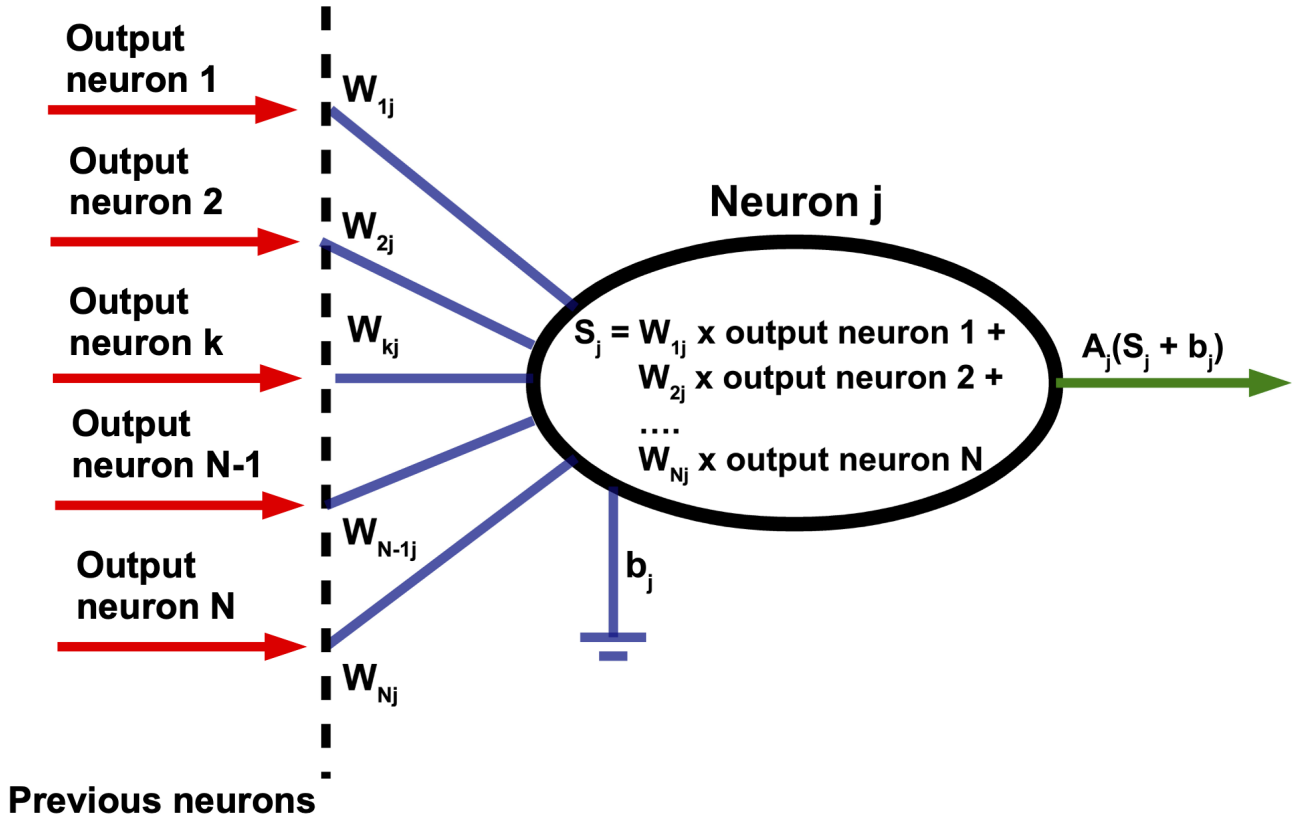
- Most popular example for machine learning algorithms
- Belongs to the class of feedforward neural networks
- **Architecture:** Hidden layers with a set of neurons

The Multilayer Perceptron



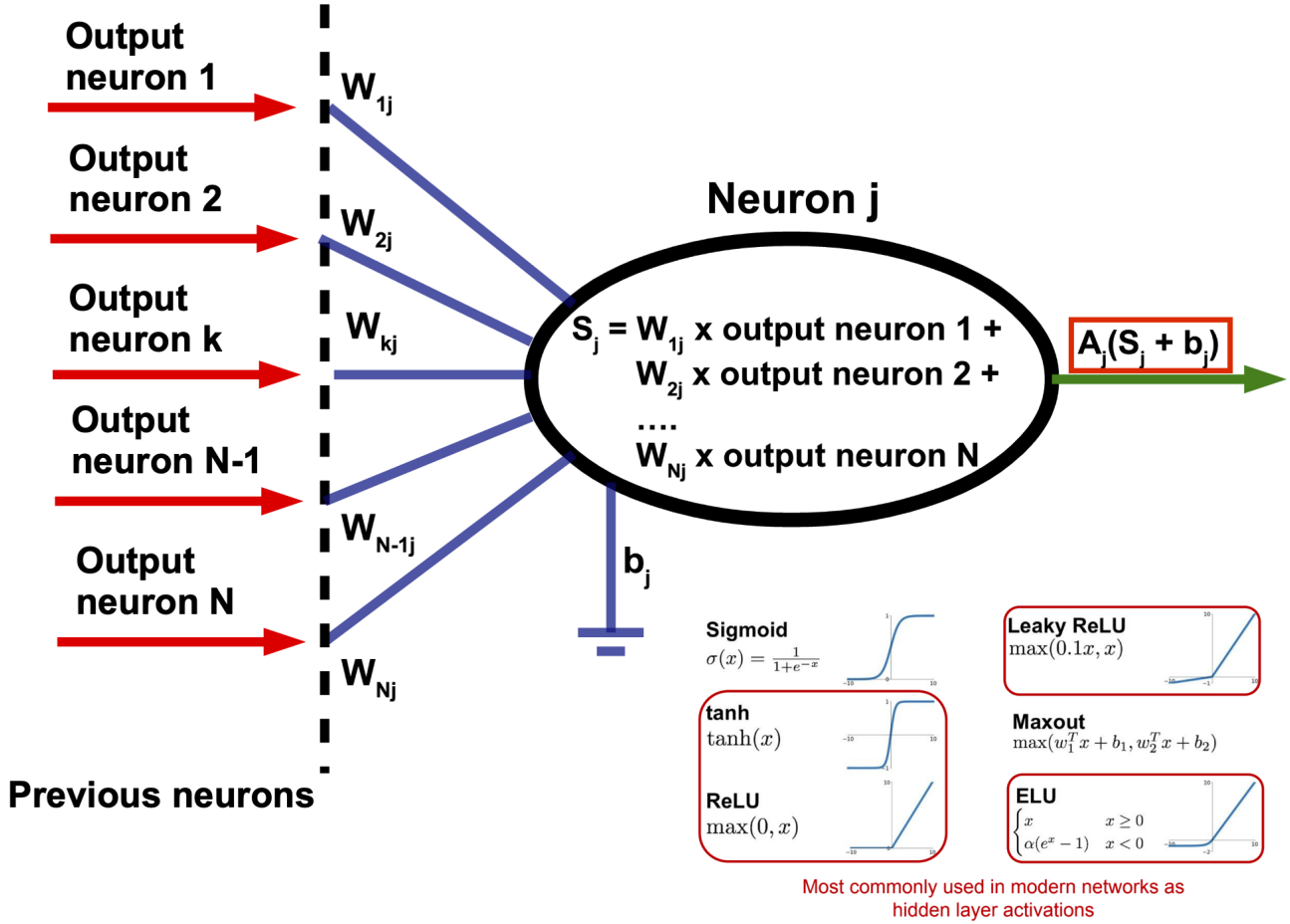
- Most popular example for machine learning algorithms
- Belongs to the class of feedforward neural networks
- **Architecture:** Hidden layers with a set of neurons

A single Neuron



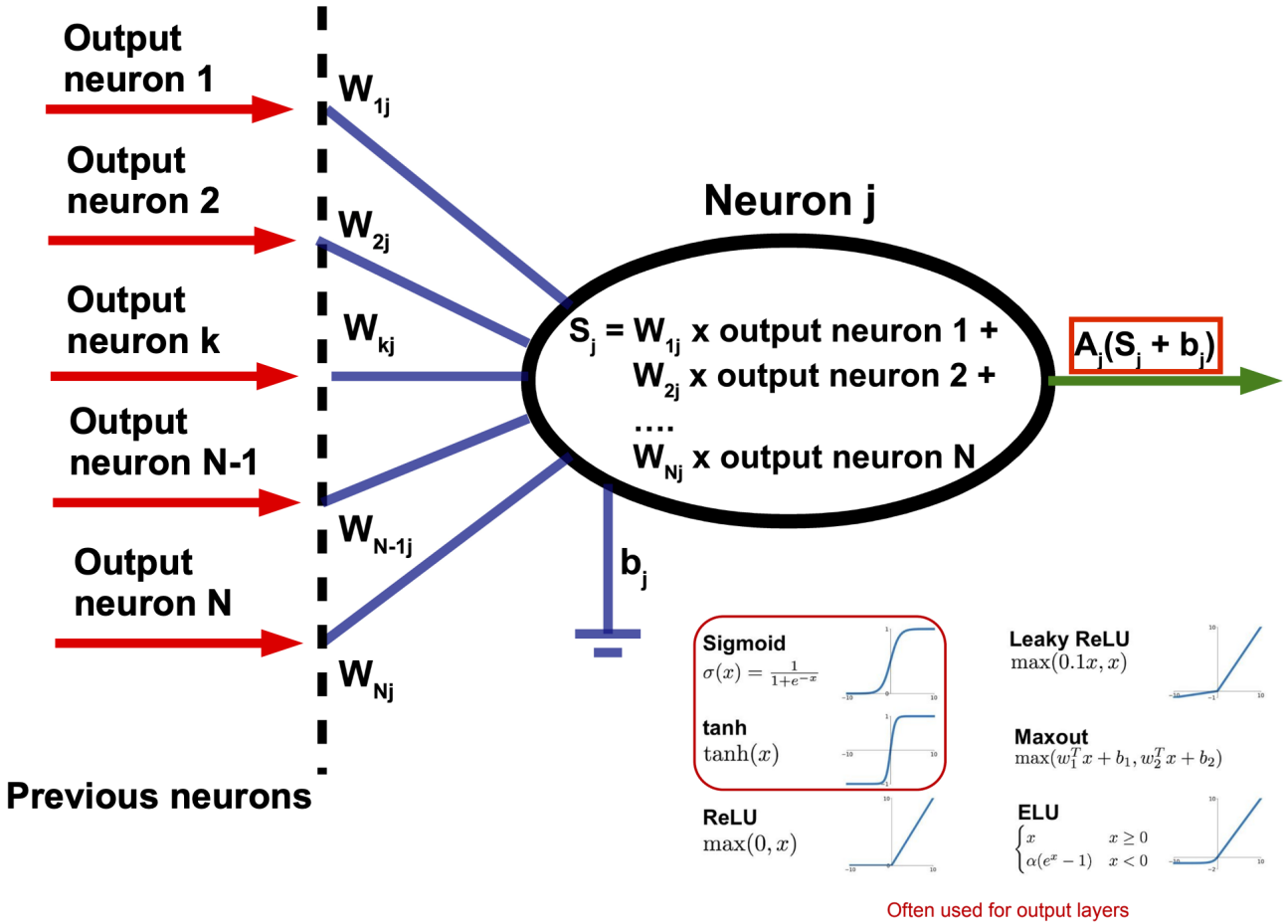
- **Basic ingredients:** Information from previous neurons, weights, bias and activation function

A single Neuron



- **Basic ingredients:** Information from previous neurons, weights, bias and activation function
- Activation function plots taken from Mustafa Mustafas lecture at the: **deep learning for science school 2019**

A single Neuron

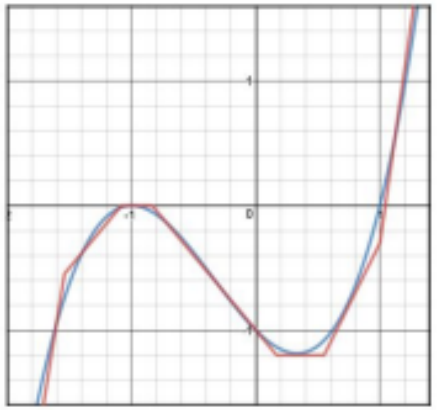


- **Basic ingredients:** Information from previous neurons, weights, bias and activation function
- Activation function plots taken from Mustafa Mustafas lecture at the: **deep learning for science school 2019**

The Universal Approximation Theorem for Neural Networks

"a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units" -- Hornik, 1991,
http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf

This, of course, does not imply that we have an optimization algorithm that can find such a function. The layer could also be too large to be practical.



$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

Fig. credit towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6

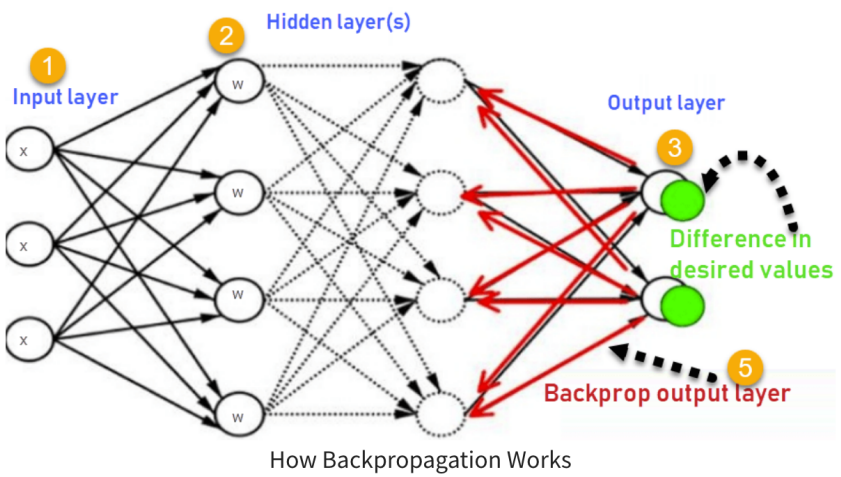
Screenshot taken from Mustafa Mustafas lecture at the: deep learning for science school 2019

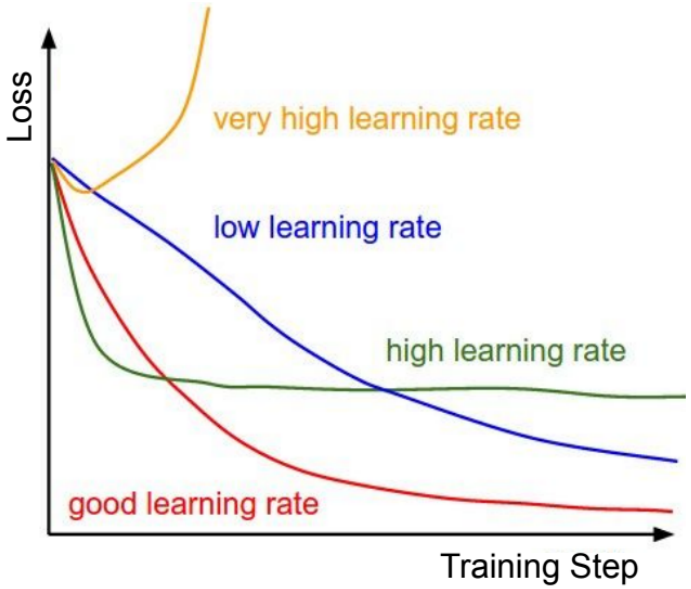
$\Rightarrow$ Similarly formulated in 1990 by the **Stone-Weierstrass-Theorem**
"[...] there are no nemesis functions that cannot be modeled by neural networks"

Training a (deep) Neural Network

Bottom left picture taken from here

Bottom right picture taken from Mustafa Mustafas talk at the: deep learning for science school 2019



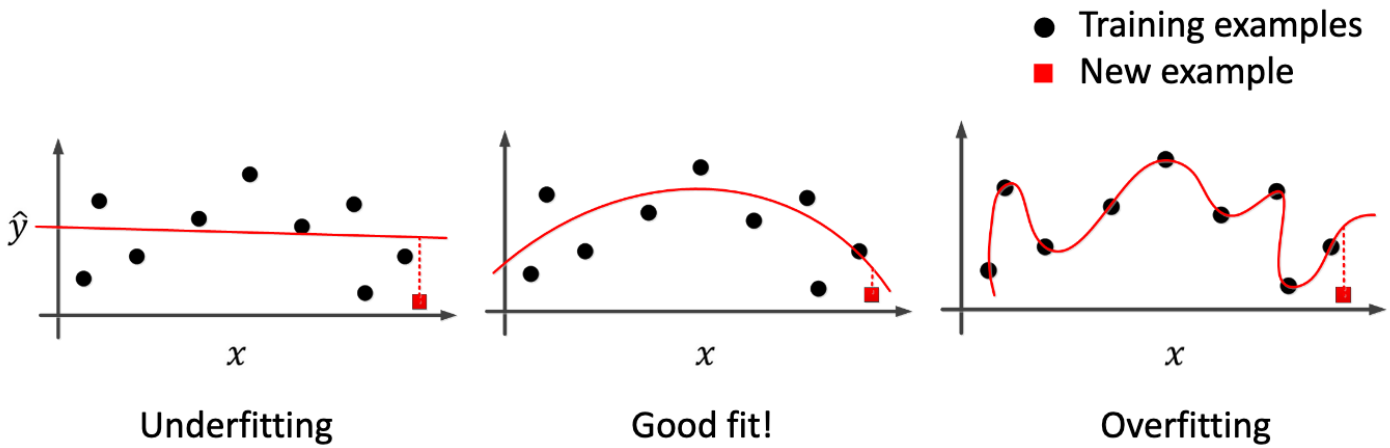


$$\textbf{Gradient Descent: } w_{k+1} = w_k - \eta \nabla L(w_k) = w_k - \frac{\eta}{m} \sum_{i=1}^{m} \nabla L(x_i, w_k) \tag{1}$$

- It is all about updating the weights w_k
  L: Loss function $\propto$ prediction - truth
  η : Learning rate (Common choice: 1/N(Training epochs))
  m: Batch size
- Problem: $\nabla L = 0$ (vanishing gradient)
- Different variants of stochastic gradient descent (L-BFGS, Adam, SGD,...)

Overfitting

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)



Picture taken from Brenda Ngs introductory talk at the: deep learning for science school 2019

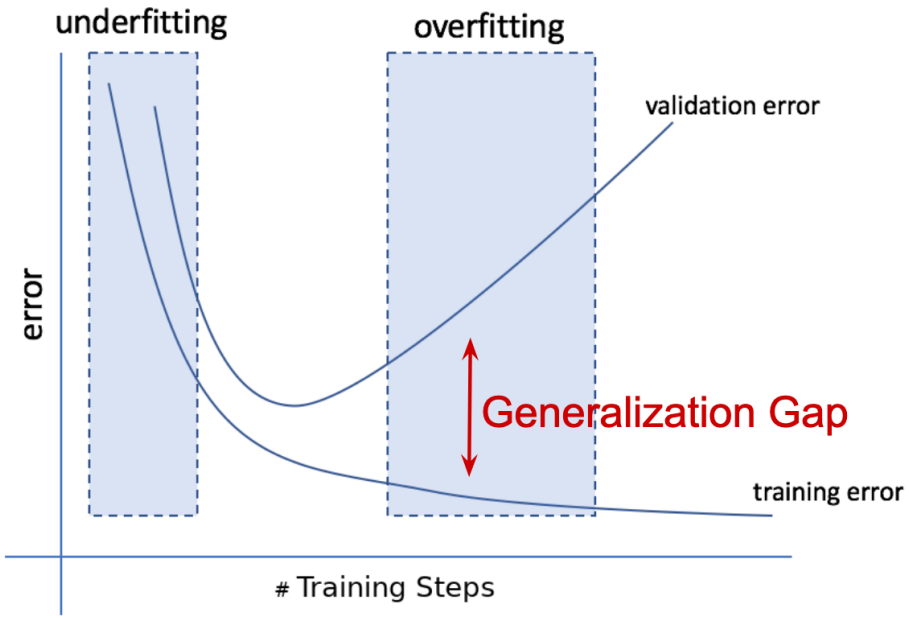
Overfitting

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)
- **Validation Data:** Part of training data that is NOT used to update internal parameters[1], but used to determine when training is complete

[1] This data is "unseen" by the algorithm during the training stage

Overfitting

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)
- **Validation Data:** Part of training data that is NOT used to update internal parameters[1], but used to determine when training is complete

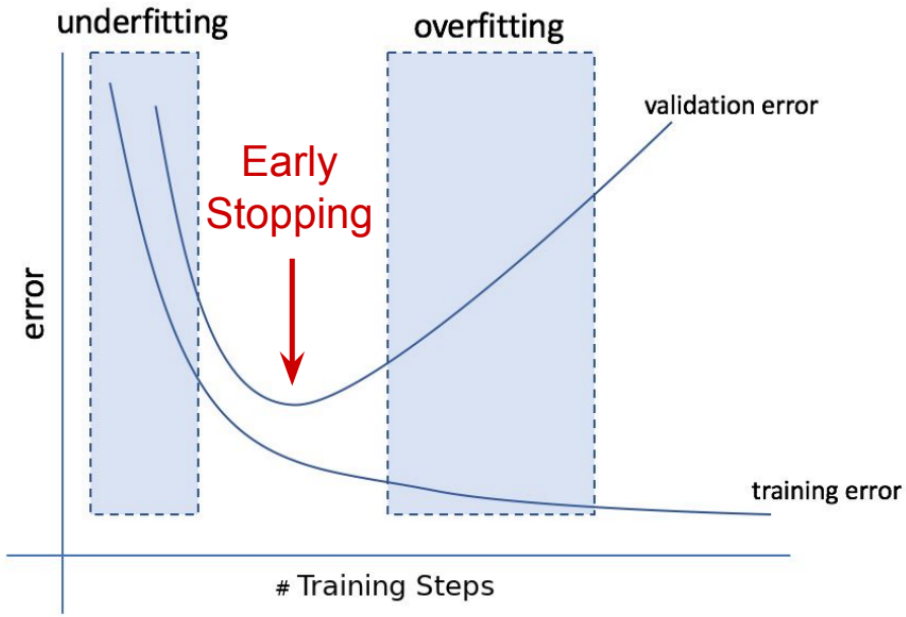


Picture taken from Mustafa Mustafas talk at the: deep learning for science school 2019

[1] This data is "unseen" by the algorithm during the training stage

Overfitting

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)
- **Validation Data:** Part of training data that is NOT used to update internal parameters[1], but used to determine when training is complete



Picture taken from Mustafa Mustafas talk at the: deep learning for science school 2019

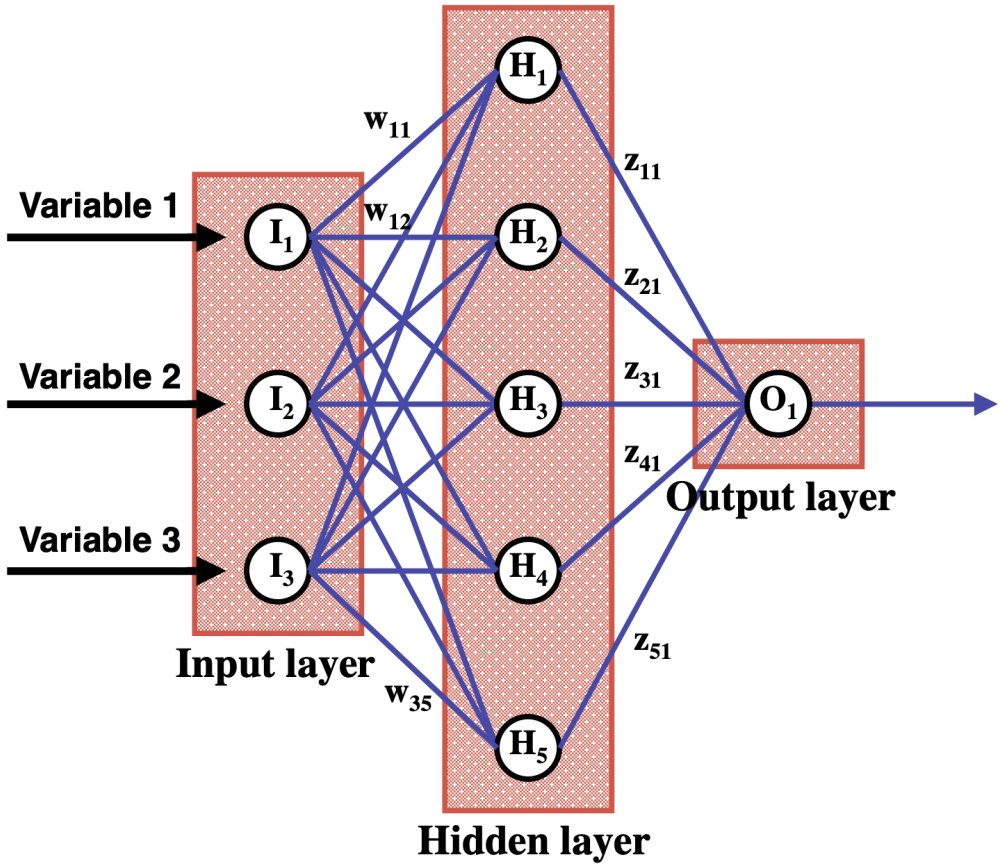
[1] This data is "unseen" by the algorithm during the training stage

From Machine to Deep Learning (in a very naive picture)

- "Classical" Machine learning

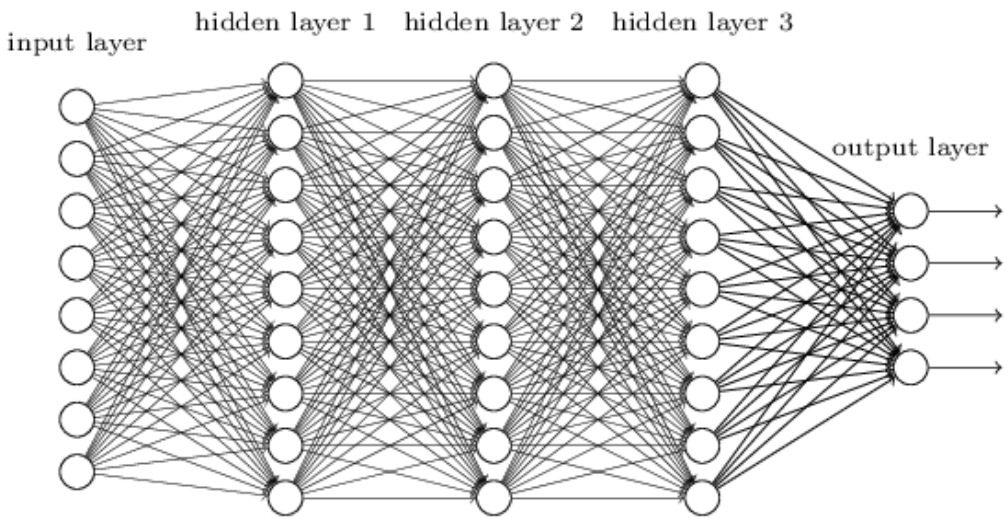
From Machine to Deep Learning (in a very naive picture)

- "Classical" Machine learning
  - Features (inputs for your algorithm) are obtained after some pre-processing (calibration, analysis cuts, variable selection,...) $\rightarrow$ Also known as feature-engineering
  - By pre-processing, you already encode information into the data
  - Moderate model size (e.g 1-2 hidden layers in a neural network)



From Machine to Deep Learning (in a very naive picture)

- "Classical" Machine learning
  - Features (inputs for your algorithm) are obtained after some pre-processing (calibration, analysis cuts, variable selection,...) $\rightarrow$ Also known as feature-engineering
  - By pre-processing, you already encode information into the data
  - Moderate model size (e.g 1-2 hidden layers in a neural network)
- Deep learning
  - Still machine learning, but uses neural networks only
  - Leave out (certain) pre-processing steps $\rightarrow$ Let the model do the work for you
  - The neural network becomes deep $\rightarrow$ Pre-processing is basically done in extra hidden layers

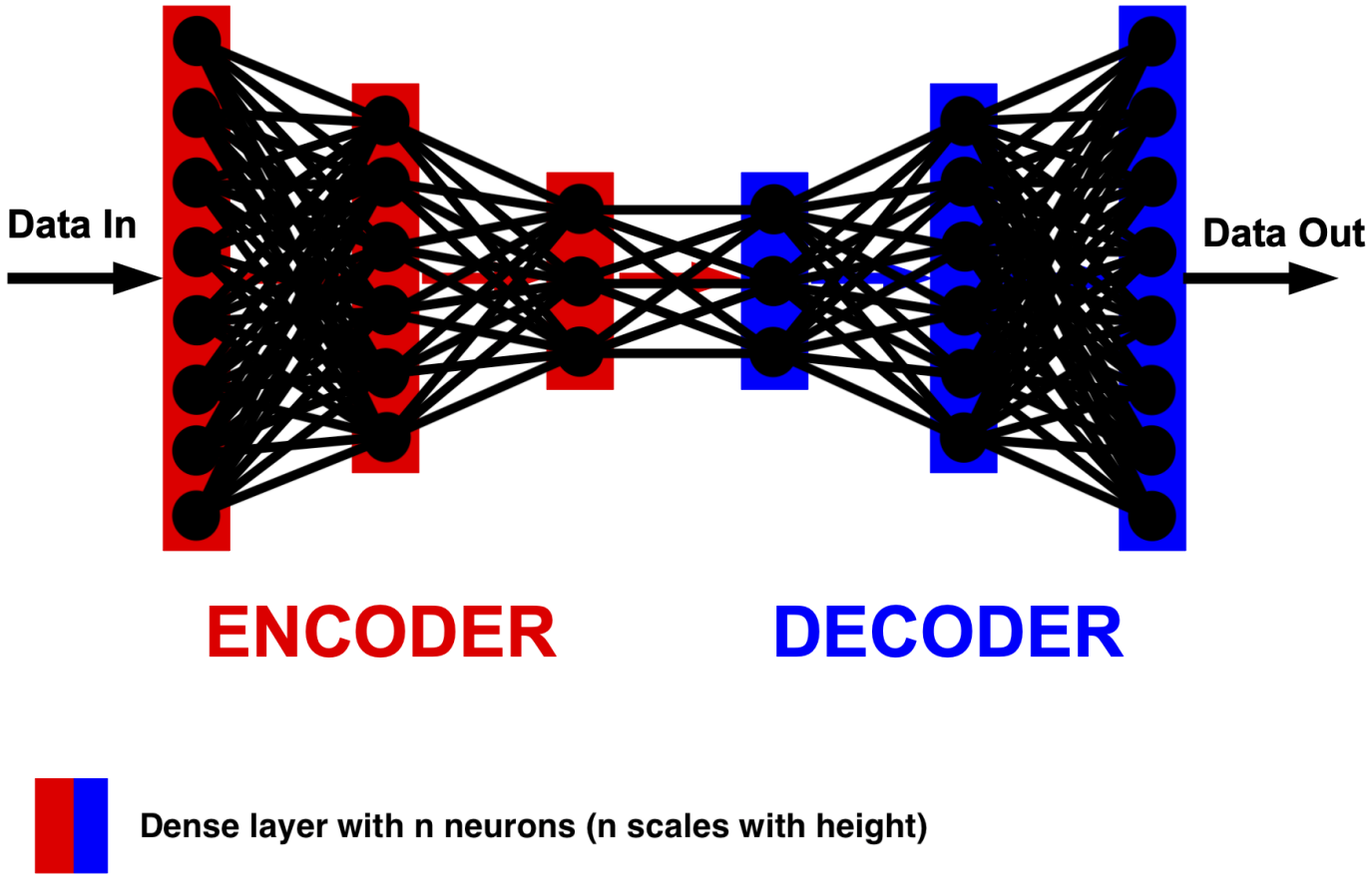


Picture taken from here

From Machine to Deep Learning (in a very naive picture)

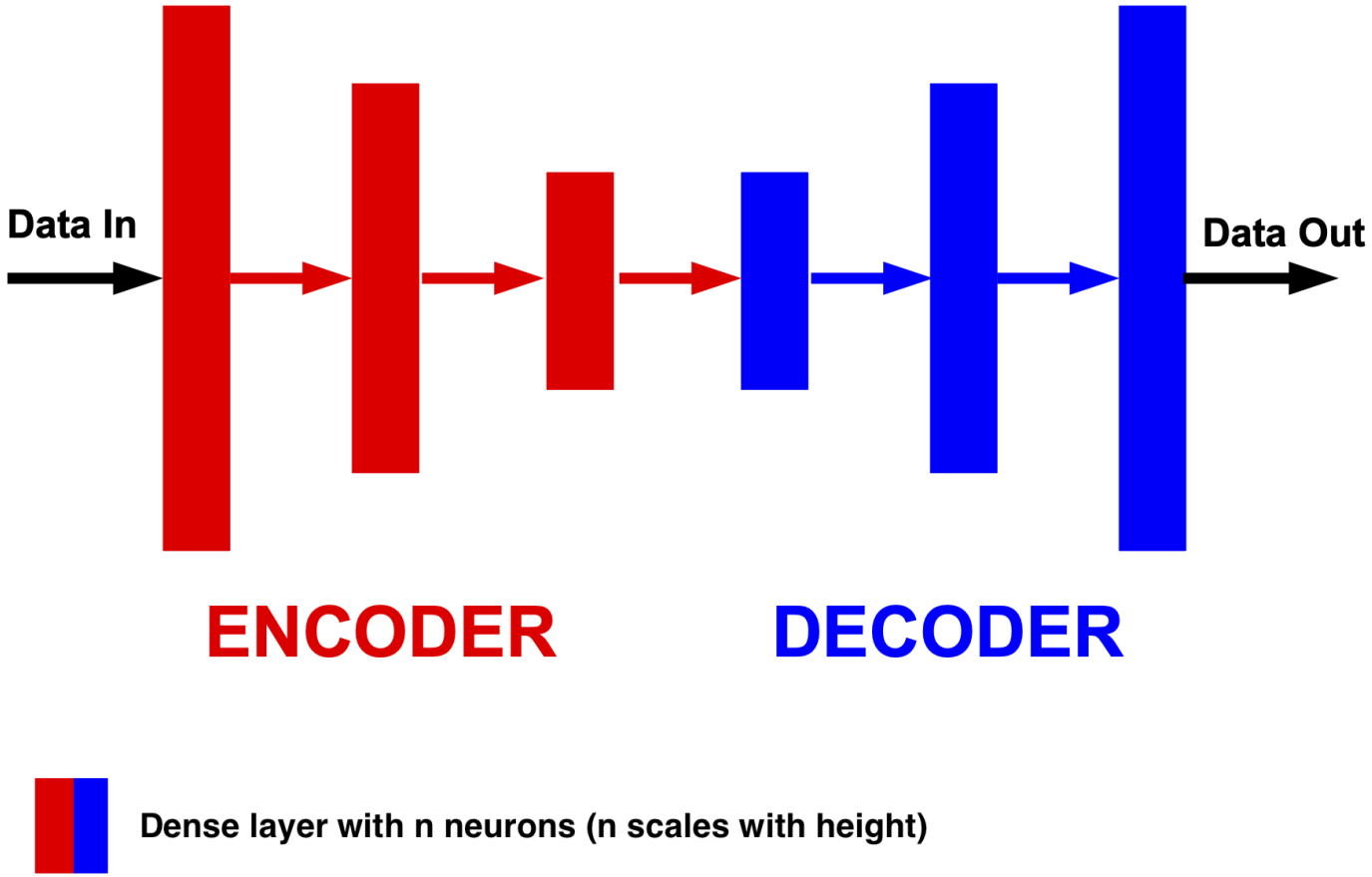
- "Classical" Machine learning
  - Features (inputs for your algorithm) are obtained after some pre-processing (calibration, analysis cuts, variable selection,...) $\rightarrow$ Also known as feature-engineering
  - By pre-processing, you already encode information into the data
  - Moderate model size (e.g 1-2 hidden layers in a neural network)
- Deep learning
  - Still machine learning, but uses neural networks only
  - Leave out (certain) pre-processing steps $\rightarrow$ Let the model do the work for you
  - The neural network becomes deep $\rightarrow$ Pre-processing is basically done in extra hidden layers
- Deep learning is not trivial, but fortunately there are many frameworks
  - Pytorch
  - Keras
  - Tensorflow (used for the work presented here)
  - ...

Autoencoders: Basics



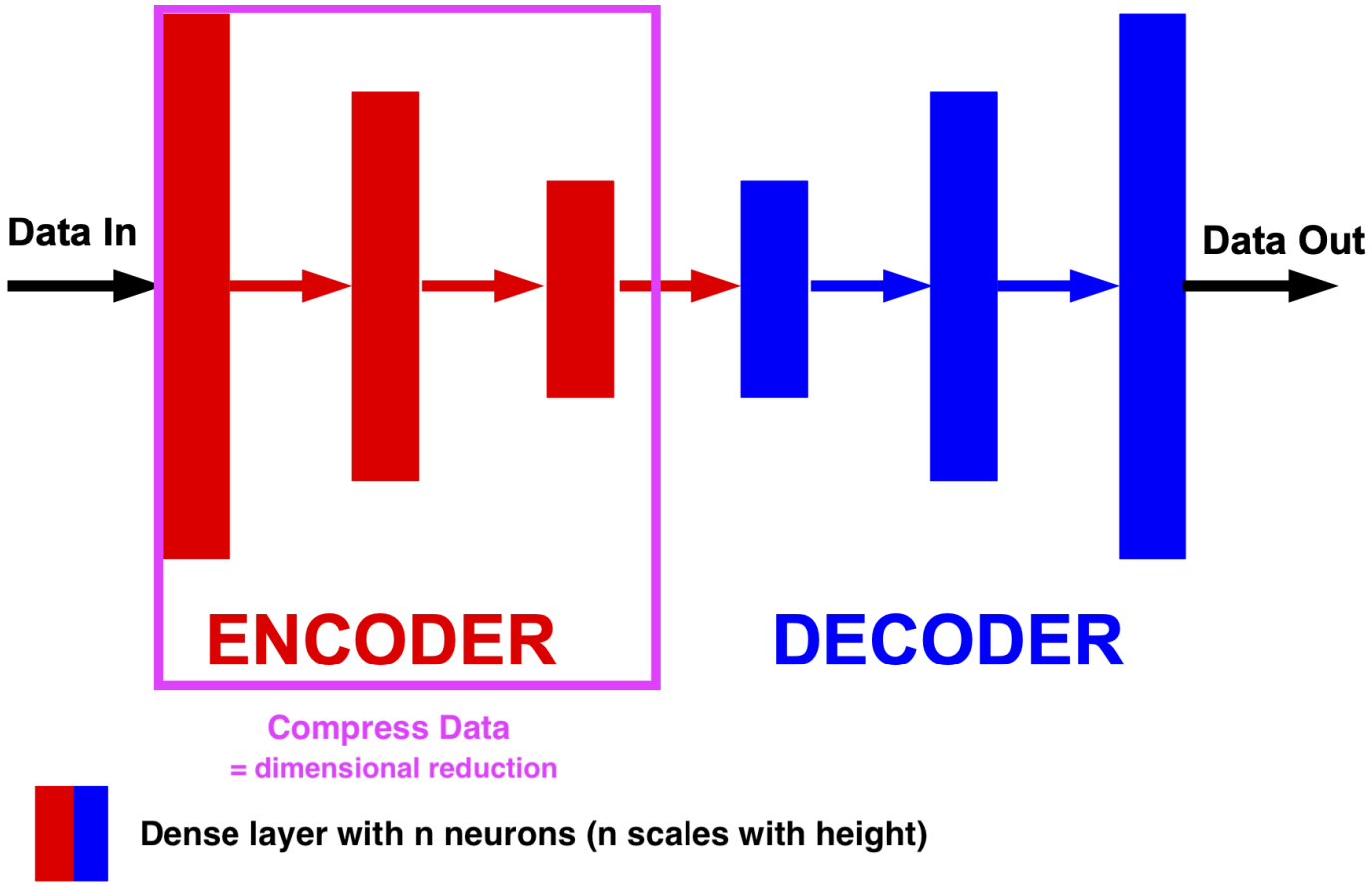
- Consists of two parts: encoder + decoder $\rightarrow$ Symmetric sandwich architecture
- Train autoencoder for: **Data In** $\approx$ **Data Out**
- **But:** Number of neurons needs to decrease in encoding part and increase in decoding part $\rightarrow$ Otherwise no effect $\leftrightarrow$ Identity function

Autoencoders: Basics



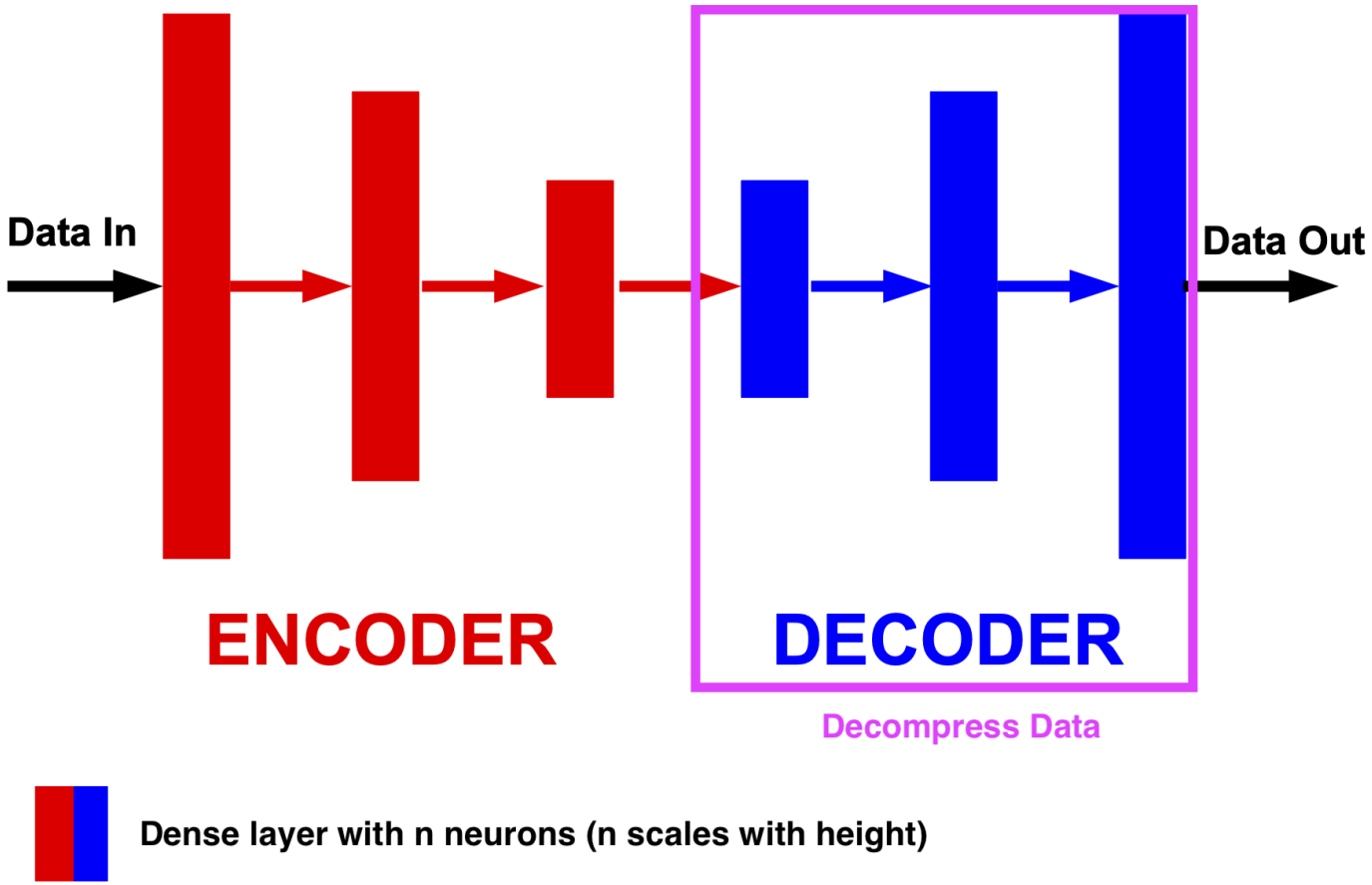
- Consists of two parts: encoder + decoder $\rightarrow$ Symmetric sandwich architecture
- Train autoencoder for: **Data In** $\approx$ **Data Out**
- **But:** Number of neurons needs to decrease in encoding part and increase in decoding part $\rightarrow$ Otherwise no effect $\leftrightarrow$ Identity function

Autoencoders: Basics



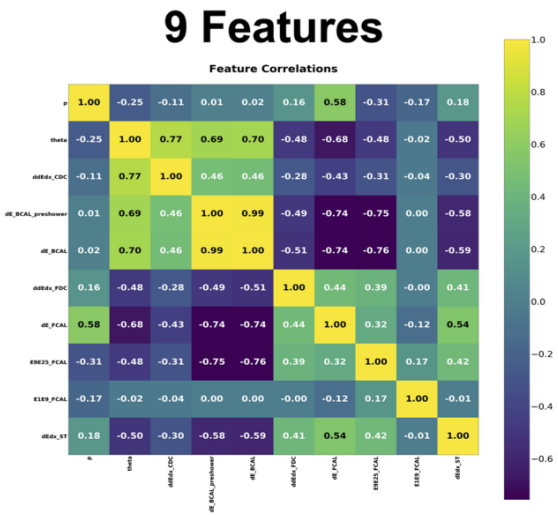
- Consists of two parts: encoder + decoder $\rightarrow$ Symmetric sandwich architecture
- Train autoencoder for: **Data In** $\approx$ **Data Out**
- **But:** Number of neurons needs to decrease in encoding part and increase in decoding part $\rightarrow$ Otherwise no effect $\leftrightarrow$ Identity function

Autoencoders: Basics



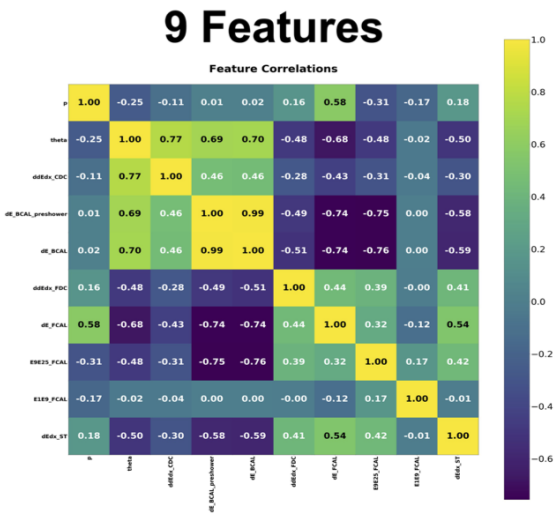
- Consists of two parts: encoder + decoder $\rightarrow$ Symmetric sandwich architecture
- Train autoencoder for: **Data In** $\approx$ **Data Out**
- **But:** Number of neurons needs to decrease in encoding part and increase in decoding part $\rightarrow$ Otherwise no effect $\leftrightarrow$ Identity function

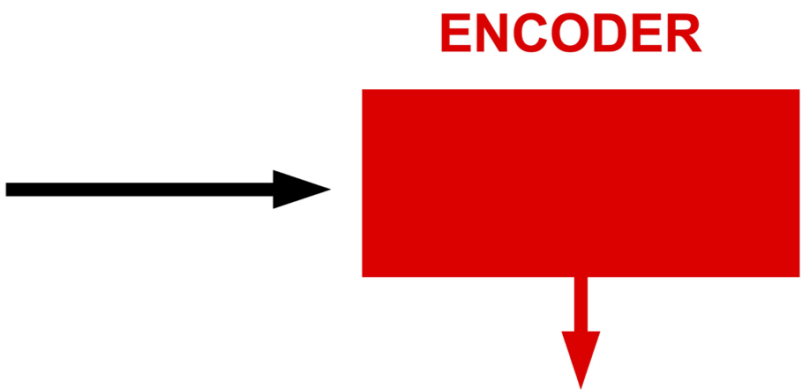
Autoencoders: Compression and Decompression



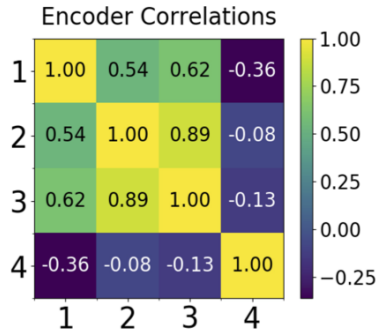
Features:
- momentum
- theta
- ddEdx(CDC)
- dE(BCAL 0)
- dE(BCAL all)
- ddEdx(FDC)
- dE(FCAL)
- E9E25 (FCAL)
- E1E9 (FCAL)

Autoencoders: Compression and Decompression



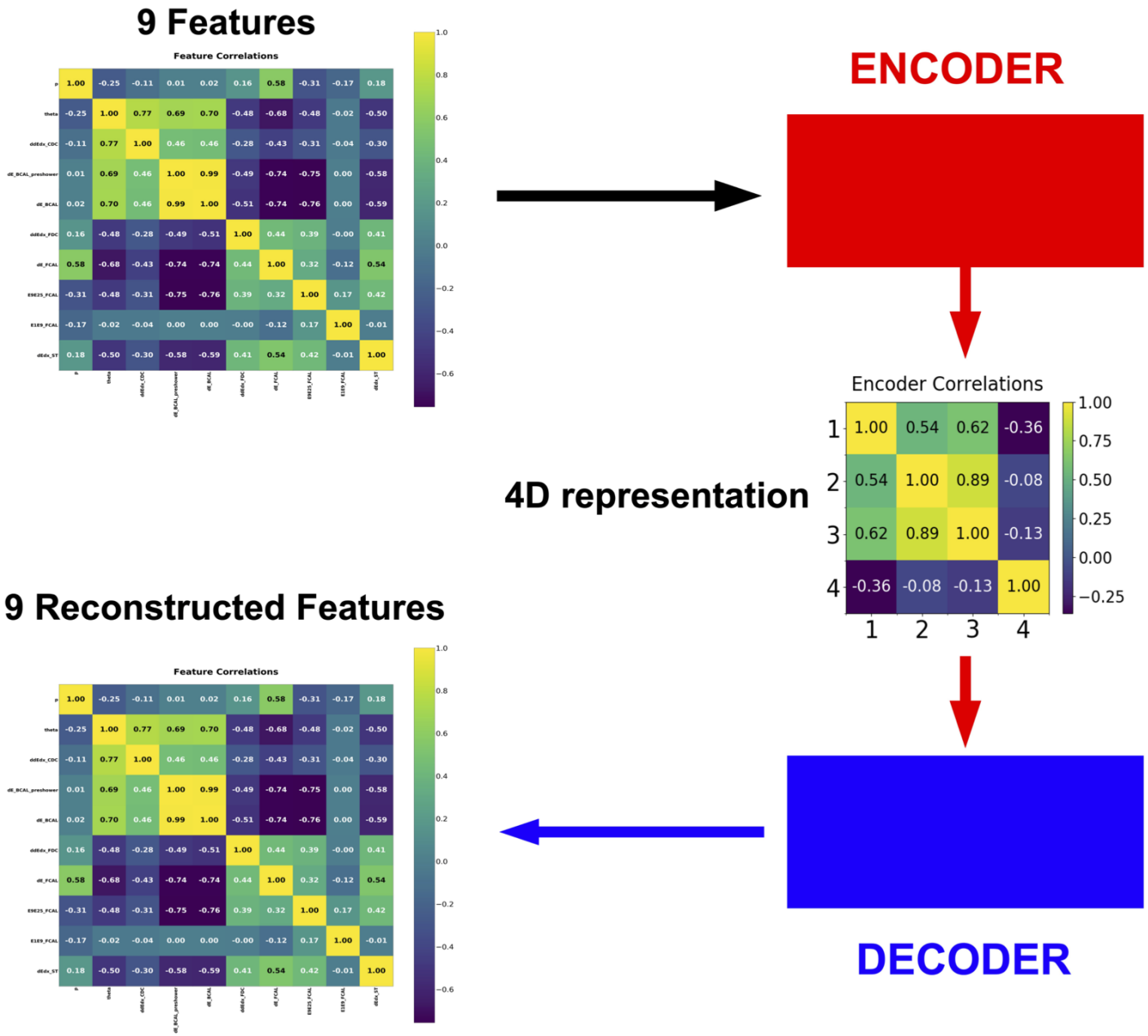


4D representation

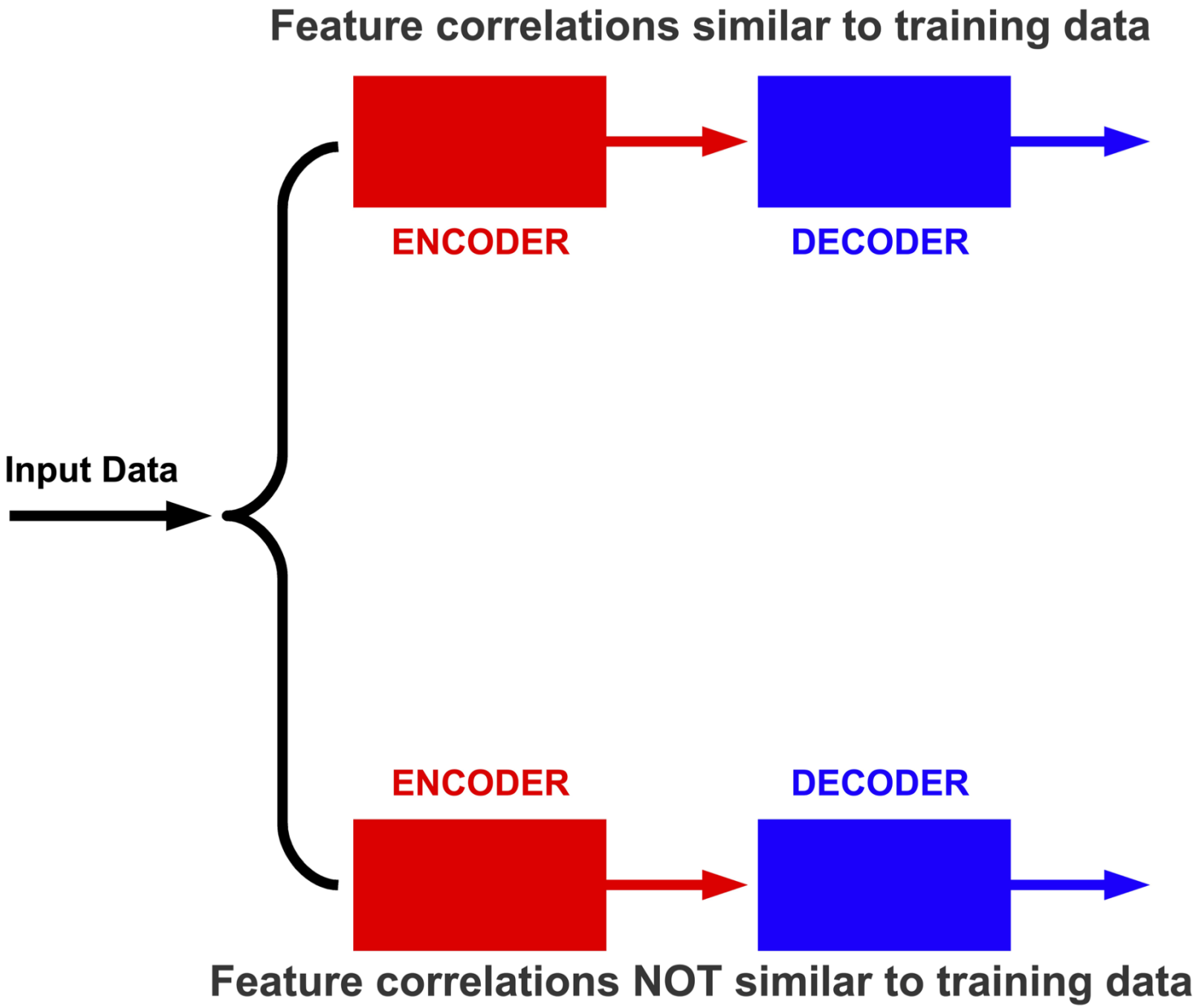


Compression / dimensional reduction
from 9 —> 4 dimensions

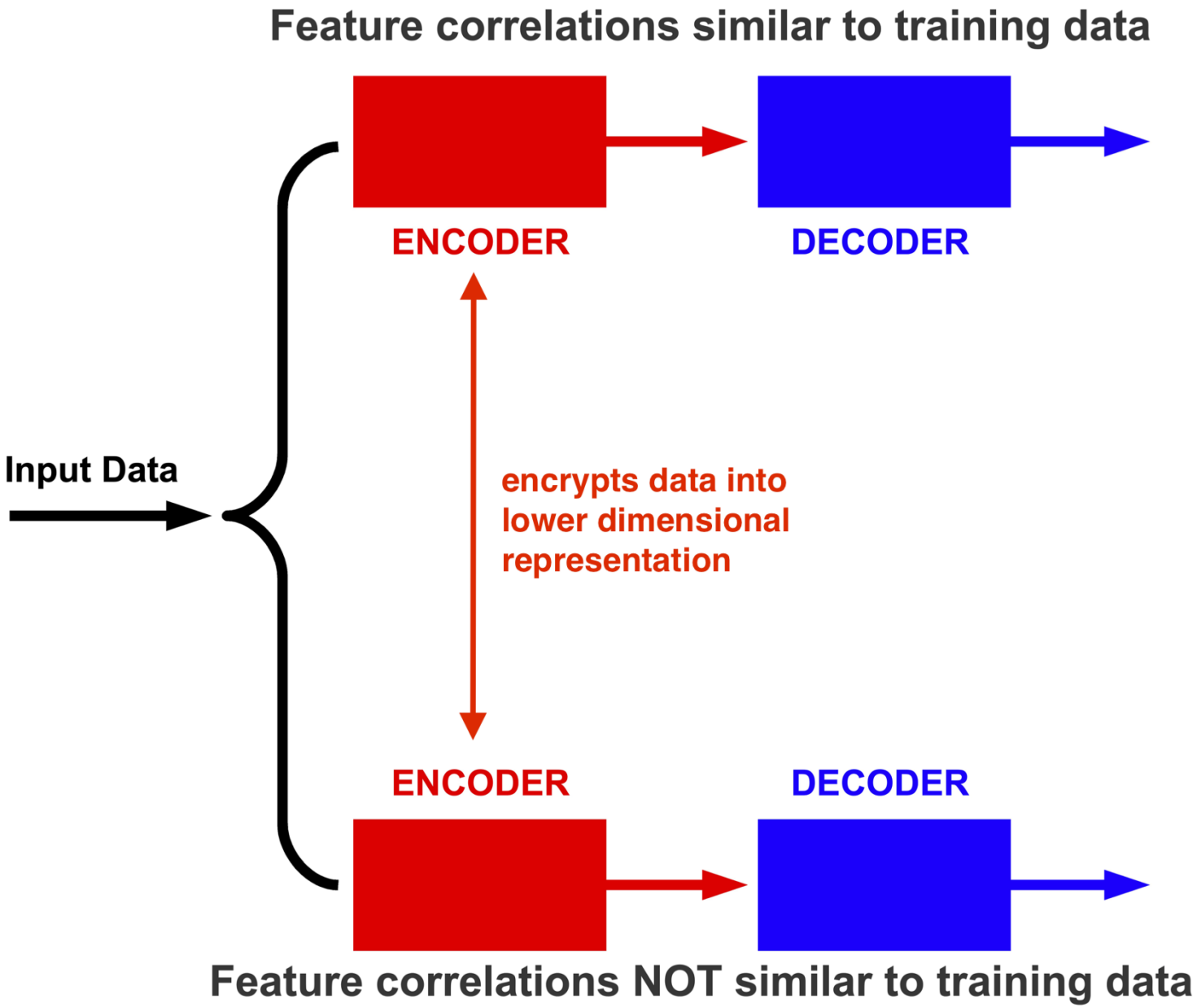
Autoencoders: Compression and Decompression



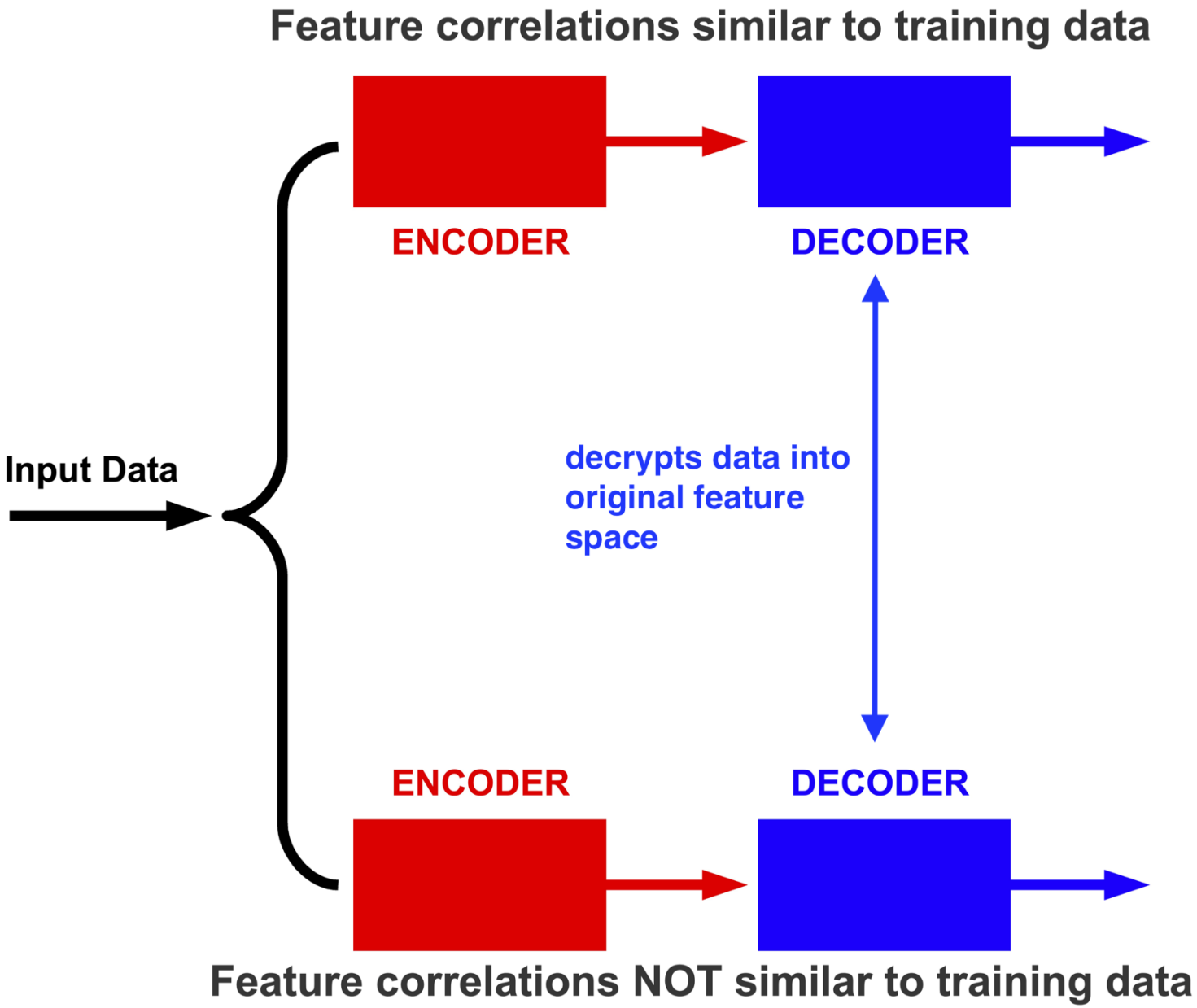
Autoencoders: Anomaly Detection in Data



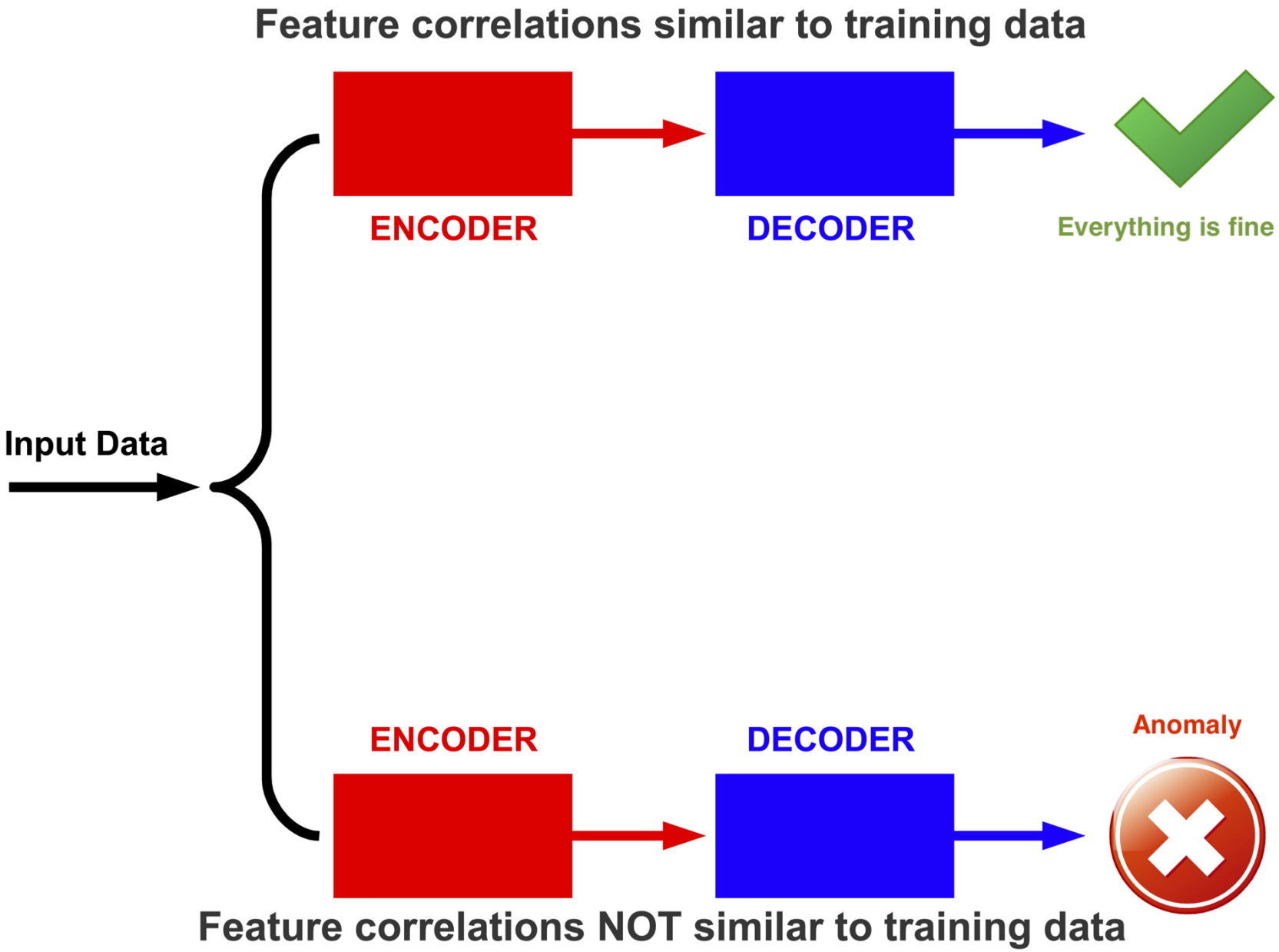
Autoencoders: Anomaly Detection in Data



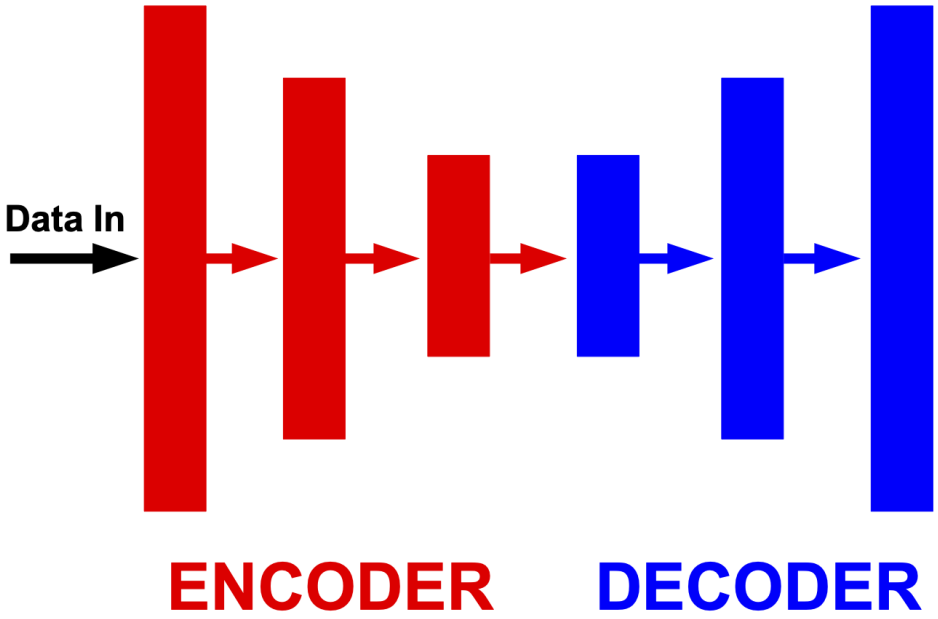
Autoencoders: Anomaly Detection in Data

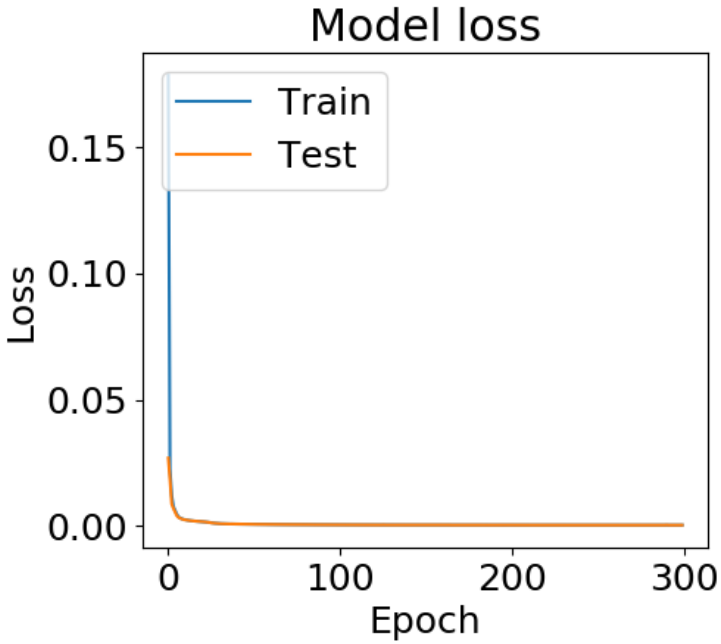


Autoencoders: Anomaly Detection in Data



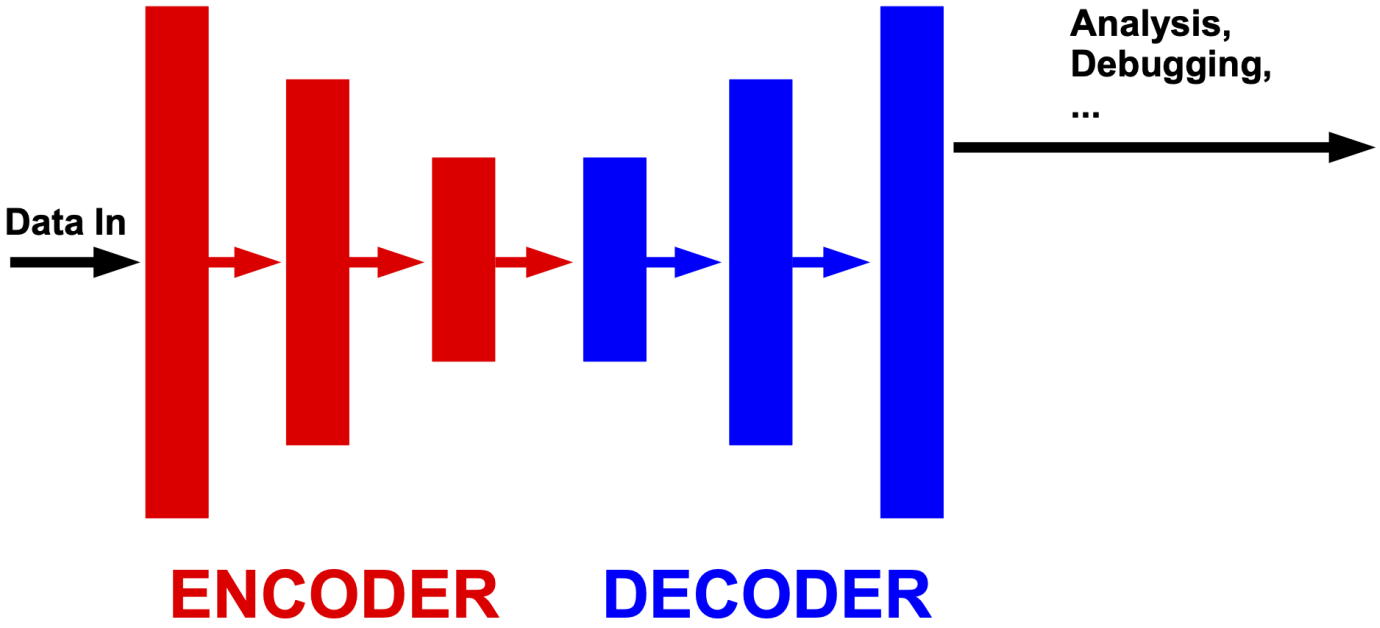
Autoencoders: (Current) Realization at GlueX





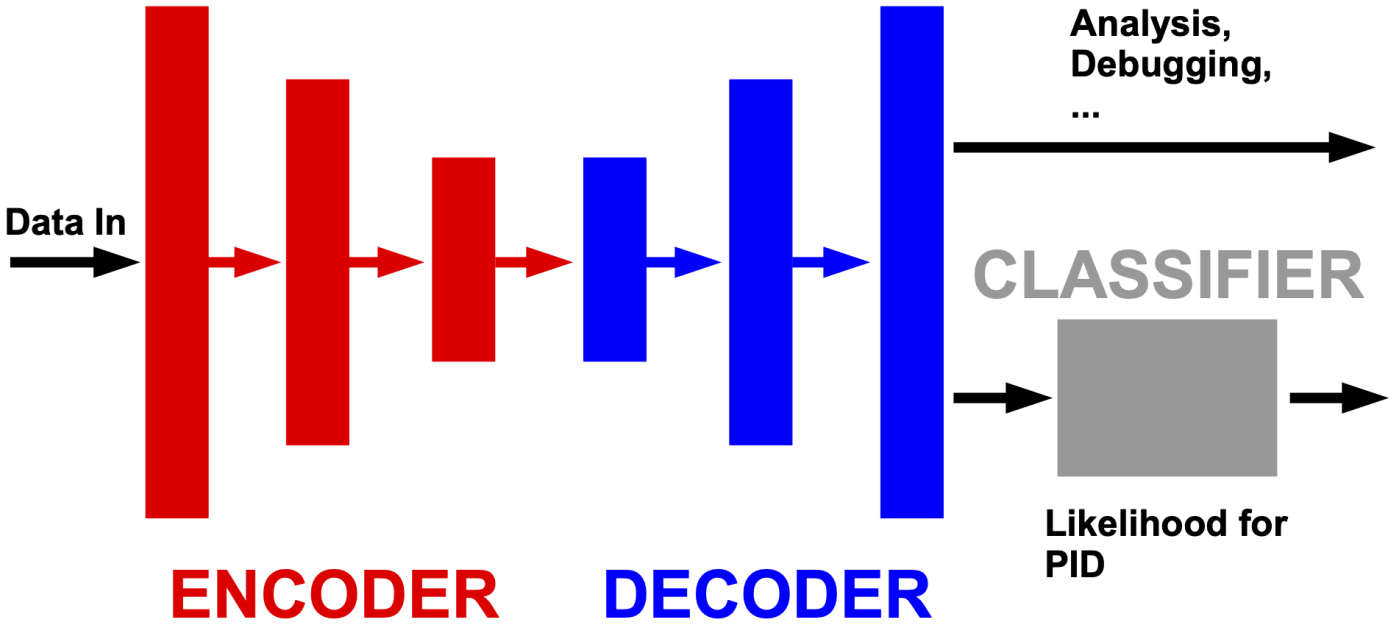
- Train autoencoder on simulated single particle tracks $\rightarrow$ One autoencoder per particle species and per charge (as apposed to "classic" machine learning)

Autoencoders: (Current) Realization at GlueX



- Train autoencoder on simulated single particle tracks $\rightarrow$ One autoencoder per particle species and per charge (as apposed to "classic" machine learning)
- Autoencoder outputs help to understand model performance

Autoencoders: (Current) Realization at GlueX



- Train autoencoder on simulated single particle tracks → One autoencoder per particle species and per charge (as apposed to "classic" machine learning)
- Autoencoder outputs help to understand model performance
- Anomaly detector (aka classifer) allows to translate the decoded data into a likelihood

Reference Models and Analysis Procedures

- It is very helpful to have a reference model[2], running in parallel to your machine / deep learning analysis

  $\Rightarrow$ Helps to understand / debug the model your are developing

[2] Preferably not machine learning based

Reference Models and Analysis Procedures

- It is very helpful to have a reference model[2], running in parallel to your machine / deep learning analysis

  $\Rightarrow$ Helps to understand / debug the model your are developing

- In this Analysis use:
  1. E/p-cut: Uses energy deposit E in either BCAL / FCAL and the momentum $p \rightarrow$ Not machine learning based and straight forward to apply
  2. "Classic" machine learning approach: Neural network trained on leptons vs. pions (one for each charge)

[2] Preferably not machine learning based

Reference Models and Analysis Procedures

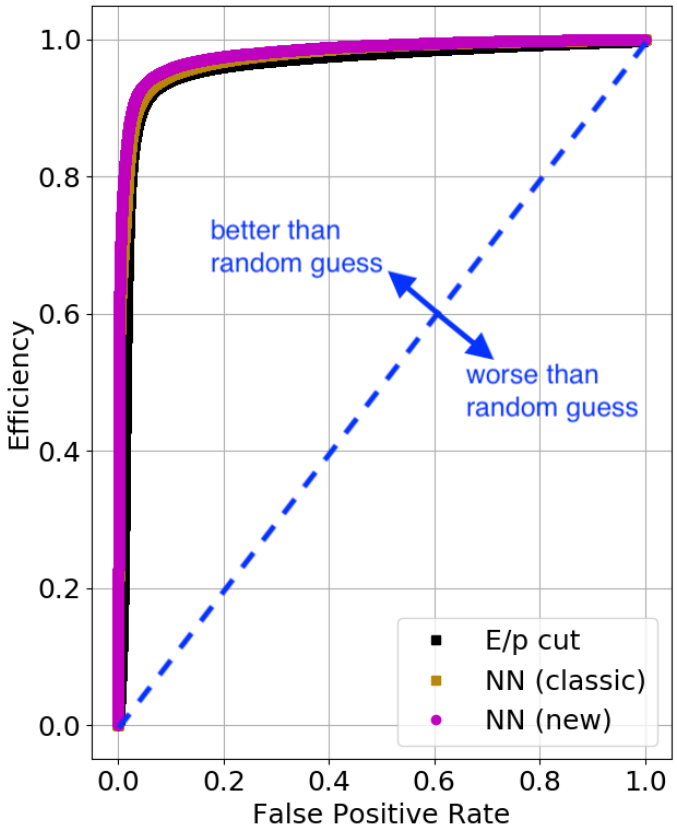
- It is very helpful to have a reference model[2], running in parallel to your machine / deep learning analysis

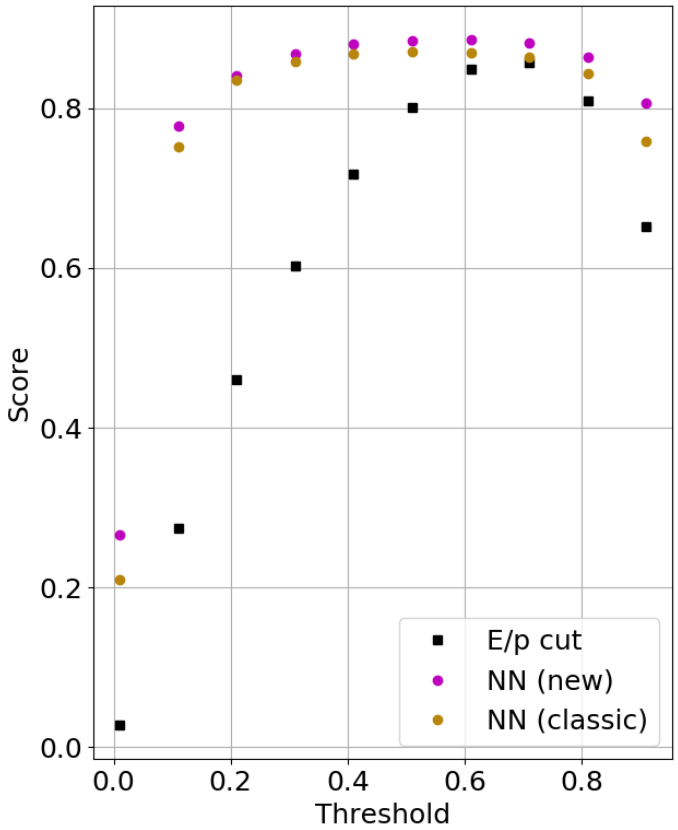
  $\Rightarrow$ Helps to understand / debug the model your are developing

- In this Analysis use:
  1. E/p-cut: Uses energy deposit E in either BCAL / FCAL and the momentum $p \rightarrow$ Not machine learning based and straight forward to apply
  2. "Classic" machine learning approach: Neural network trained on leptons vs. pions (one for each charge)
- Use Bayesian formalism to combine predictions from pion- and lepton-autoencoders into a likelihood

[2] Preferably not machine learning based

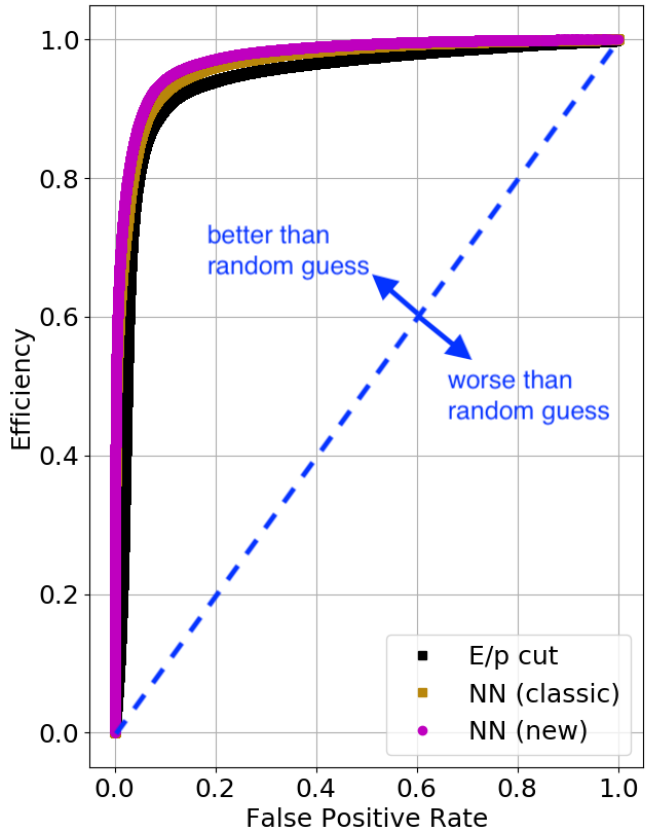
Lepton Identification Performance on simulated single e^+-/π^+-Tracks

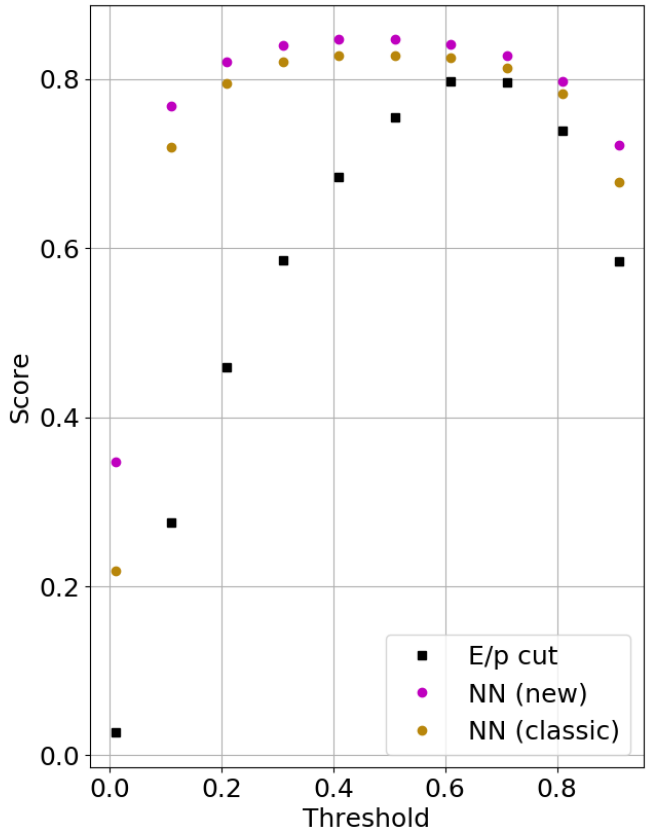




PID Method	Efficiency [%]	False Rate [%]	MCC-Score
E/p	92	6	0.86
Classic ML	93	6	0.87
Autoencoder	94	5	0.89

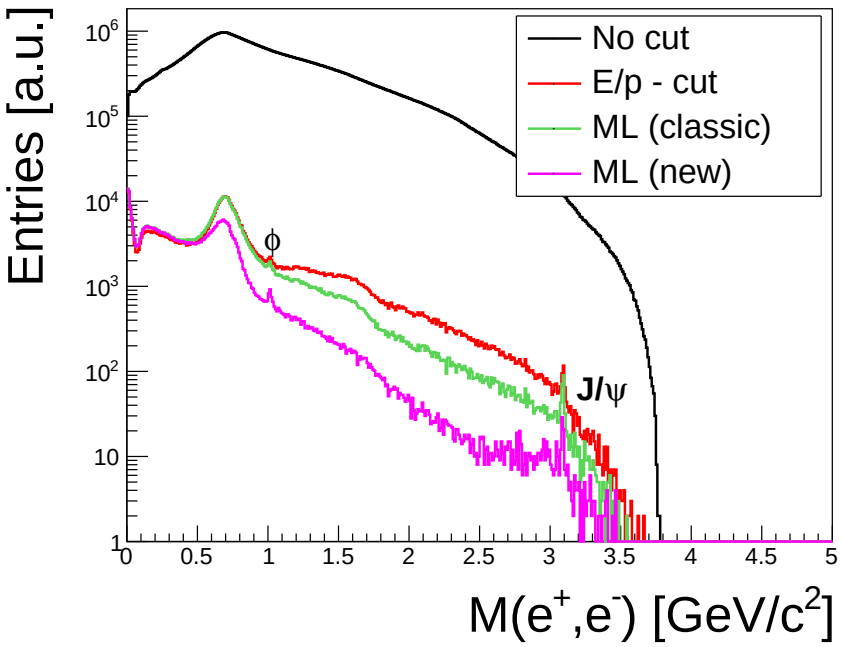
Lepton Identification Performance on simulated single e^--/π^--Tracks

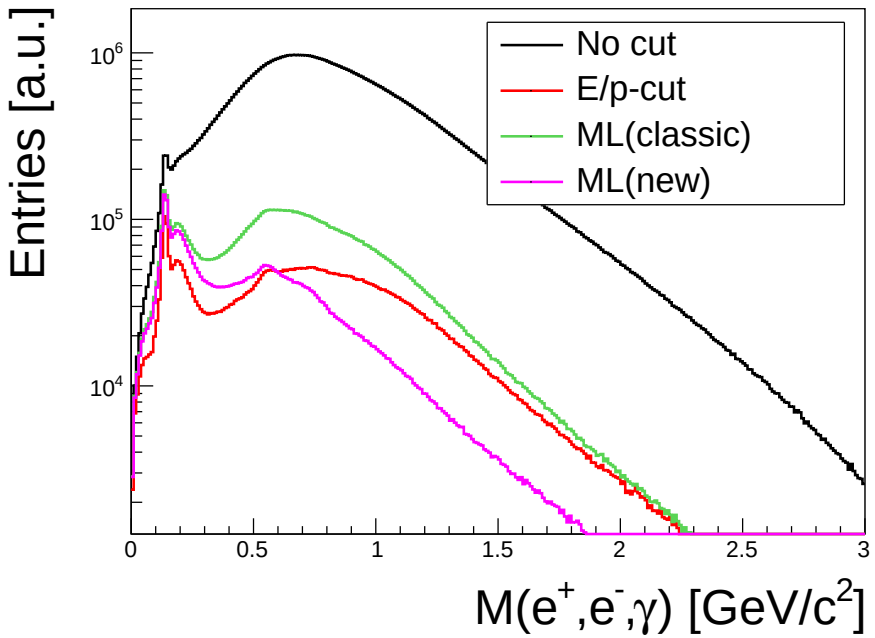




PID Method	Efficiency [%]	False Rate [%]	MCC-Score
E/p	90	10	0.8
Classic ML	91	8	0.83
Autoencoder	93	8	0.85

Application on GlueX Lepton Data

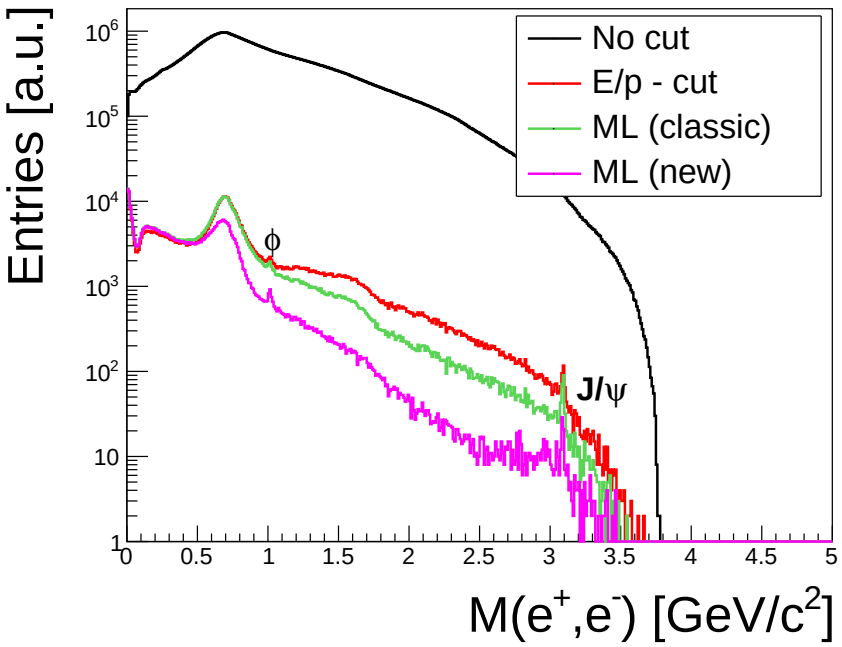


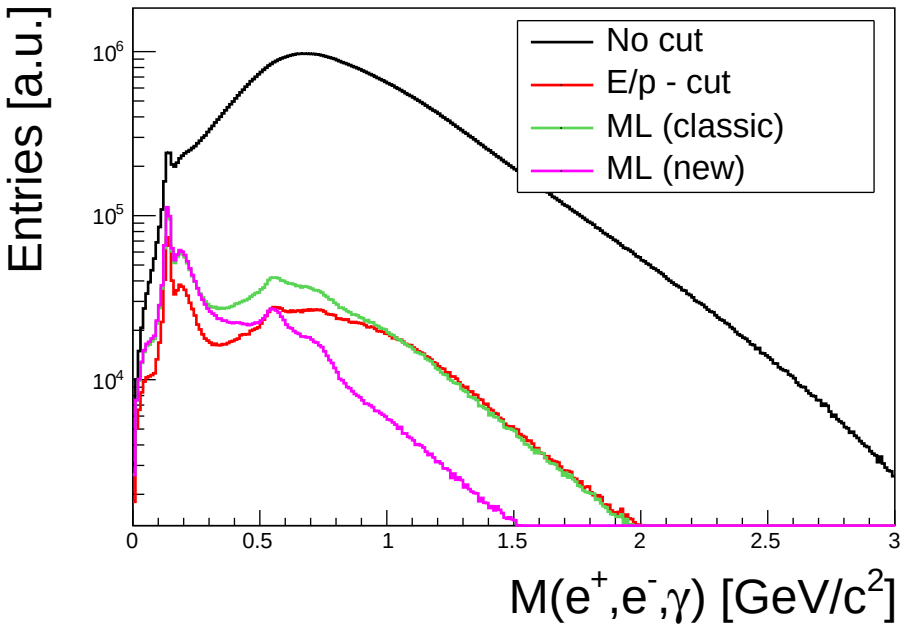


- Analyzed small sample of GlueX e^+e^--trees
- Did not tune cuts
- Found: $\Phi \to e^+e^-$ and $J/\psi \to e^+e^-$

- Analyzed small sample of GlueX $e^+e^-\gamma$-trees
- Used rather tight cuts: Networks $\gtrsim 70\%$, $E/p \in [0.8, 1.1]$
- Found: $\pi^0 \to e^+e^-\gamma$ and $\eta \to e^+e^-\gamma$

Application on GlueX Lepton Data

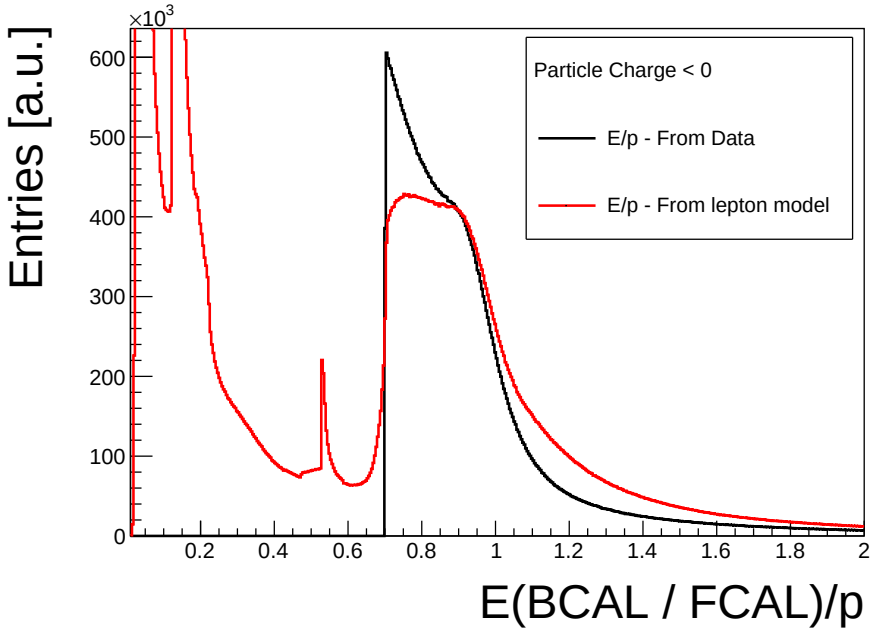


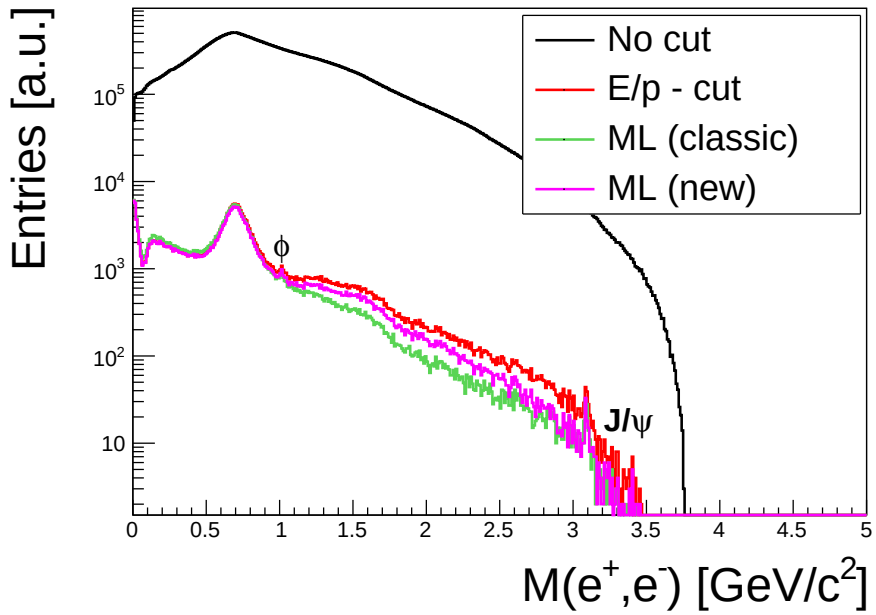


- Analyzed small sample of GlueX e^+e^--trees
- Did not tune cuts
- Found: $\Phi \to e^+e^-$ and $J/\psi \to e^+e^-$

- Analyzed small sample of GlueX $e^+e^-\gamma$-trees
- Used rather tight cuts: Networks $\gtrsim 90\%$, $E/p \in [0.85, 1.05]$
- Found: $\pi^0 \to e^+e^-\gamma$ and $\eta \to e^+e^-\gamma$

Predicted Lepton Features





- **Remeber:** $E/p \sim 1$ for leptons and predominantly ~ 0 for pions
- Autoencoder reconstructs E/p by taking correlations of provided features into account
  - $\Rightarrow$ basically a multidimensional fit
  - $\Rightarrow$ similar to a kinematic fit, but on a single track level!
- Cuts on predicted E/p allow for background suppression

Debugging / Understanding your Model

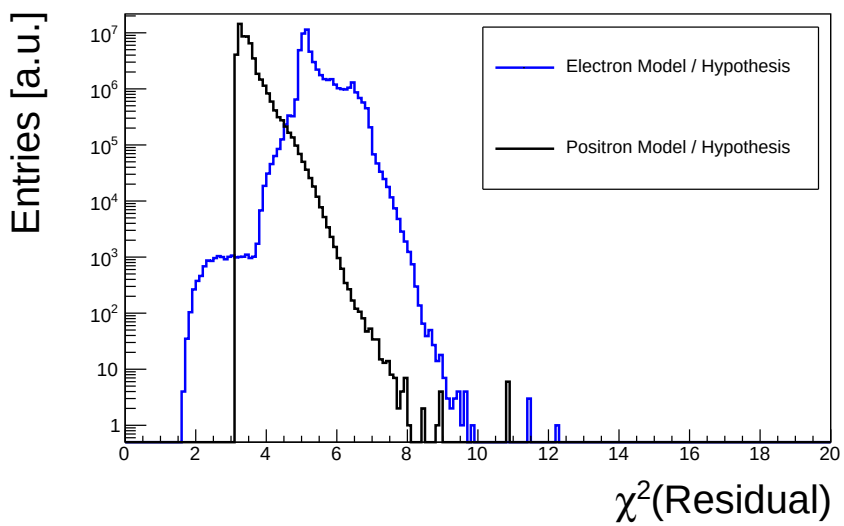
- Very important - not only for you, but also for the analyzer who is using your model

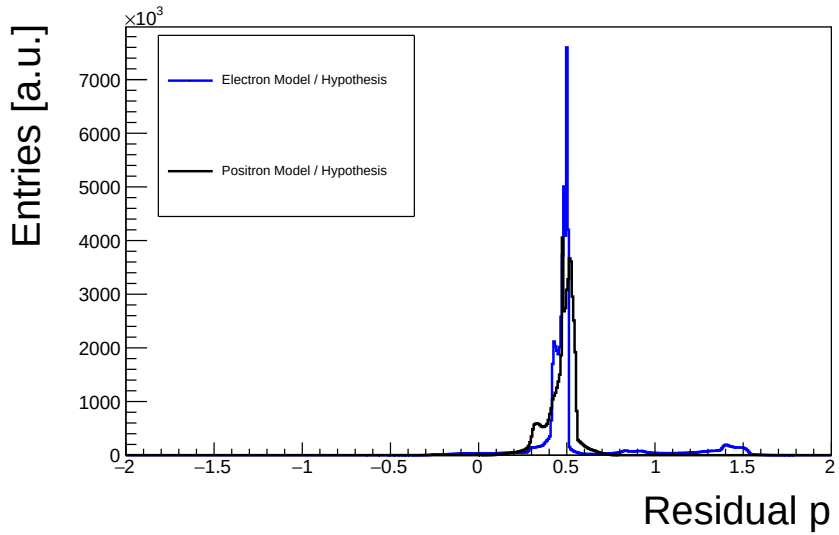
Debugging / Understanding your Model

- Very important - not only for you, but also for the analyzer who is using your model
- Autoencoder allows to formulate residuals: Network Input - Network Output

Debugging / Understanding your Model

- Very important - not only for you, but also for the analyzer who is using your model
- Autoencoder allows to formulate residuals: Network Input - Network Output
- Shown below: $\chi^2(\text{Residual}) = \sum_{i=1}^{N} \Big(\text{Network Input}[i] - \text{Network Output}[i]\Big)^2$ and a few residuals BEFORE a bug-fix

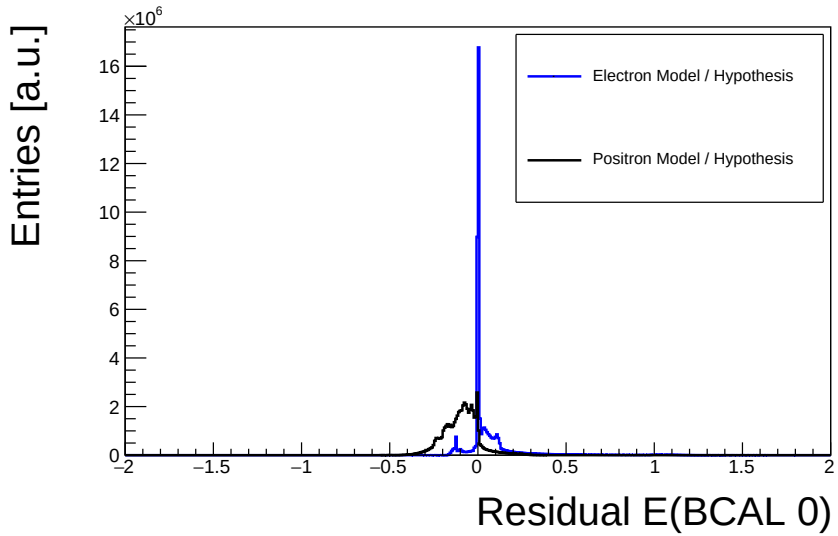


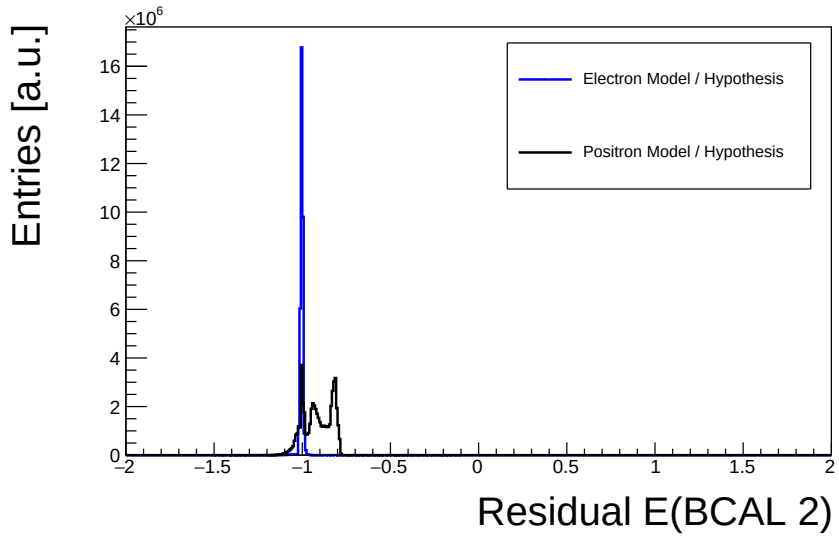


- χ^2 and residuals expected to be ≈ 0

Debugging / Understanding your Model

- Very important - not only for you, but also for the analyzer who is using your model
- Autoencoder allows to formulate residuals: Network Input - Network Output
- Shown below: $\chi^2(\text{Residual}) = \sum_{i=1}^{N} \Big(\text{Network Input}[i] - \text{Network Output}[i]\Big)^2$ and a few residuals BEFORE a bug-fix

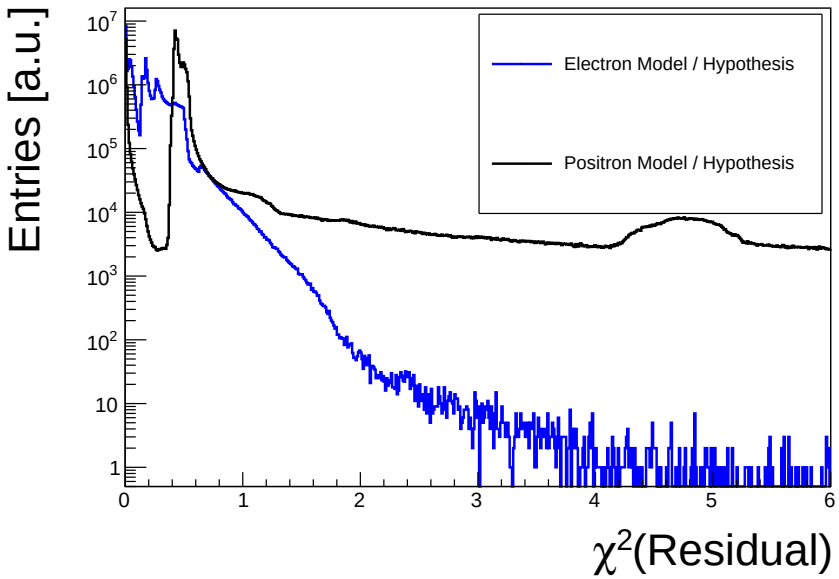


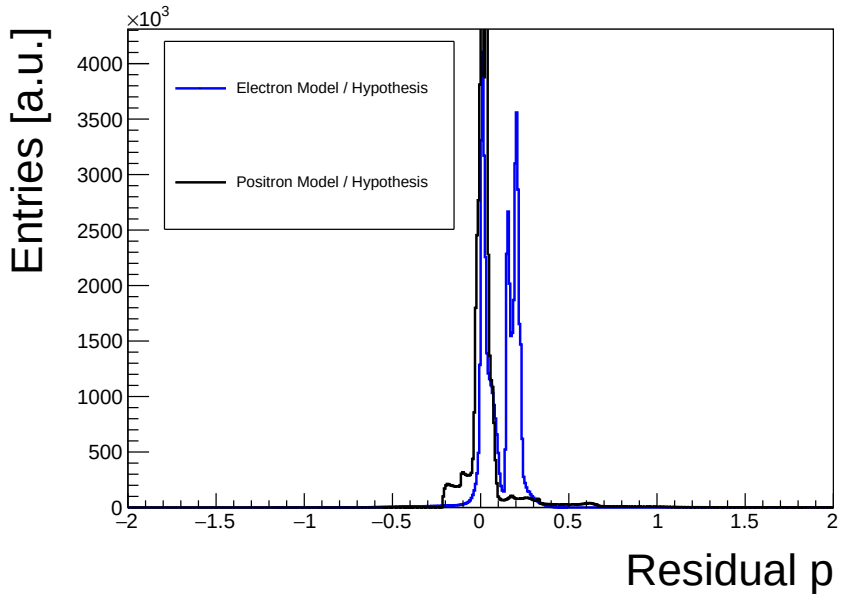


- Turned out that energy deposits of BCAL layers >=2 are -1 in DSelector

Debugging / Understanding your Model

- Very important - not only for you, but also for the analyzer who is using your model
- Autoencoder allows to formulate residuals: Network Input - Network Output
- Shown below: $\chi^2(\text{Residual}) = \sum_{i=1}^{N} \Big(\text{Network Input}[i] - \text{Network Output}[i]\Big)^2$ and a few residuals AFTER a bug-fix

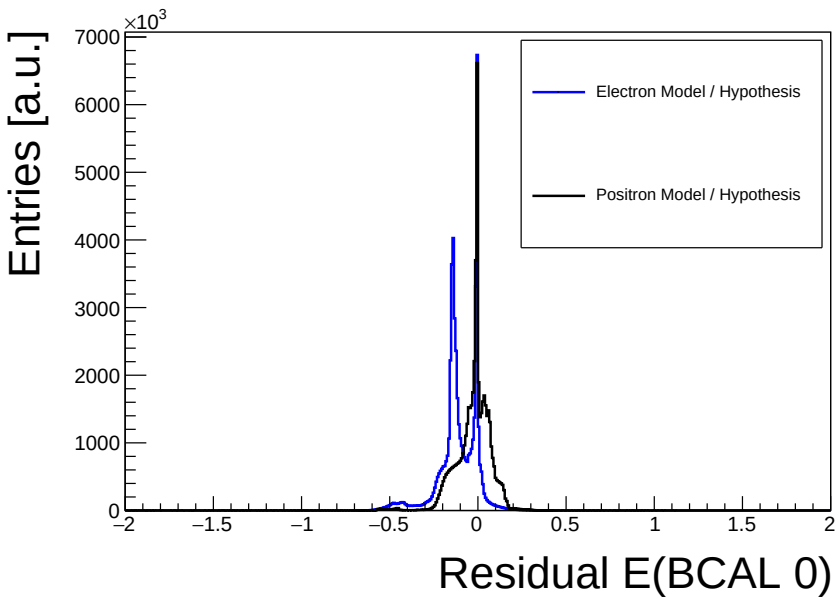


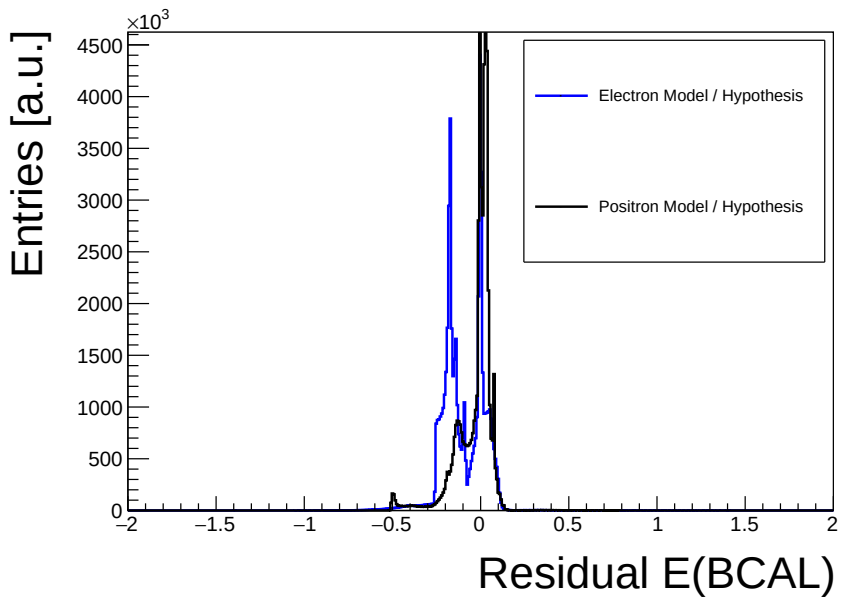


- Current model uses BCAL pre-shower and sum over all Layers only
- χ^2 and residuals are $\approx 0 \rightarrow$ Deviations due to pions and poorly reconstructed events

Debugging / Understanding your Model

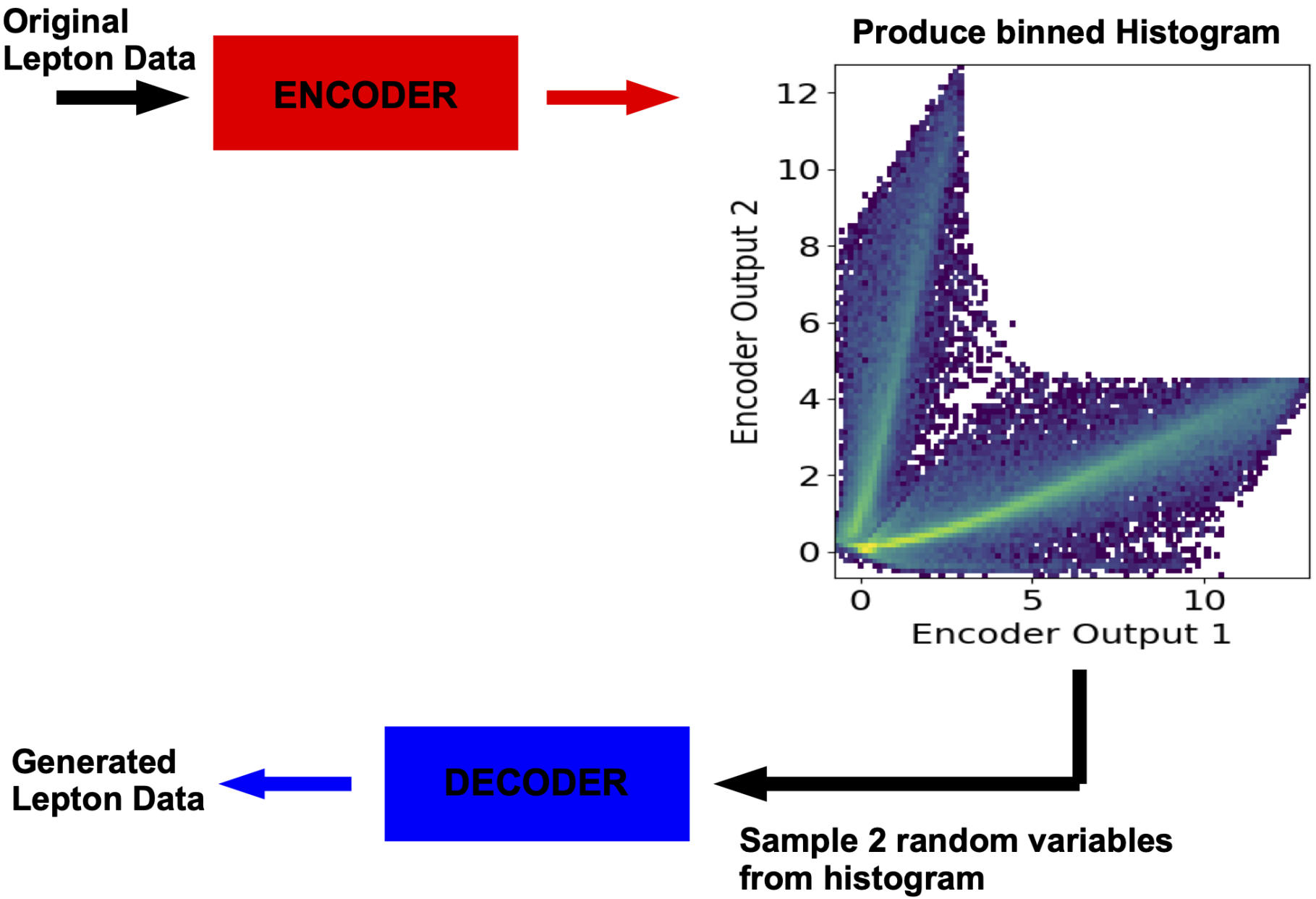
- Very important - not only for you, but also for the analyzer who is using your model
- Autoencoder allows to formulate residuals: Network Input - Network Output
- Shown below: $\chi^2(\text{Residual}) = \sum_{i=1}^{N} \Big(\text{Network Input}[i] - \text{Network Output}[i]\Big)^2$ and a few residuals AFTER a bug-fix



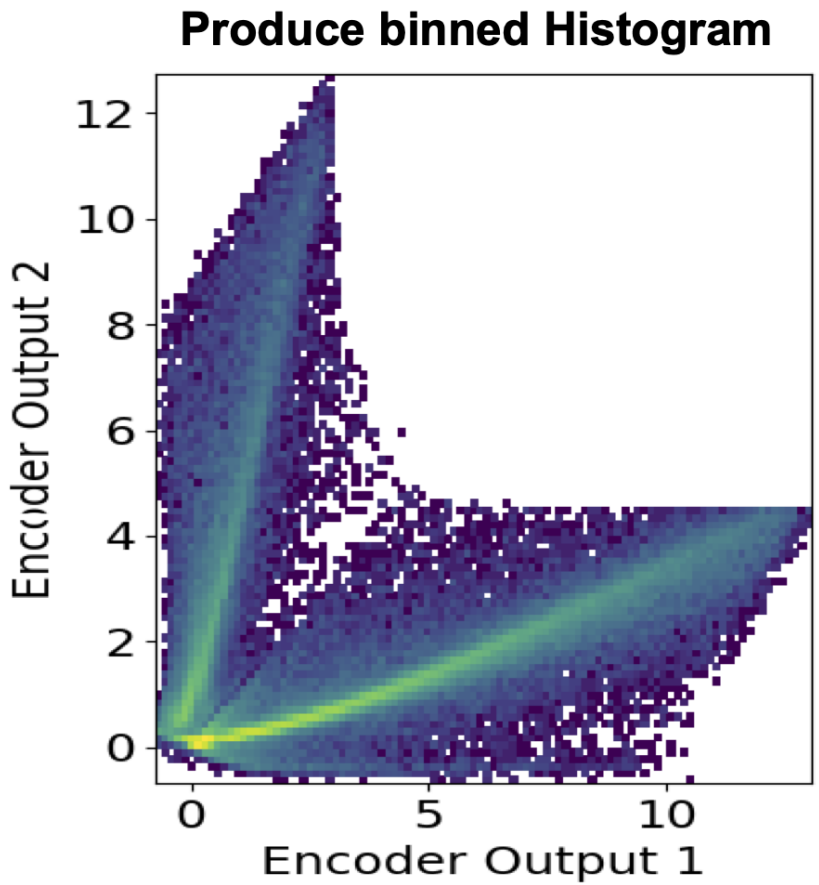


- Current model uses BCAL pre-shower and sum over all Layers only
- χ^2 and residuals are $\approx 0 \rightarrow$ Deviations due to pions and poorly reconstructed events

Inspecting the Decoder (first generation autoencoder)



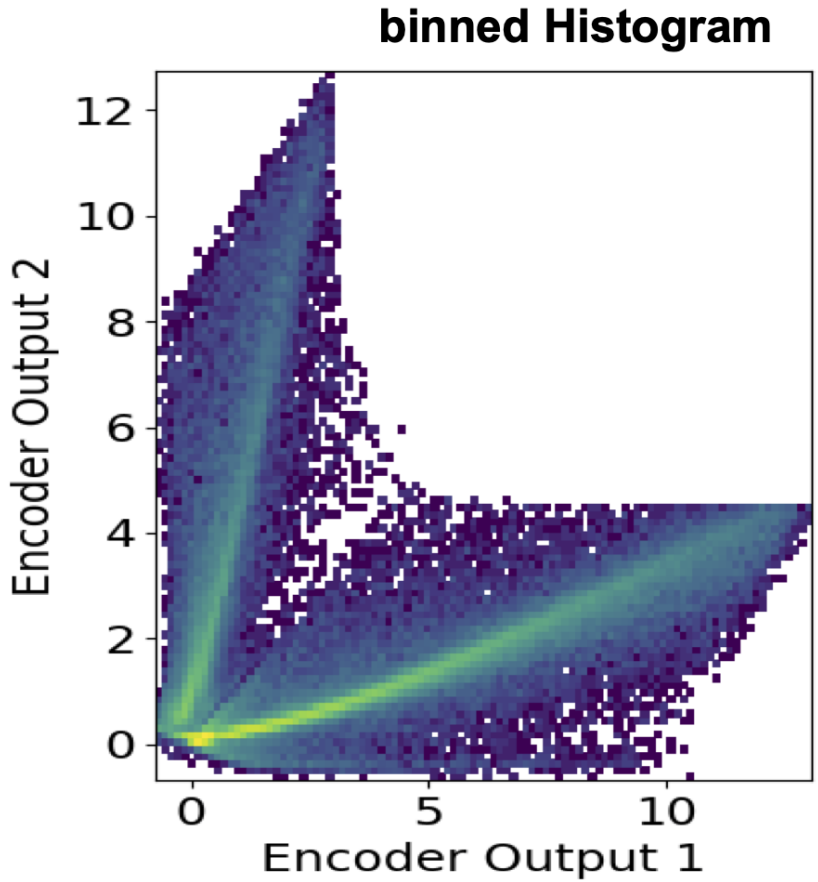
Inspecting the Decoder (first generation autoencoder)



Ideally done only once,
unless your model needs to be tuned…

Inspecting the Decoder (first generation autoencoder)

Can be done many times
—> data generator



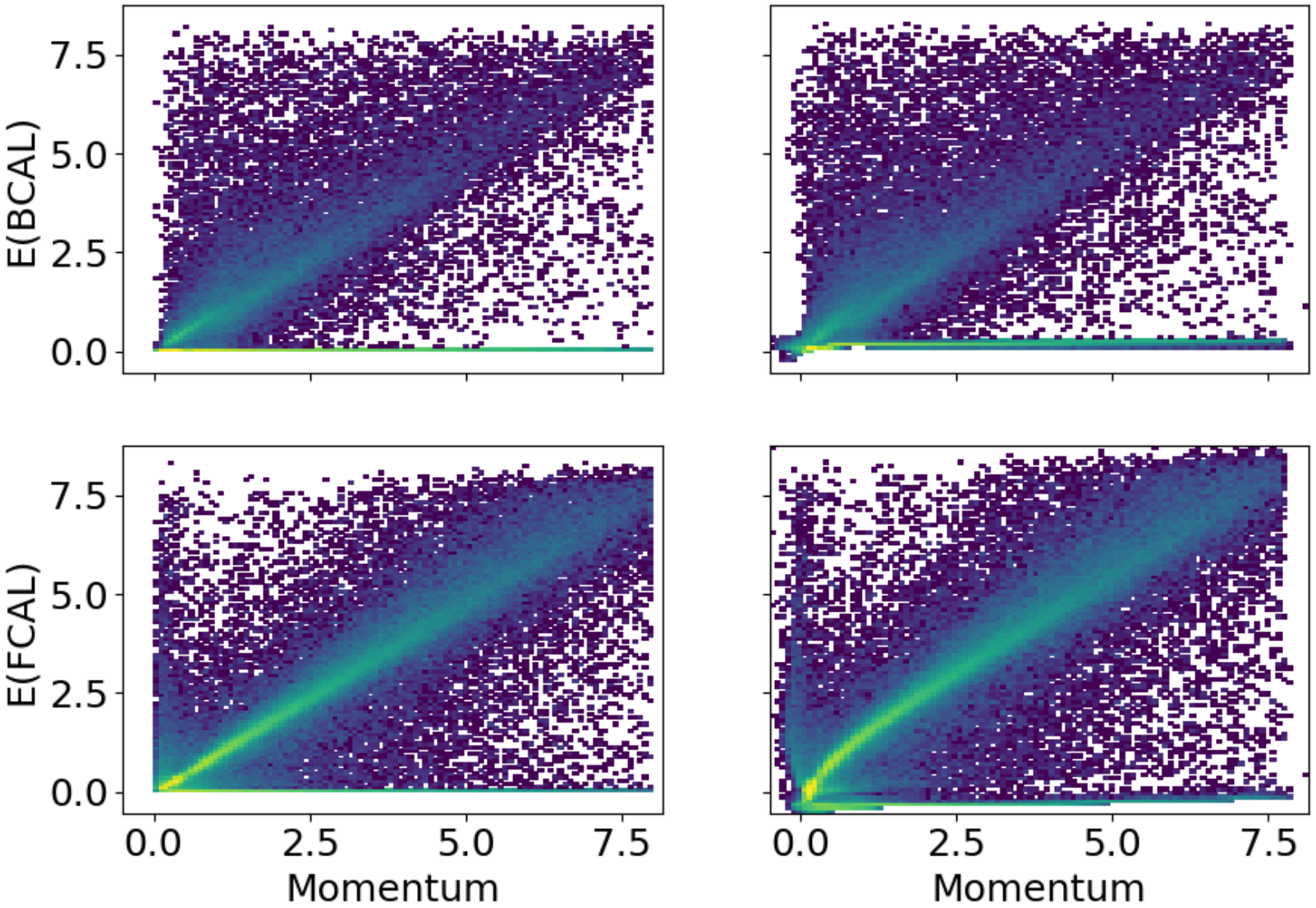
Generated Lepton Data

DECODER

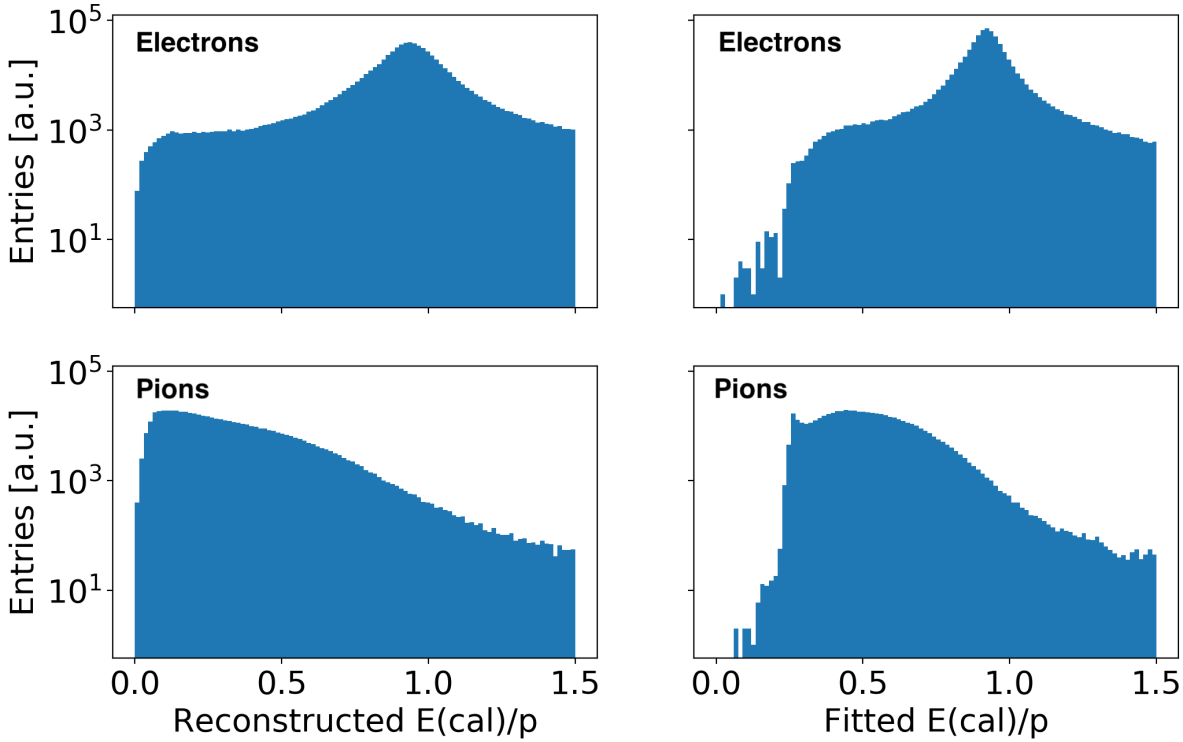
Sample 2 random variables from histogram

Generating Lepton (like) Data (first generation autoencoder)

Left: Original Data / Right: Generated Data



Moving forward: Constraint hypothesis fitting with Autoencoders



- Analogue to kinematic fitter: Include constraints related to particle hypothesis
- For leptons: Energy deposits in calorimeter $E(cal) \sim$ Momentum p
- Right column shows network predictions, using the lepton constraints
  $\Rightarrow$ Improvement in resolution!

Summary and Outlook

- Used autoencoder neural networks for PID in GlueX
- Obtained promising results
- Might be a bit too much for lepton ID, but:
  - the goal is to design a generic algorithm
  - want to proof and develop the concept on an "easy" case
- Besides simple classification, autoencoder networks...
  - ... support the user regarding debugging and understanding the model
  - ... interpret the data with respect to a given particle hypothesis (analogue to a kinematic fitter)
  - ... are easy to train due to the constraint architecture (used tied weights here)
  - ... de-noise the data (not shown here today)
  - ... are somewhat more robust with respect to training data / real data mismatch
- The price one has to pay is the model size: $\sim 10^3$ vs $\sim 10^2$ parameters