HUGS 2024

Machine Learning for Nuclear Physics

Lecture 1

Daniel Lersch

Tuesday, June 4, 2024

Jefferson Lab

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

On the Menu

- **Lecture 1**
  - Machine learning workflow
  - Neural networks
  - Deep learning
- **Lecture 2**
  - Network types and applications in Nuclear Physics
  - Methods and tools

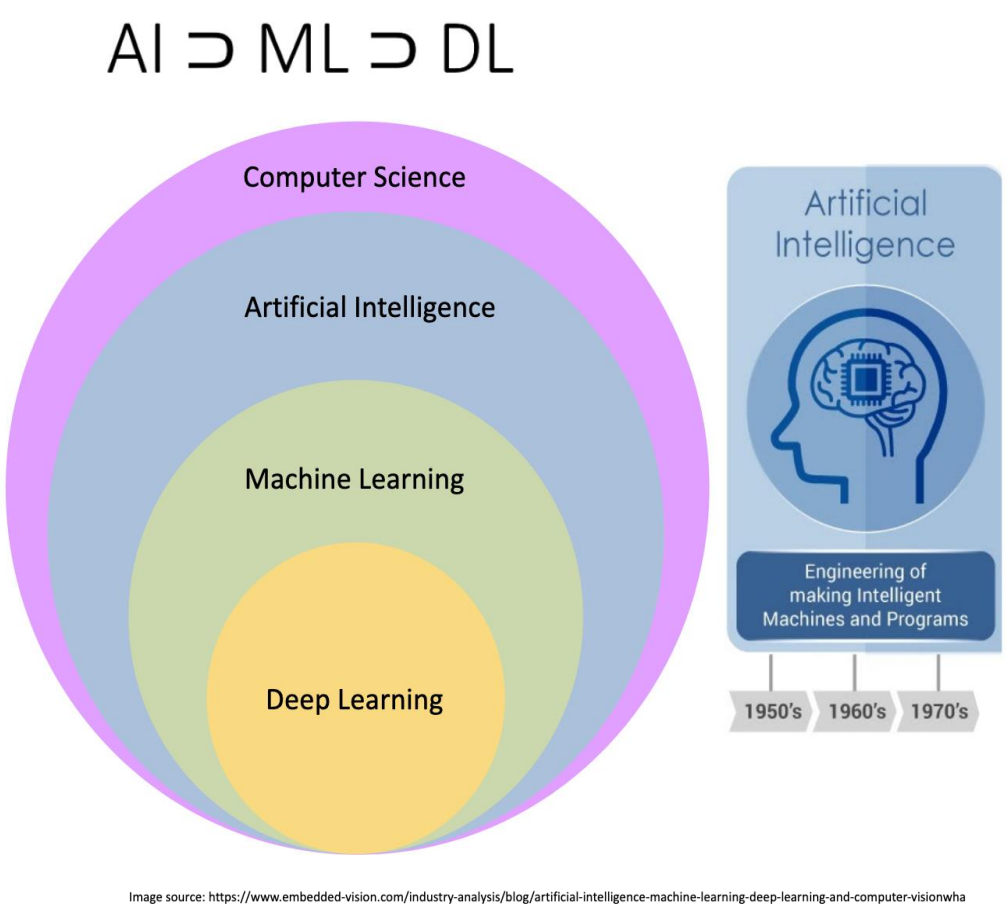


Image source: https://www.embedded-vision.com/industry-analysis/blog/artificial-intelligence-machine-learning-deep-learning-and-computer-visionwha

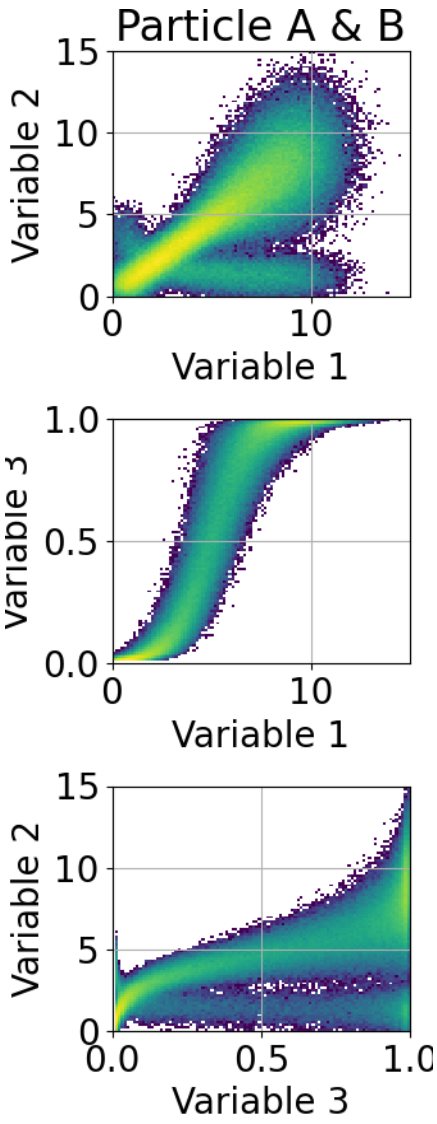
Plot taken from [Brenda Ngs talk at deep learning for science school 2019]

Machine Learning Workflow

Andrzej Kupsc: "Analysis is a matter of taste [...]"
Malachi Schram: "[...] but there are rules"

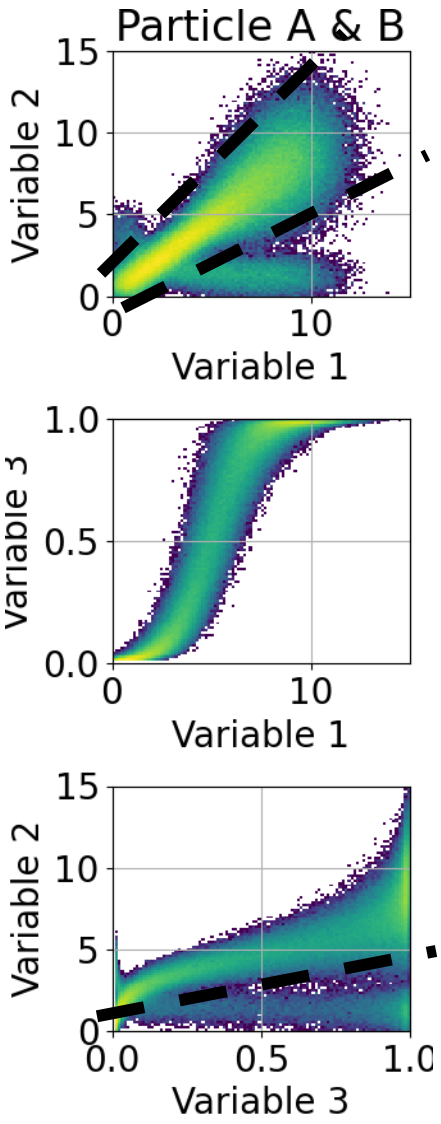
- "Typical" nuclear physics analysis
  - Particle Identification (PID)
  - Binary classification problem on a fake data set
- Basic idea behind machine learning
- Performance evaluation metrics

A Particle Identification (PID) Problem



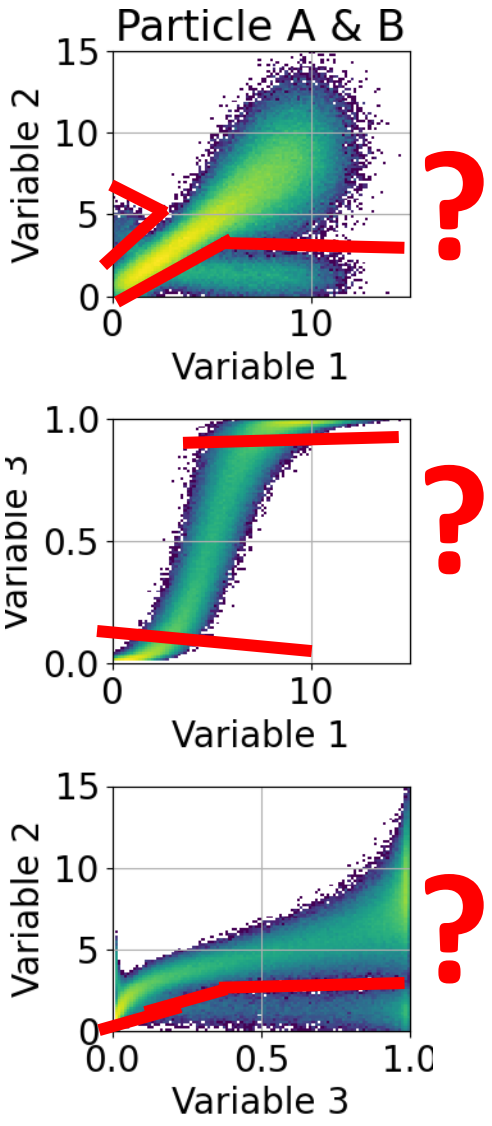
- Obtained data set from experiment(s)
  - 275 k events recorded with detector
  - Data set contains two particles A & B (**e.g. Pions and Kaons**)
  - Do not know which events correspond to which particle
  - Do not know exact abundancy of each particle type
- **Goal:** Identify particles A and B within given data set
  - Might need only one particle type for a specific analysis (**e.g. dalitz plot, cross section,..**)
  - Identified three variables suitable for PID
- **Approach:** Use Variable 1, 2 and 3 to identify each particle

What are we looking for?



- We could try to solve this "by hand"
- Use linear cuts to separate particle **(nothing wrong with this approach)**
- Only drawbacks:
  - Overlapping regions cause misidentification
  - Do not fully utilize (unknown) variable correlations --> Linear cut is too simple
- Spend more time on tuning the cuts --> Use a more complex function ?
- What is the underlying function that helps us to separate the two particles ?

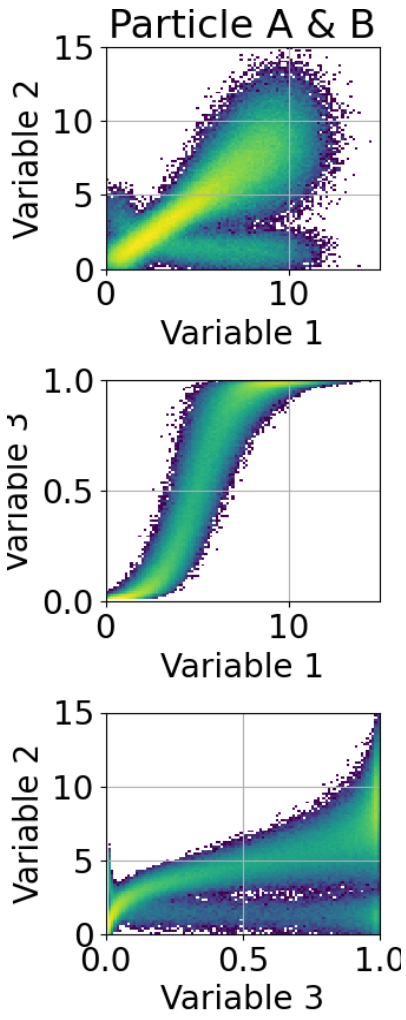
What are we looking for?



- We could try to solve this "by hand"
- Use linear cuts to separate particle **(nothing wrong with this approach)**
- Only drawbacks:
  - Overlapping regions cause misidentification
  - Do not fully utilize (unknown) variable correlations --> Linear cut is too simple
- Spend more time on tuning the cuts --> Use a more complex function ?
- **What is the underlying function that helps us to separate the two particles ?**

What are we looking for?

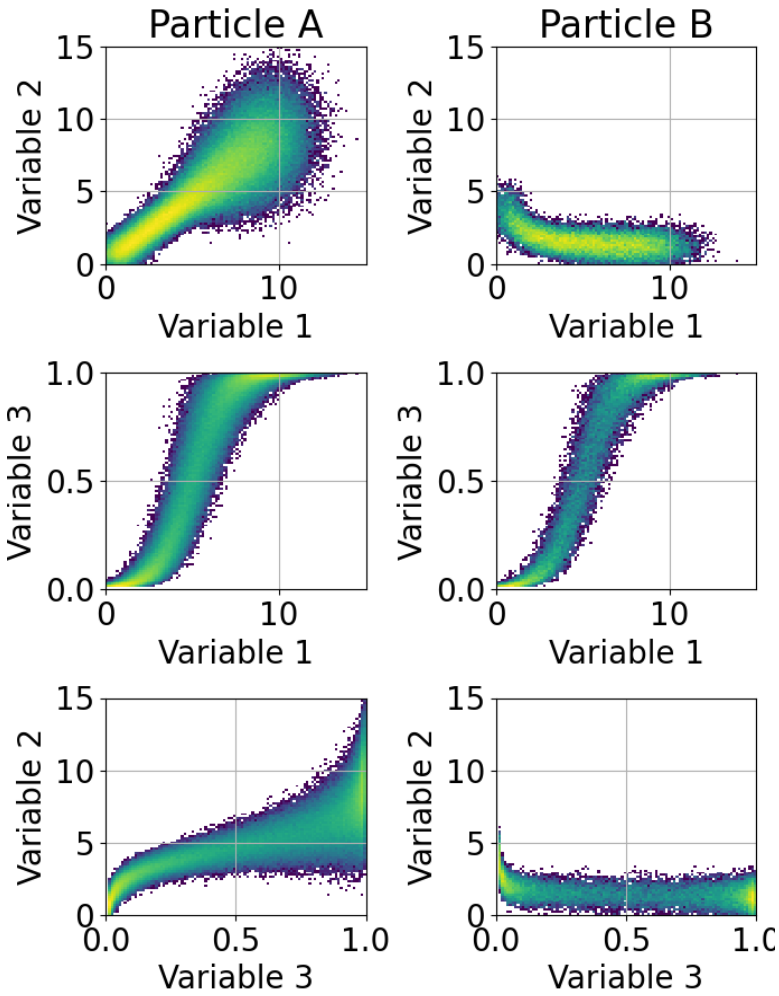
Model Input



Find a model that is complex enough to mimic the underlying function

Mysterious Model

Model Output



The Model

Input Data **Model** **Response**

- Model has internal parameters θ
- Response depends on input data and internal parameters: $\hat{Y} = f_\theta(X)$
- f_θ is, not necessarily, continuous and differentiable

The Model

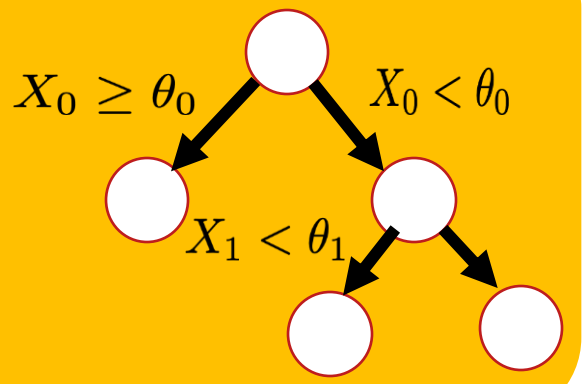
Input Data → **Model** → **Response**

X → f_θ → $\hat{Y}$

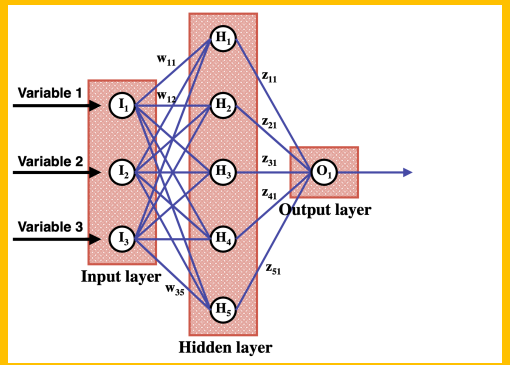
Linear Function

$$\hat{Y} = \theta_1 \cdot X + \theta_0$$

Decision Tree



Neural Network



and many more...

The Model

Input Data

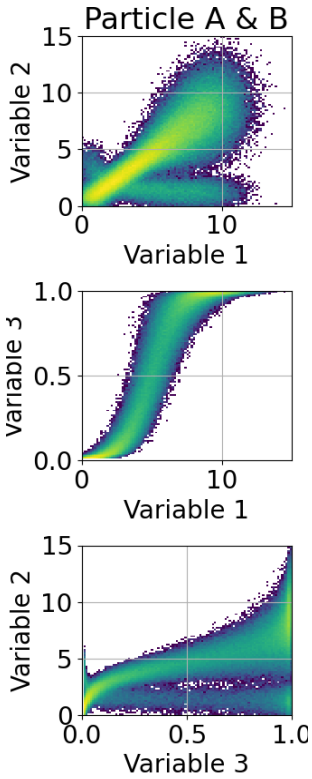
Model

Response

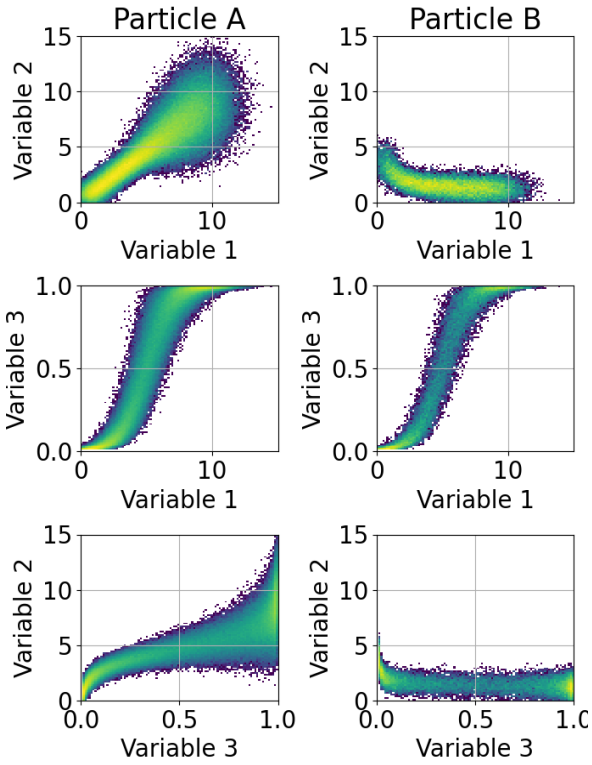
X

f_θ

$\hat{Y}$



How do we find these ?



Model Training / Fitting

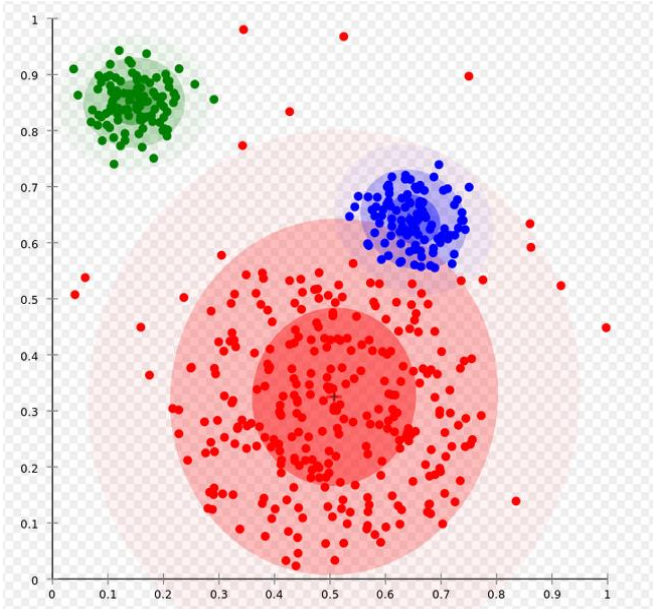
Input Data **Model** **Response**

- Find θ that maximize / minimize objective function $F(\hat{Y})$
- F could be χ^2, mean squared error, likelihood,...
- **Supervised Learning**
  - $F(\hat{Y}) = F(\hat{Y}, Y)$
  - Y are known targets (e.g. labels)
- **Unsupervised Learning**
  - No (or unknown) targets
  - Clustering algorithm $F(\hat{Y}) \propto$ Distance
  - Autoencoder Models $F(\hat{Y}) = F(\hat{Y}, X)$

Model Training / Fitting

Input Data **Model** **Response**

- Find θ that maximize / minimize objective function $F(\hat{Y})$
- F could be χ^2, mean squared error, likelihood,...
- **Supervised Learning**
  - $F(\hat{Y}) = F(\hat{Y}, Y)$
  - Y are known targets (e.g. labels)
- **Unsupervised Learning**
  - No (or unknown) targets
  - Clustering algorithm $F(\hat{Y}) \propto$ Distance
  - Autoencoder Models $F(\hat{Y}) = F(\hat{Y}, X)$



Model Training / Fitting

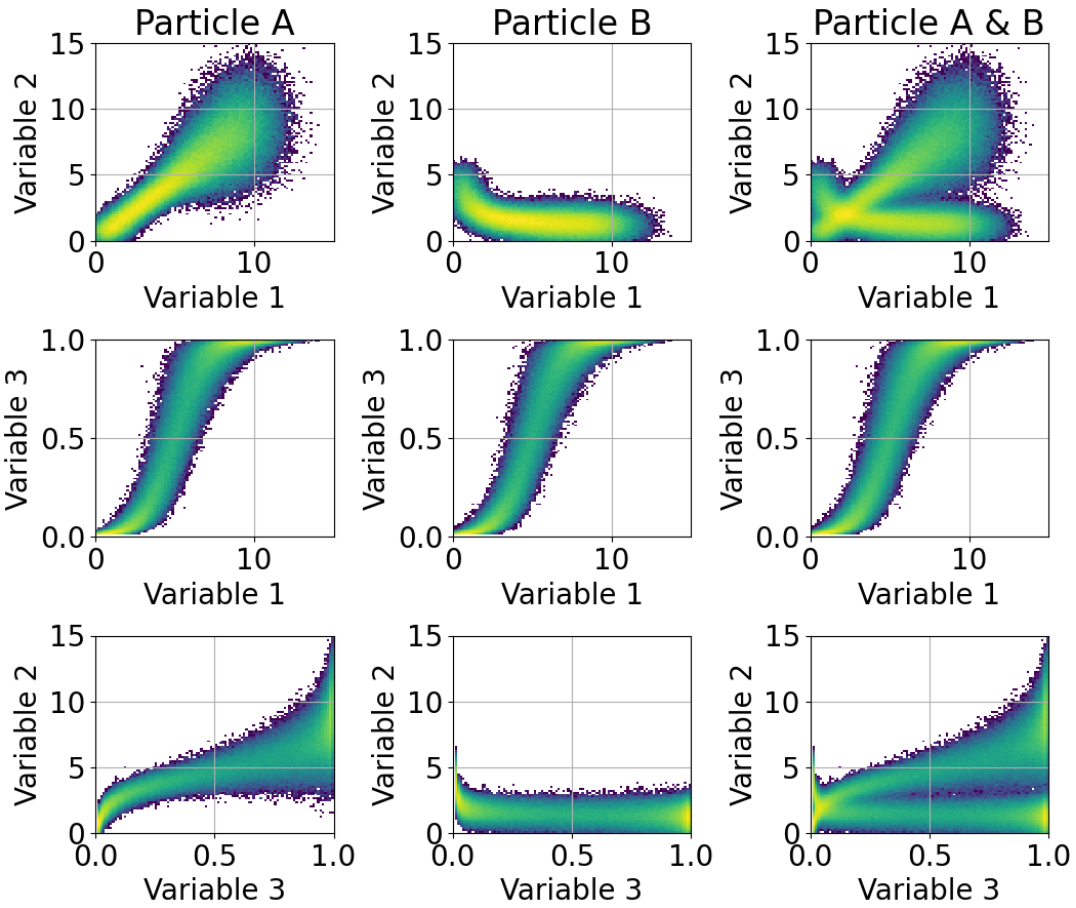
Input Data **Model** **Response**

- Find θ that maximize / minimize objective function $F(\hat{Y})$
- F could be χ^2, mean squared error, likelihood,...
- **Supervised Learning**
  - $F(\hat{Y}) = F(\hat{Y}, Y)$
  - Y are known targets (e.g. labels)

Today's focus

- **Unsupervised Learning**
  - No (or unknown) targets
  - Clustering algorithm $F(\hat{Y}) \propto$ Distance
  - Autoencoder Models $F(\hat{Y}) = F(\hat{Y}, X)$

Training Strategy for our PID Problem



- Use dedicated training data set where we know which event corresponds to what particle
- Data set could be MC simulations or well curated measured data
- Events in data set are labeled

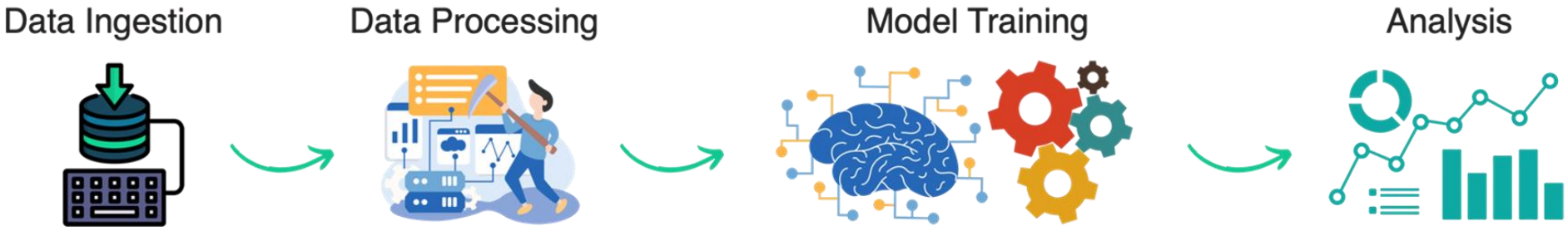
$$\text{Label } \ell = \begin{cases} 0, & \text{if event is particle A,} \\ 1, & \text{if event is particle B} \end{cases}$$

- 50% of data correspond to particle A and remaining 50% to particle B
- Let model learn labels from training data

$$\text{model}[\text{Variable } 1,2,3] = \ell$$

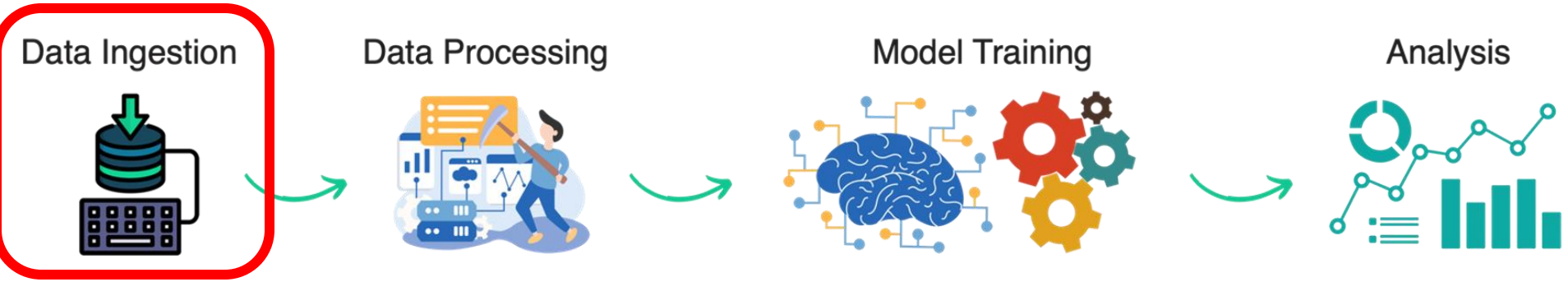
- **Apply trained model on data set from experiment**
- Assume that model generalizes well enough

A typical Workflow

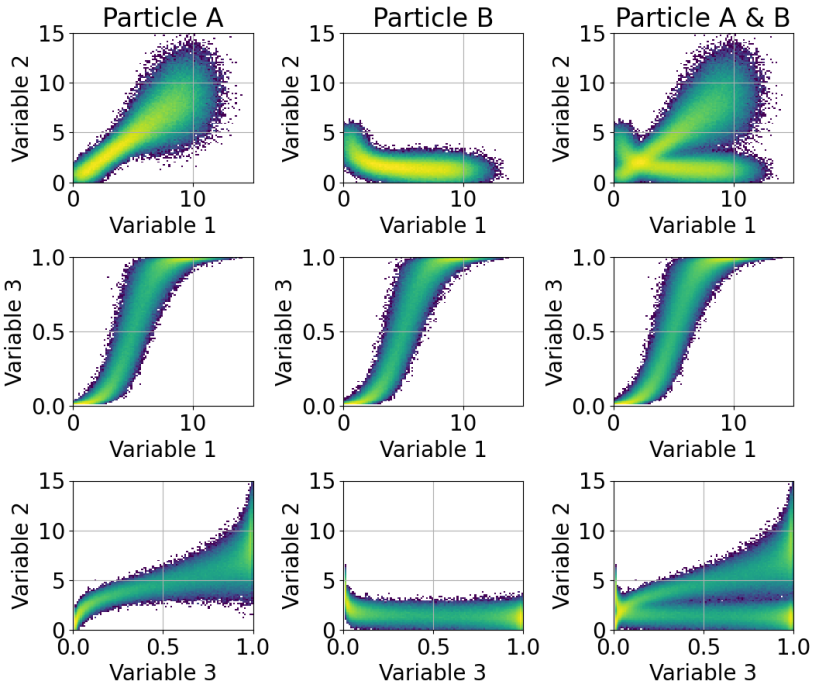


- Nearly every machine / deep learning analysis is based on these four steps
- Use scikit-learn for our PID example workflow
- Efforts in JLab Data Science group
  - Standardize machine / deep learning analyses --> Enforce reproducibility and support collaborative efforts
  - Develop generic framework

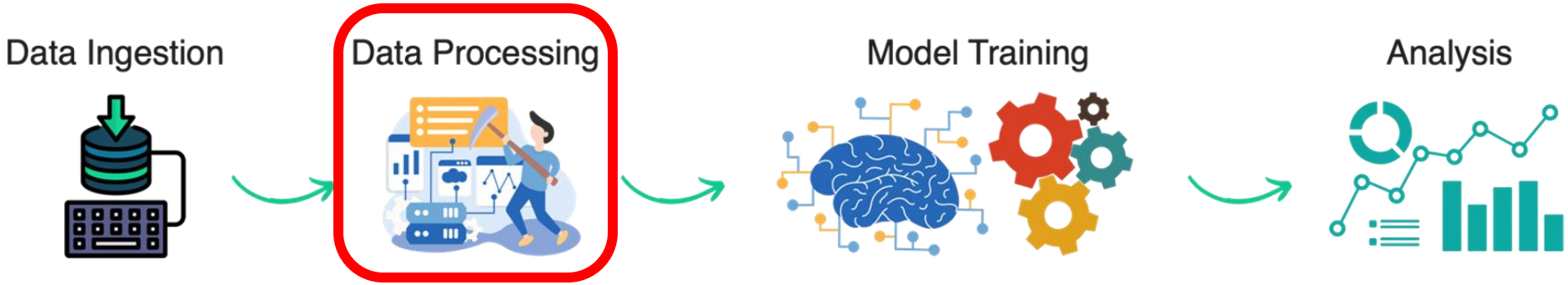
A typical Workflow



- Load data (from database, numpy array, ROOT-trees,…)
- Data types
  - Digits
  - Images
  - Videos
  - Texts
- Commonly used data formats
  - .png files
  - .npy arrays (numpy)
  - .csv, .json (dataframes) **(used for our example)**



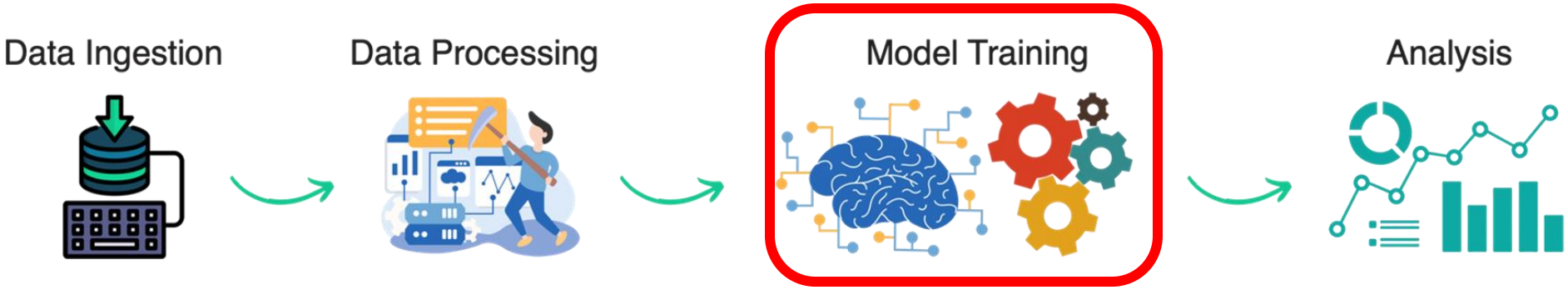
A typical Workflow



- Make sure that model can use data
- Feature engineering
- <u>For our PID problem:</u> Scale all variables to be between 0 and 1

Processing Method	Example	Why?
Adjust feature ranges	Do not feed vector (0.001,10000,40) into model	Model is likely to focus on large values
Exclude values	Acceptance holes in detector	Model may reconstruct false correlations
Select features	Particle energies, angles,…	Feed "useful" information to model

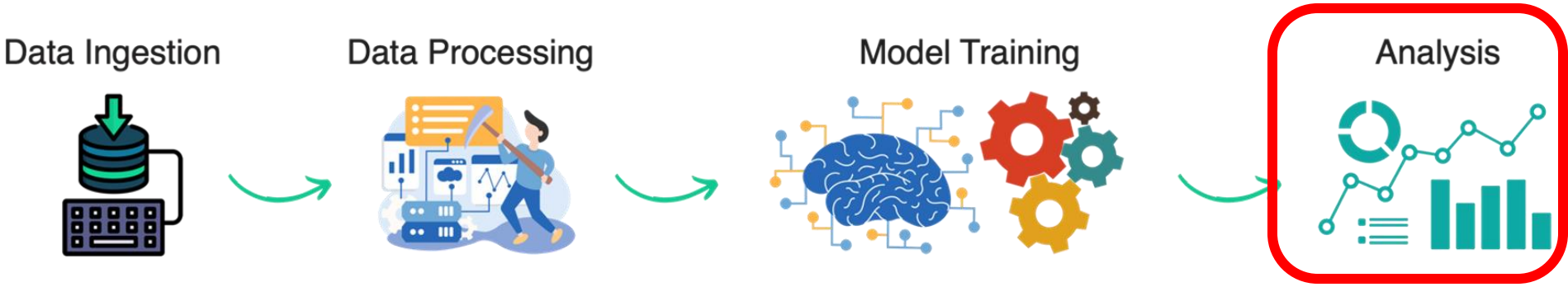
A typical Workflow



- Use 75% of the training data for model training
- Keep 25% aside for validation (**validation data**)
- Train two models: scikit-learn decision tree classifier and scikit-learn MLP classifier

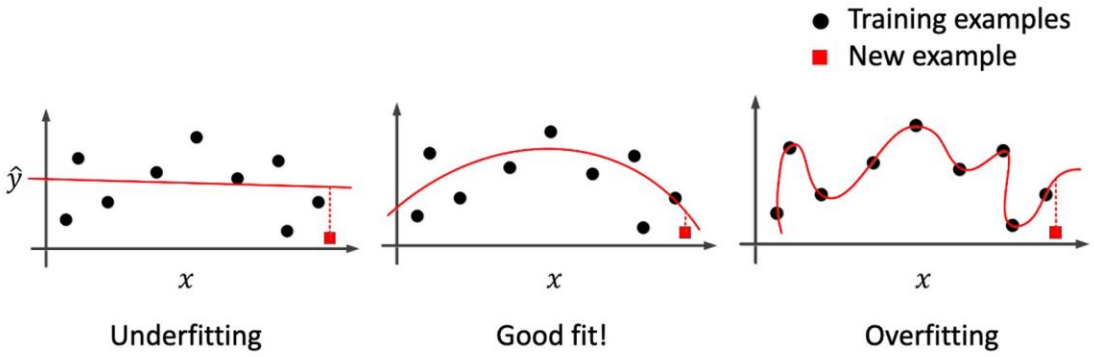
Model	Average Accuracy on Training Data [%]
Decision Tree	100
Neural Network (MLP)	95

A typical Workflow

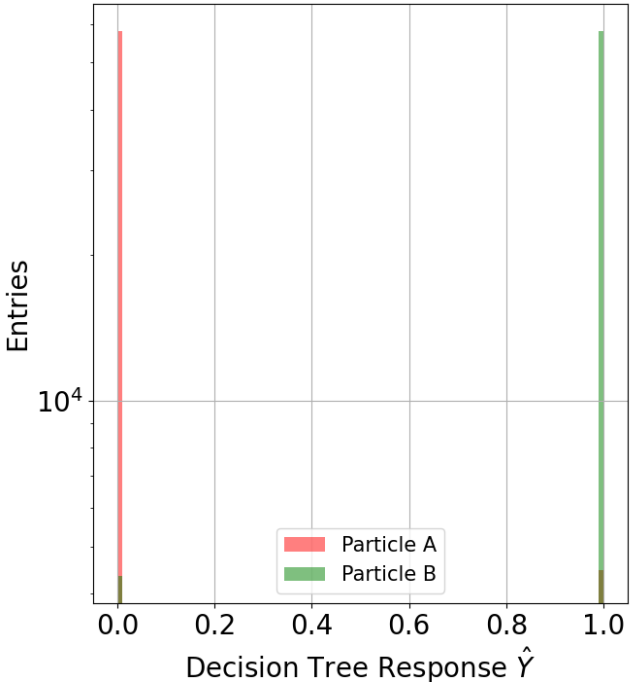


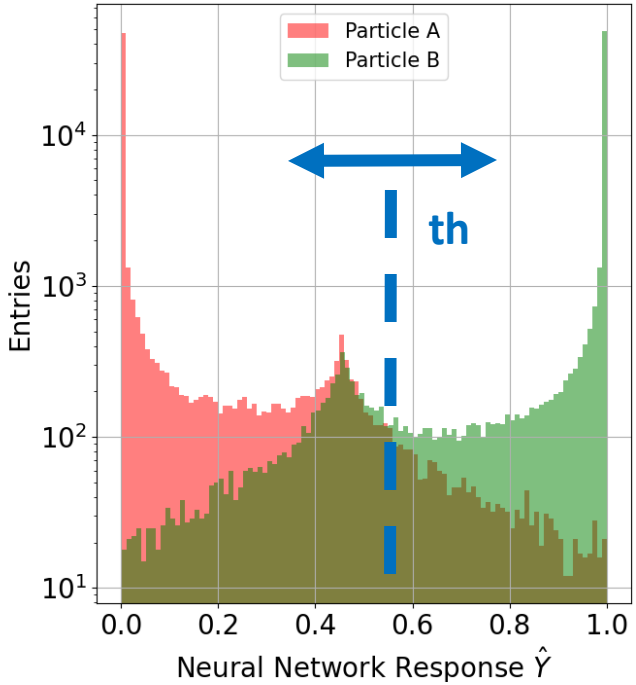
- **VERY IMPORTANT**
- Justify model to yourself, colleagues, …
- Evaluate model performance
- Try to take "black box" character out of model
- Use dedicated data set --> Not "seen" by model during training
- Evaluate our model on the validation data that we kept aside
- Going to spend next slides on analysis and performance evaluation

Plot taken from [Brenda Ngs talk at deep learning for science school 2019]



Model Response on validation Data

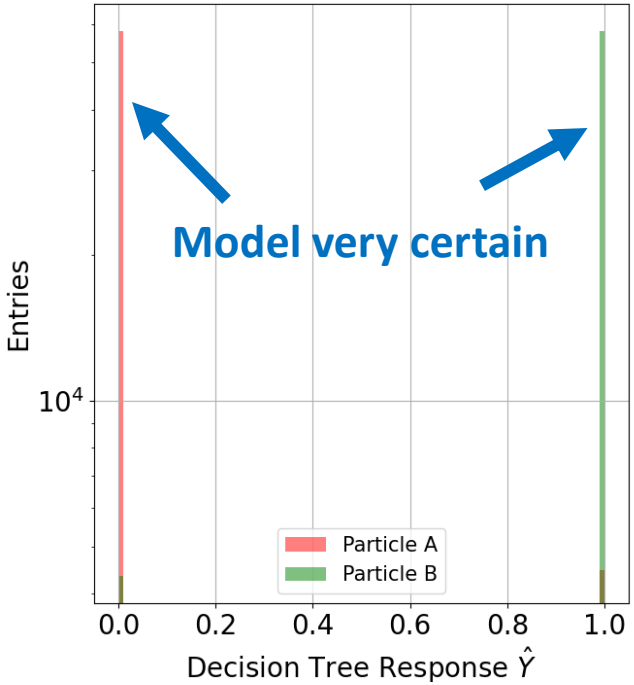


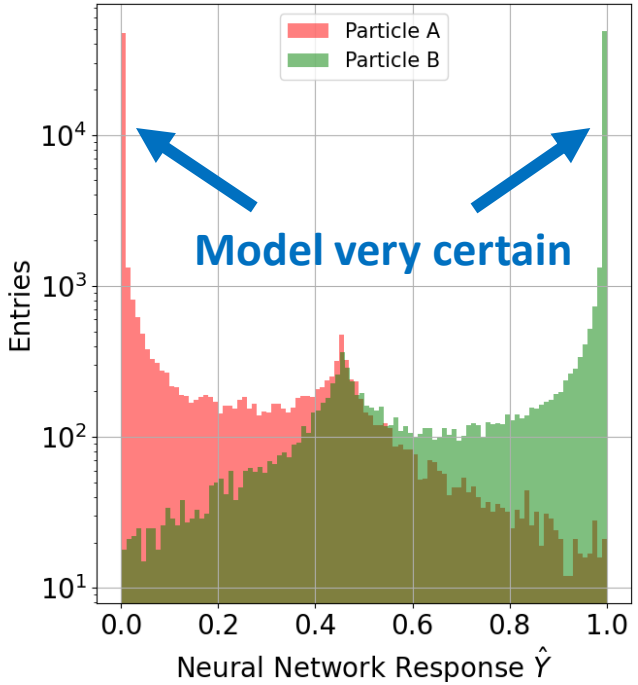


- One of the first plots to check!
- Helps to understand your model
- Decision tree show discrete response
- Translate response to label via threshold th **(works for binary classification problems)**

$$\hat{\ell} = \Theta(\hat{Y}, \text{th}) = \begin{cases} 1, \hat{Y} \geq \text{th}, \\ 0, \text{ else} \end{cases}$$

Model Response on validation Data

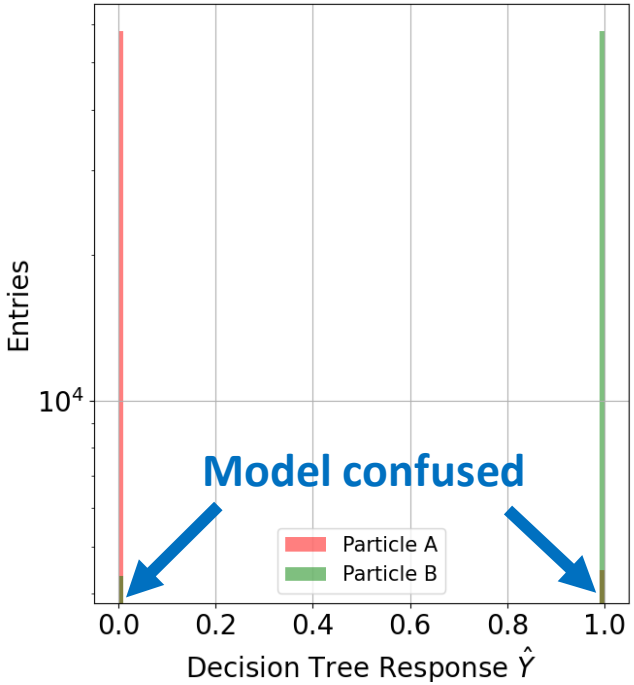


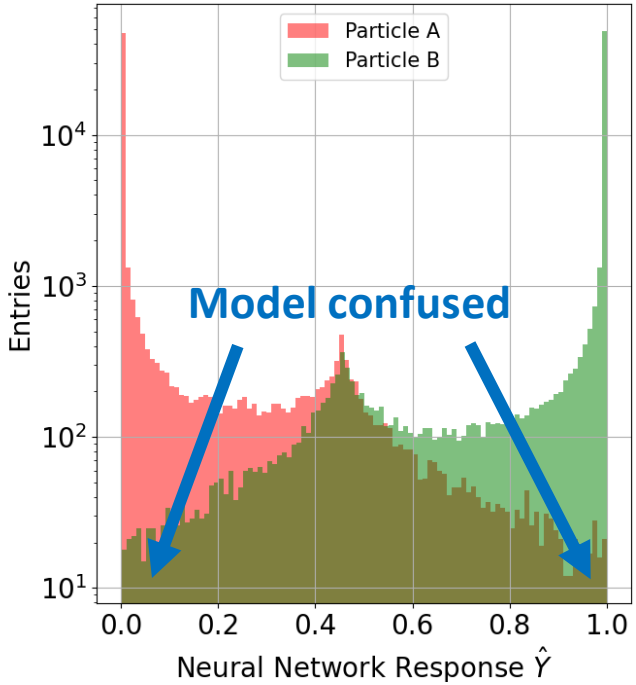


- One of the first plots to check!
- Helps to understand your model
- Decision tree show discrete response
- Translate response to label via threshold th **(works for binary classification problems)**

$$\hat{\ell} = \Theta(\hat{Y}, \text{th}) = \begin{cases} 1, \hat{Y} \geq \text{th}, \\ 0, \text{ else} \end{cases}$$

Model Response on validation Data

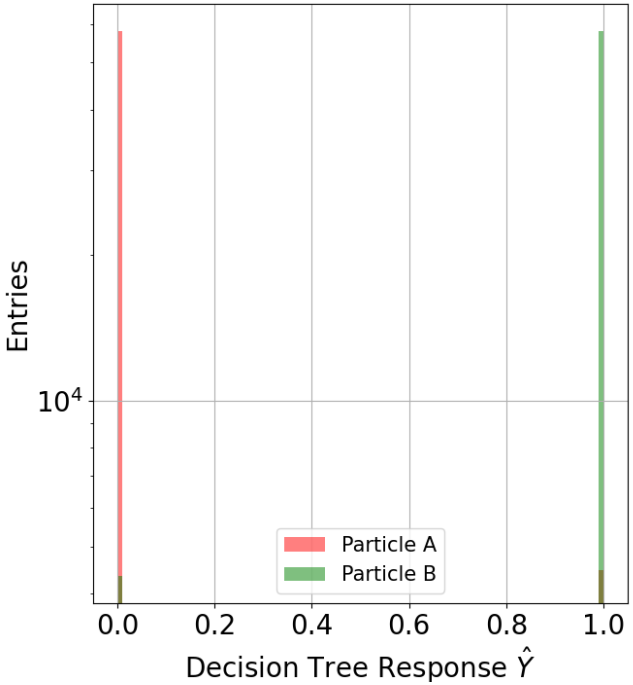


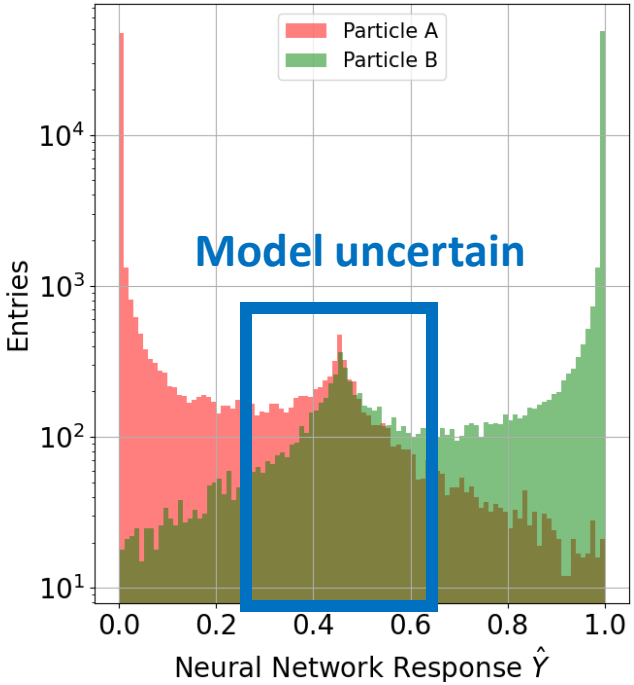


- One of the first plots to check!
- Helps to understand your model
- Decision tree show discrete response
- Translate response to label via threshold th **(works for binary classification problems)**

$$\hat{\ell} = \Theta(\hat{Y}, \text{th}) = \begin{cases} 1, \hat{Y} \geq \text{th}, \\ 0, \text{ else} \end{cases}$$

Model Response on validation Data



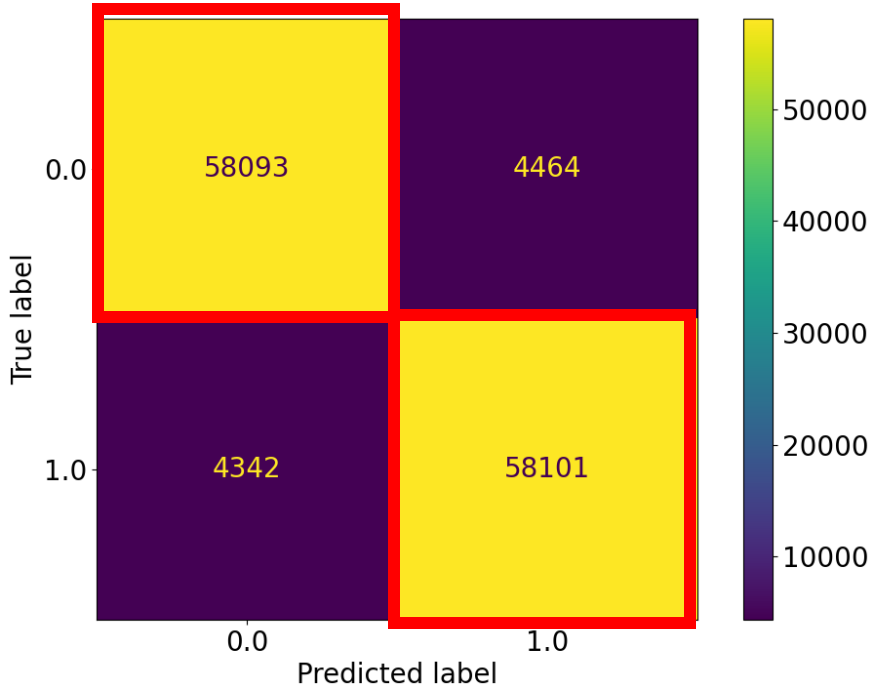


- One of the first plots to check!
- Helps to understand your model
- Decision tree show discrete response
- Translate response to label via threshold th **(works for binary classification problems)**

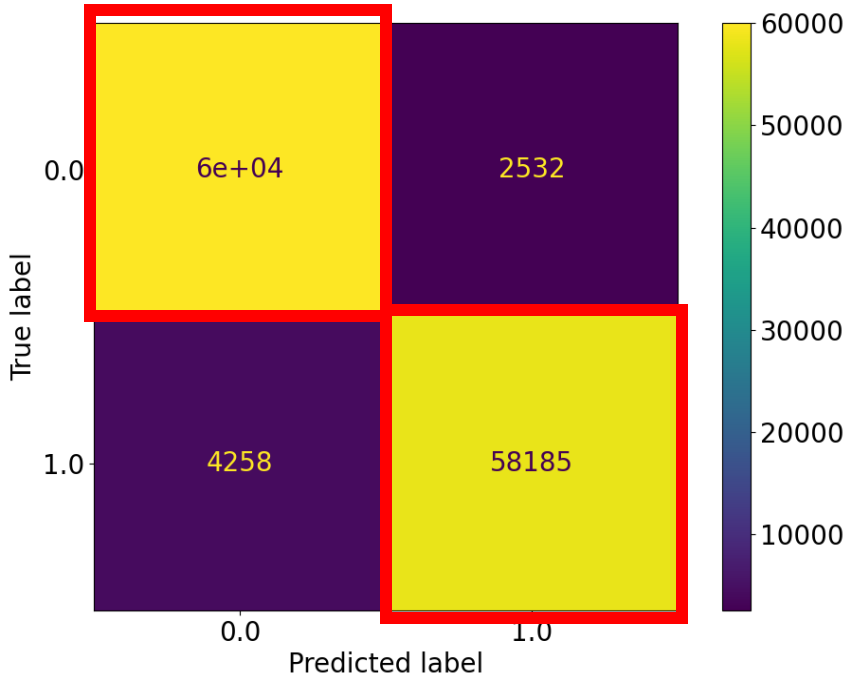
$$\hat{\ell} = \Theta(\hat{Y}, \text{th}) = \begin{cases} 1, \hat{Y} \geq \text{th}, \\ 0, \text{ else} \end{cases}$$

Confusion Matrix

Decision Tree



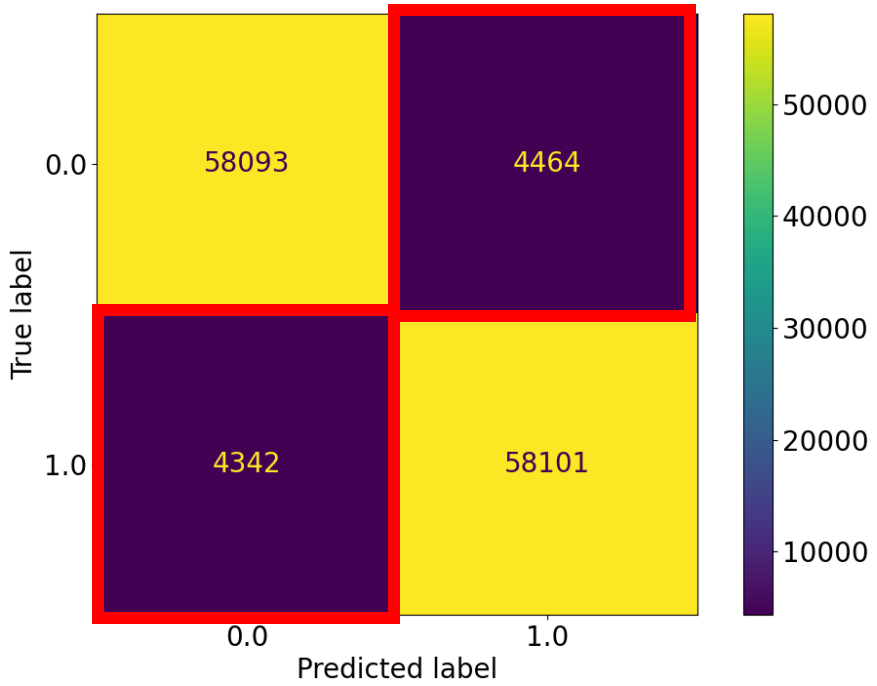
Neural Network



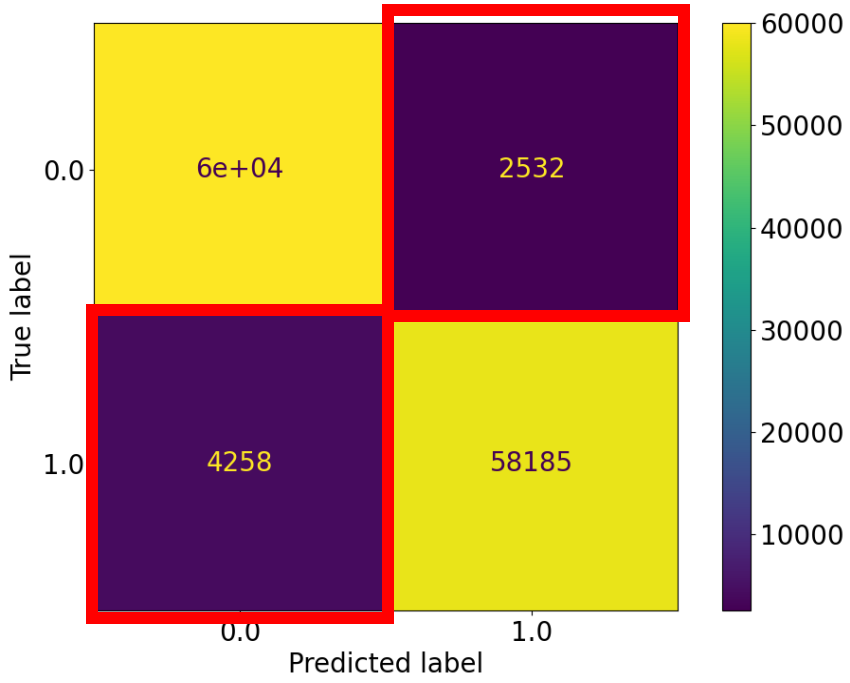
- Compute predicted labels $\hat{\ell} = \Theta(\hat{Y}, \text{th} = 0.5)$
- Count how many times model is right / wrong
- Diagonal matrix elements: $\sum_i [\delta(\ell) \cdot \delta(\hat{\ell})]$, $\sum_i [\delta(1-\ell) \cdot \delta(1-\hat{\ell})]$
- Off-Diagonal matrix elements: $\sum_i [\delta(1-\ell) \cdot \delta(\hat{\ell})]$, $\sum_i [\delta(\ell) \cdot \delta(1-\hat{\ell})]$

Confusion Matrix

Decision Tree



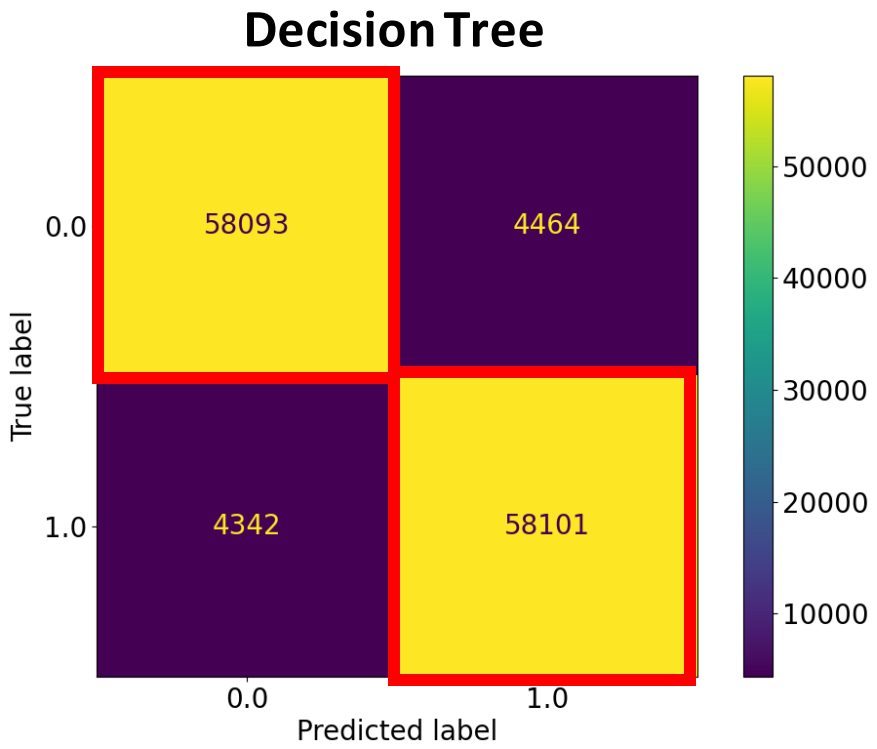
Neural Network

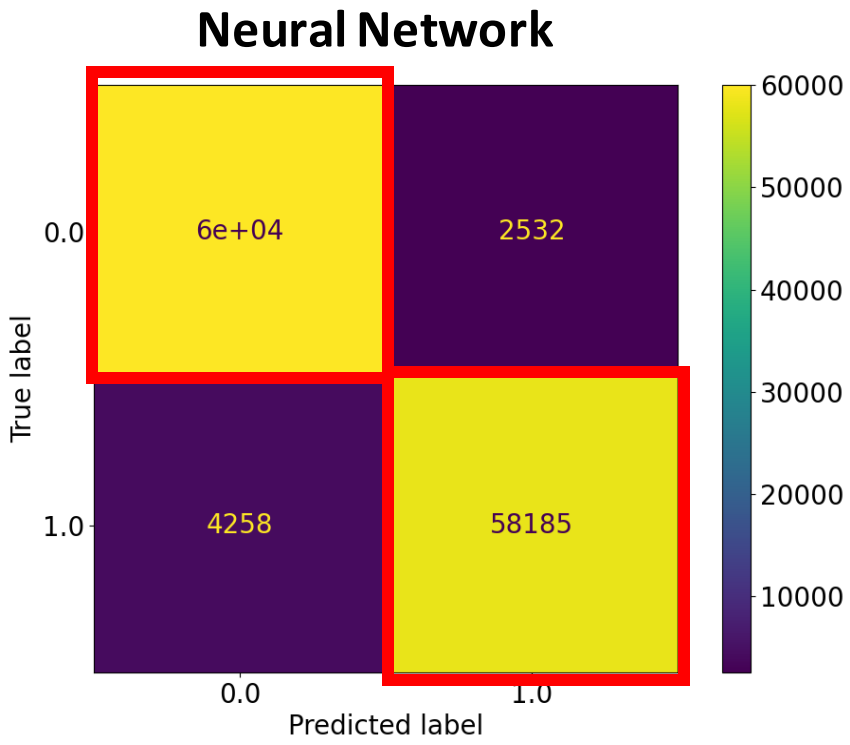


- Compute predicted labels $\hat{\ell} = \Theta(\hat{Y}, \text{th} = 0.5)$
- Count how many times model is right / wrong
- Diagonal matrix elements: $\sum_i [\delta(\ell) \cdot \delta(\hat{\ell})]$, $\sum_i [\delta(1-\ell) \cdot \delta(1-\hat{\ell})]$
- Off-Diagonal matrix elements: $\sum_i [\delta(1-\ell) \cdot \delta(\hat{\ell})]$, $\sum_i [\delta(\ell) \cdot \delta(1-\hat{\ell})]$

Ideally 0

Confusion Matrix and Accuracy

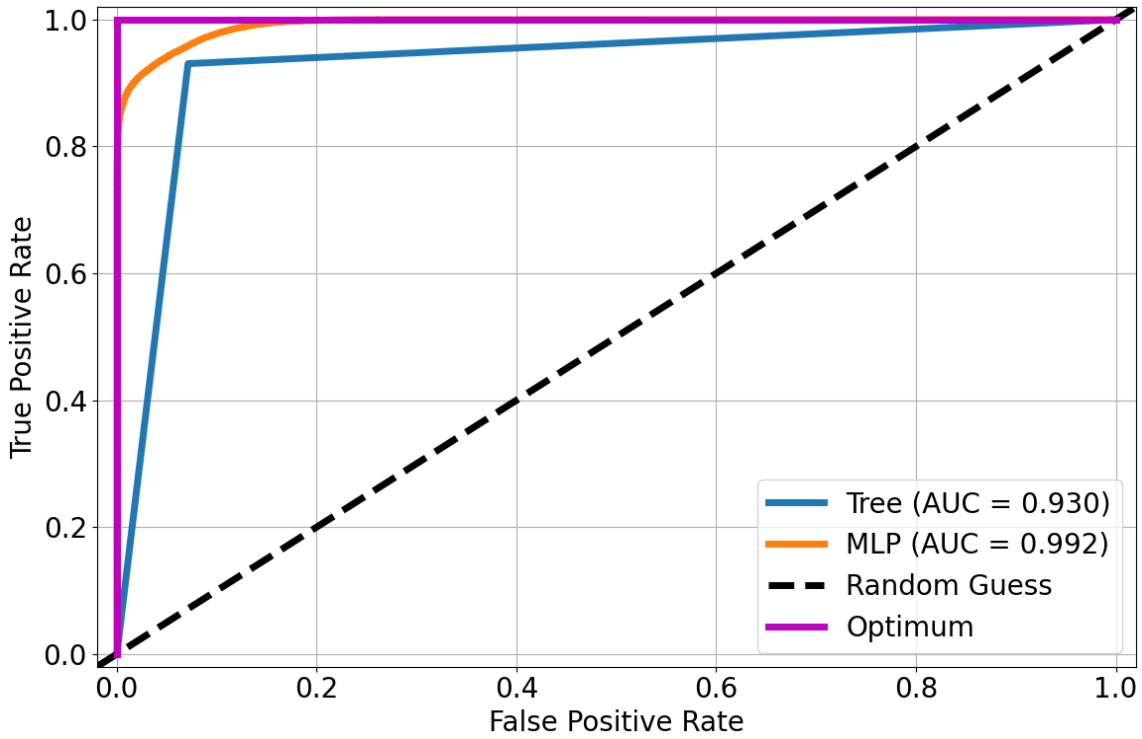




- Given confusion matrix C
- Accuracy $= tr(C) / \sum_{ij} C_{ij}$

Model	Average Accuracy on Validation Data [%]
Decision Tree	93
Neural Network (MLP)	95

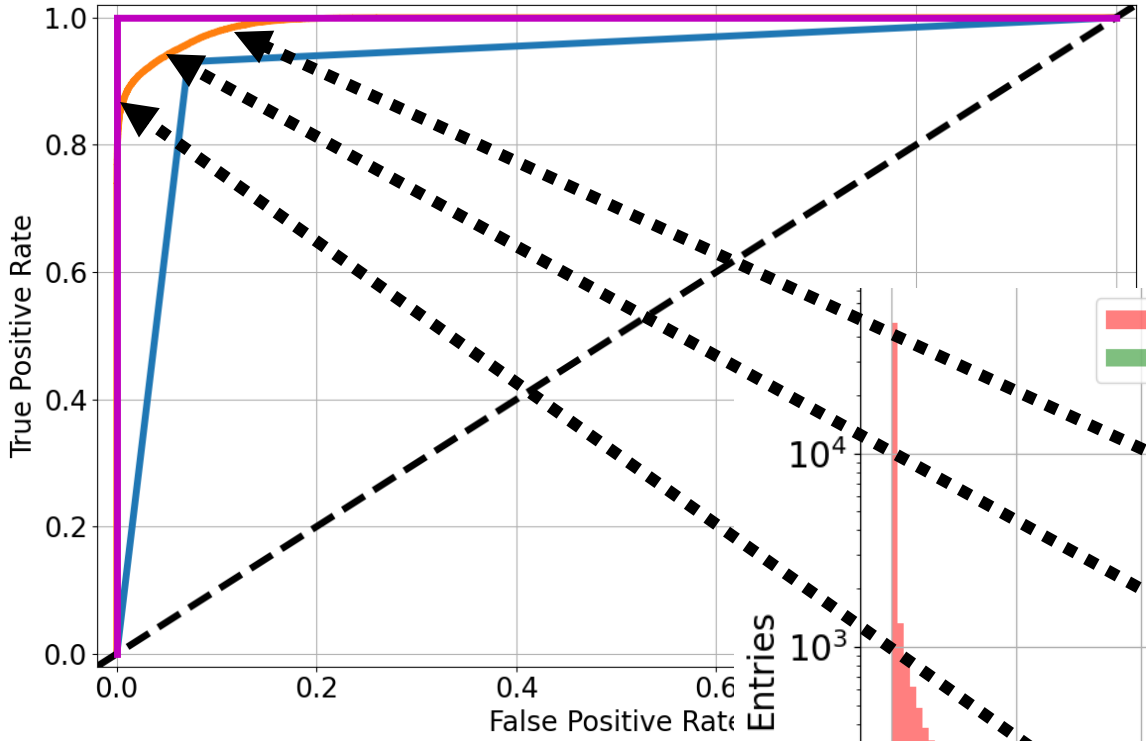
Receiver Operating Characteristic (ROC) Curve for identifying Particle B



- **True Positive Rate:** How often does classifier **correctly identify particle B** ?
- **False Positive Rate:** How often does classifier **falsely identify particle A as B** ?
- Each point on ROC corresponds to one threshold value
- Could also look at ROC for identifying particle A

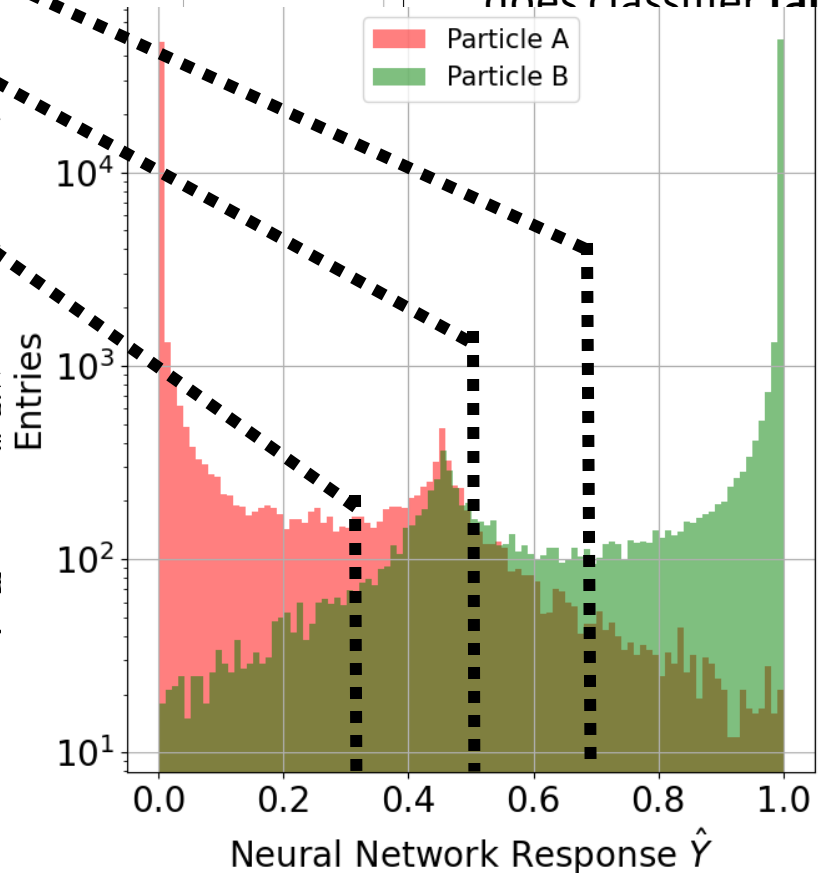
- AUC = Area Under Curve --> Ideally 1.0
- Slope of ROC is ultimately defined by response distribution

Receiver Operating Characteristic (ROC) Curve for identifying Particle B



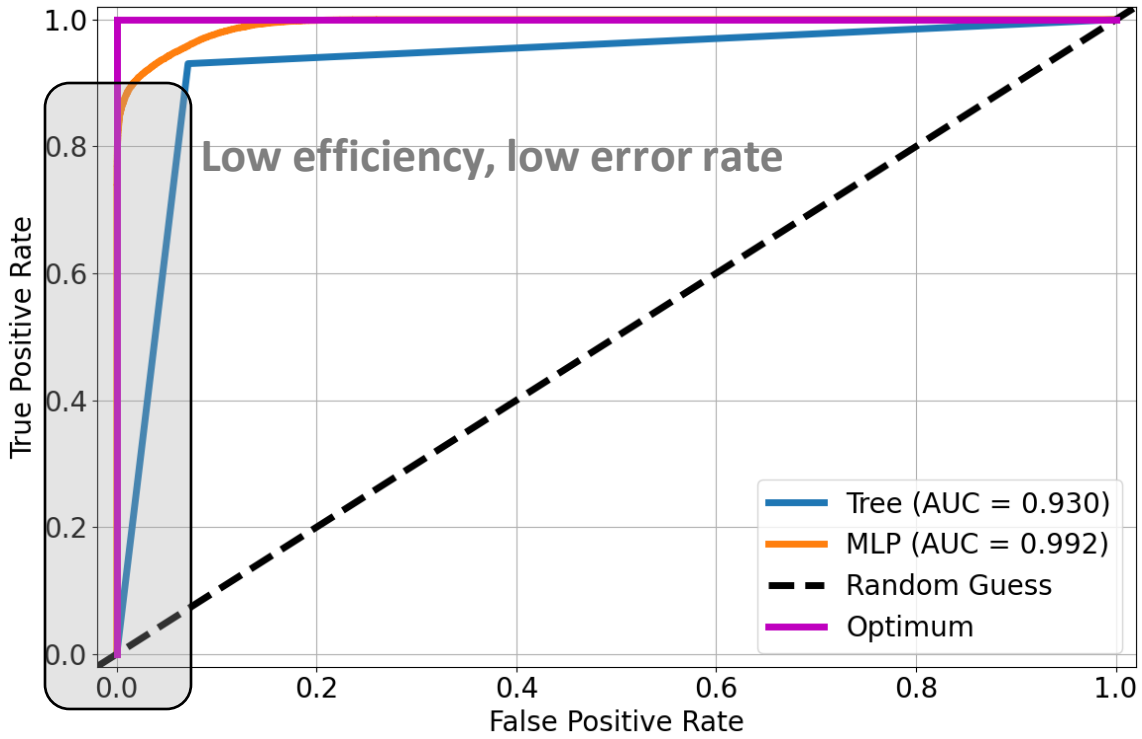
- **True Positive Rate:** How often does classifier **correctly identify particle B** ?
- **False Positive Rate:** How often does classifier **falsely identify**

C corresponds to
ue
t ROC for
le A



- AUC = Area Under Curve --> Id
- Slope of ROC is ultimately defi

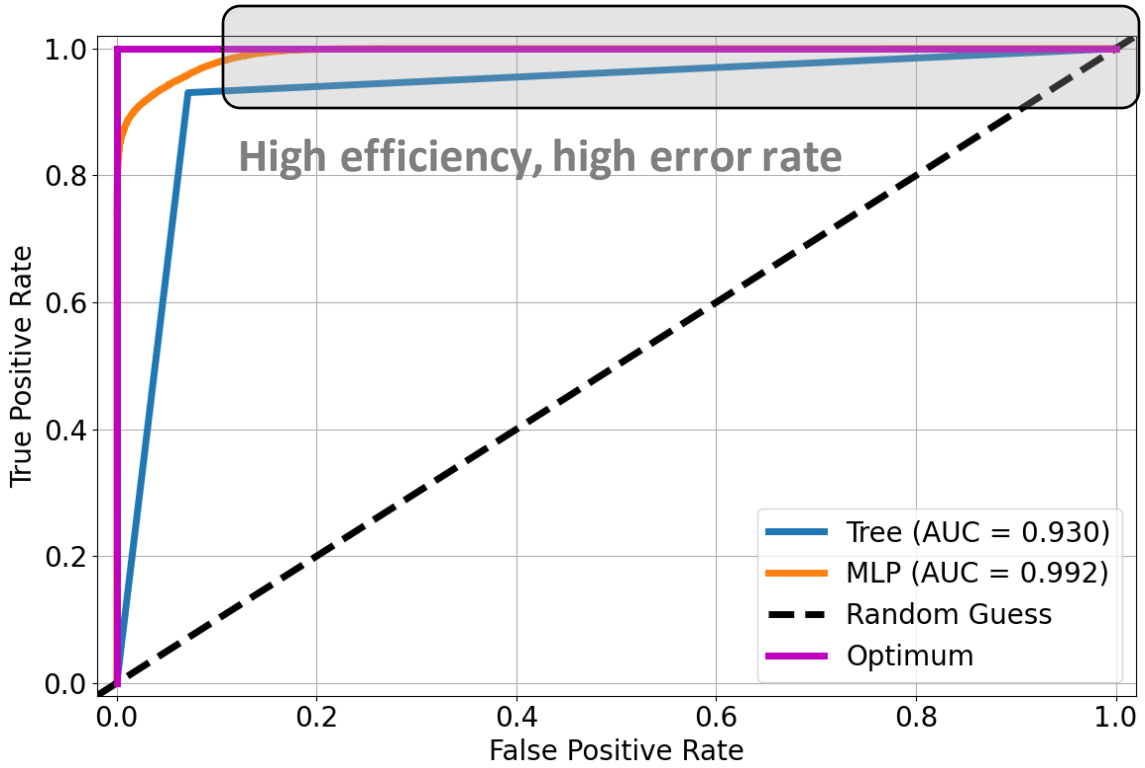
Receiver Operating Characteristic (ROC) Curve for identifying Particle B



- **True Positive Rate:** How often does classifier **correctly identify particle B** ?
- **False Positive Rate:** How often does classifier **falsely identify particle A as B** ?
- Each point on ROC corresponds to one threshold value
- Could also look at ROC for identifying particle A

- AUC = Area Under Curve --> Ideally 1.0
- Slope of ROC is ultimately defined by response distribution

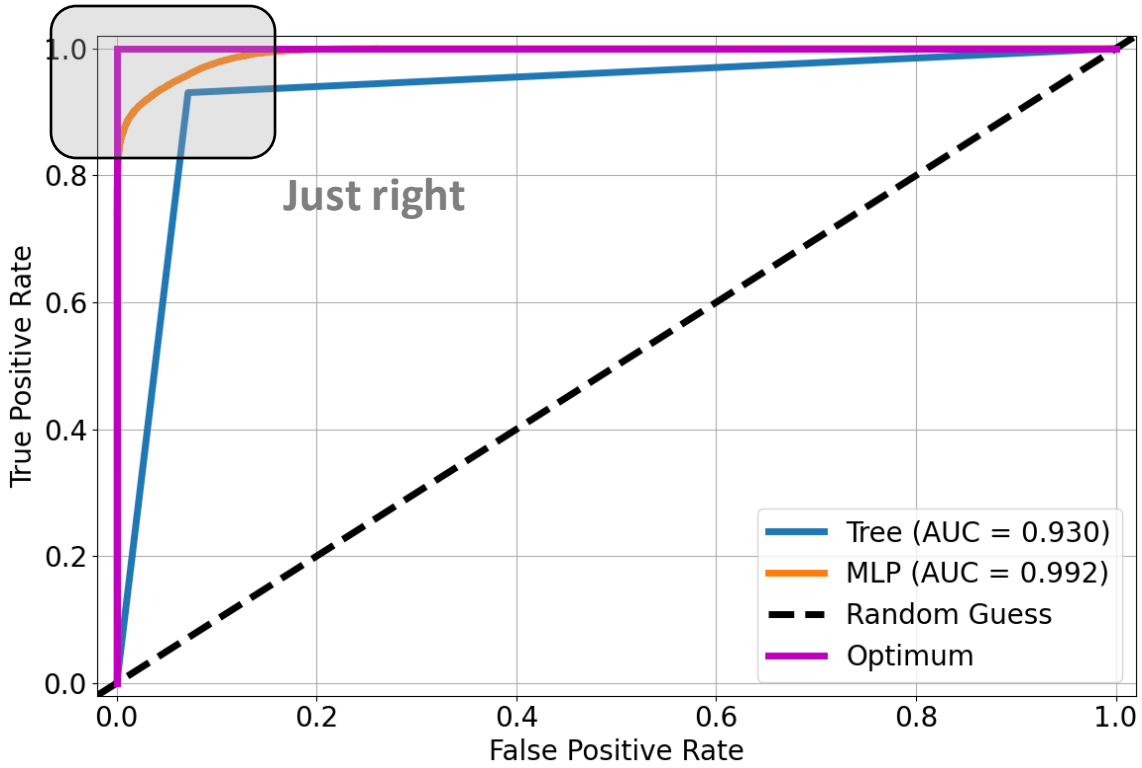
Receiver Operating Characteristic (ROC) Curve for identifying Particle B



- **True Positive Rate:** How often does classifier **correctly identify particle B** ?
- **False Positive Rate:** How often does classifier **falsely identify particle A as B** ?
- Each point on ROC corresponds to one threshold value
- Could also look at ROC for identifying particle A

- AUC = Area Under Curve --> Ideally 1.0
- Slope of ROC is ultimately defined by response distribution

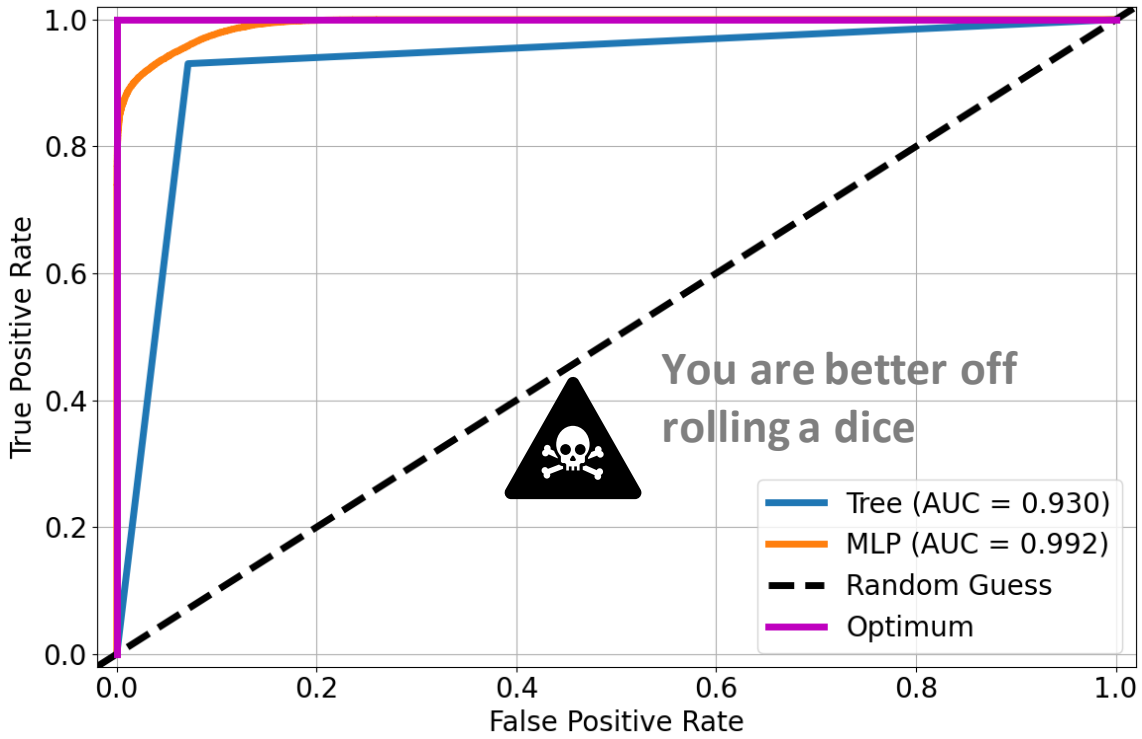
Receiver Operating Characteristic (ROC) Curve for identifying Particle B



- **True Positive Rate:** How often does classifier **correctly identify particle B** ?
- **False Positive Rate:** How often does classifier **falsely identify particle A as B** ?
- Each point on ROC corresponds to one threshold value
- Could also look at ROC for identifying particle A

- AUC = Area Under Curve --> Ideally 1.0
- Slope of ROC is ultimately defined by response distribution

Receiver Operating Characteristic (ROC) Curve for identifying Particle B



- **True Positive Rate:** How often does classifier **correctly identify particle B** ?
- **False Positive Rate:** How often does classifier **falsely identify particle A as B** ?
- Each point on ROC corresponds to one threshold value
- Could also look at ROC for identifying particle A

- AUC = Area Under Curve --> Ideally 1.0
- Slope of ROC is ultimately defined by response distribution

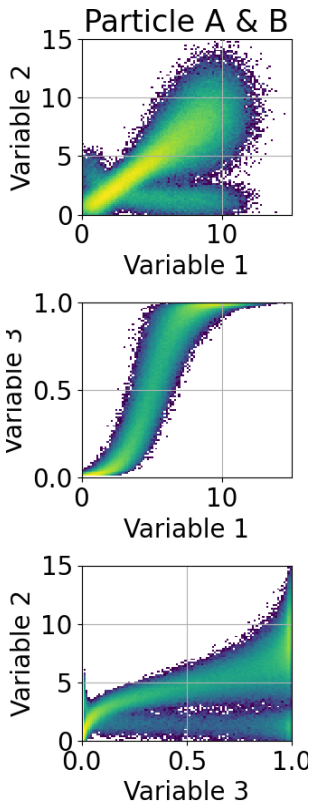
Performance Summary

Model	Accuracy [%]	AUC	True Positive Rate [%]	False Positive Rate [%]
Decision Tree	93	0.93	93	7
Neural Newtork	95	0.99	94	5

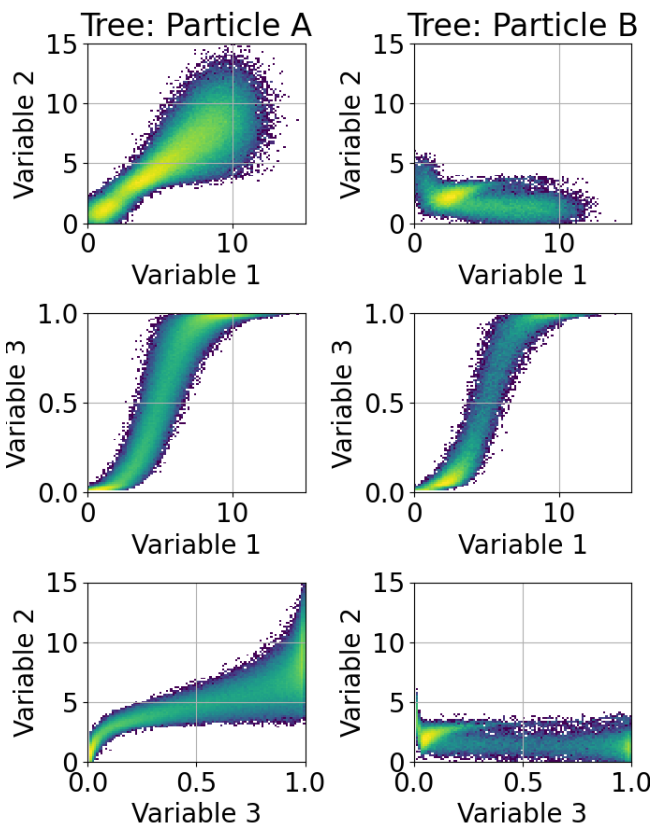
- All metrics determined from validation data set
- Use threshold: th = 0.5
- Models perform similar
- Next step: Apply models on experimental data set

Apply Model on experimental Data Set

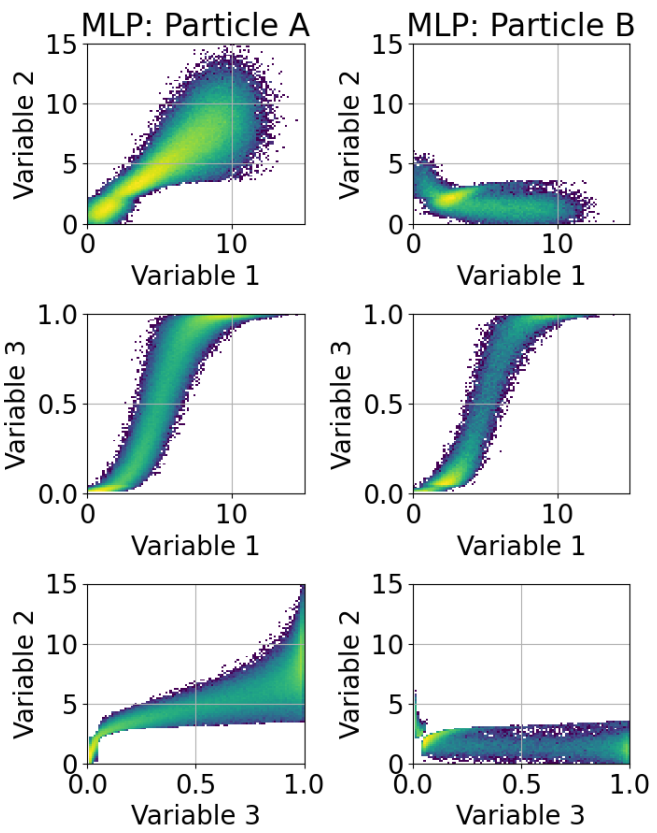
Input



Decision Tree



Neural Network



What is left to do?

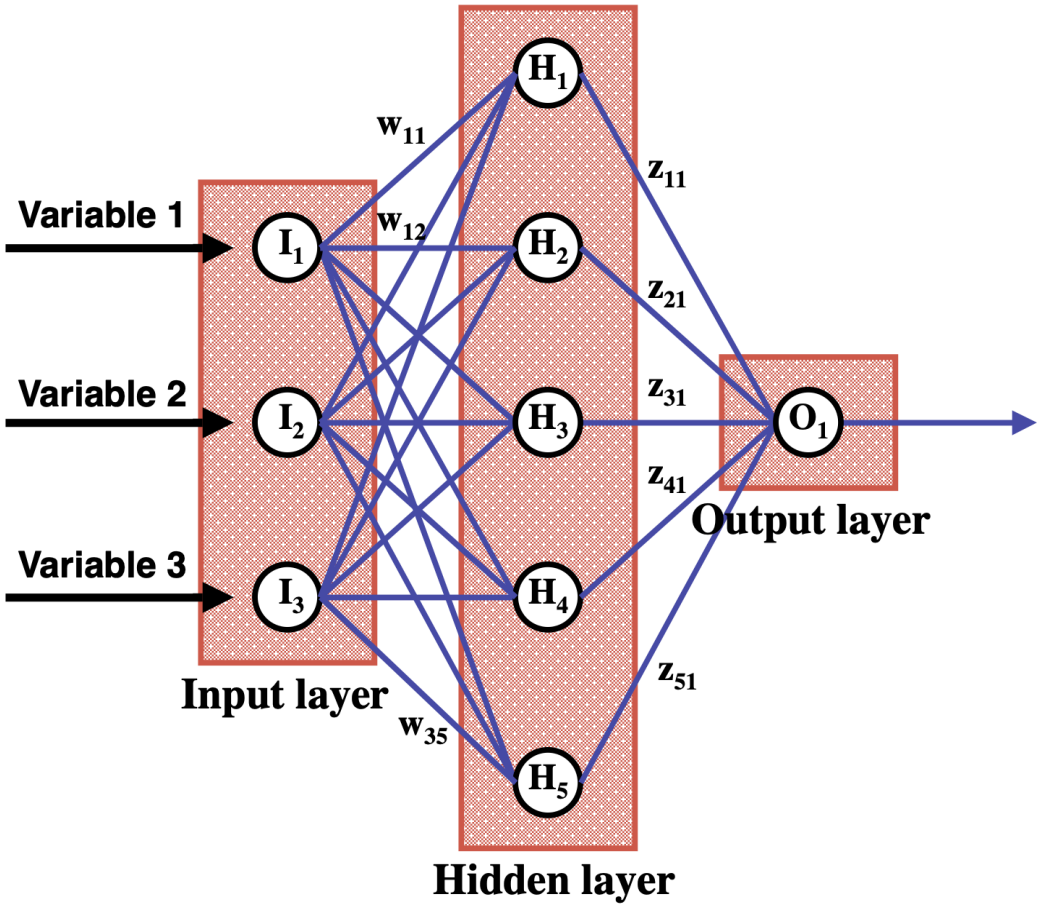
- Define metric to judge model performance on "real" data
  - Use missing mass spectra to determine expected abundance
  - Use curated experimental data (with known abundances)
  - Compare to "conservative" analysis
- Try different settings for each model and check for improved performance --> Hyper Parameter Optimization (HPO)
  - Internal parameters θ are found by training
  - Model complexity defined by hyper parameters (e.g. size, number of training steps,..)
- Estimate model uncertainty <--> How reliable are model predictions ?
  - Is model complex enough to solve given task ?
  - Is training data too different from "real" data (e.g. detector resolution, acceptance,..)
  - When and why does model fail ?
  - ….

Neural Networks

Stone-Weierstrass-Theorem (1990): "[...] there are no nemesis functions that can not be modeled by neural networks"

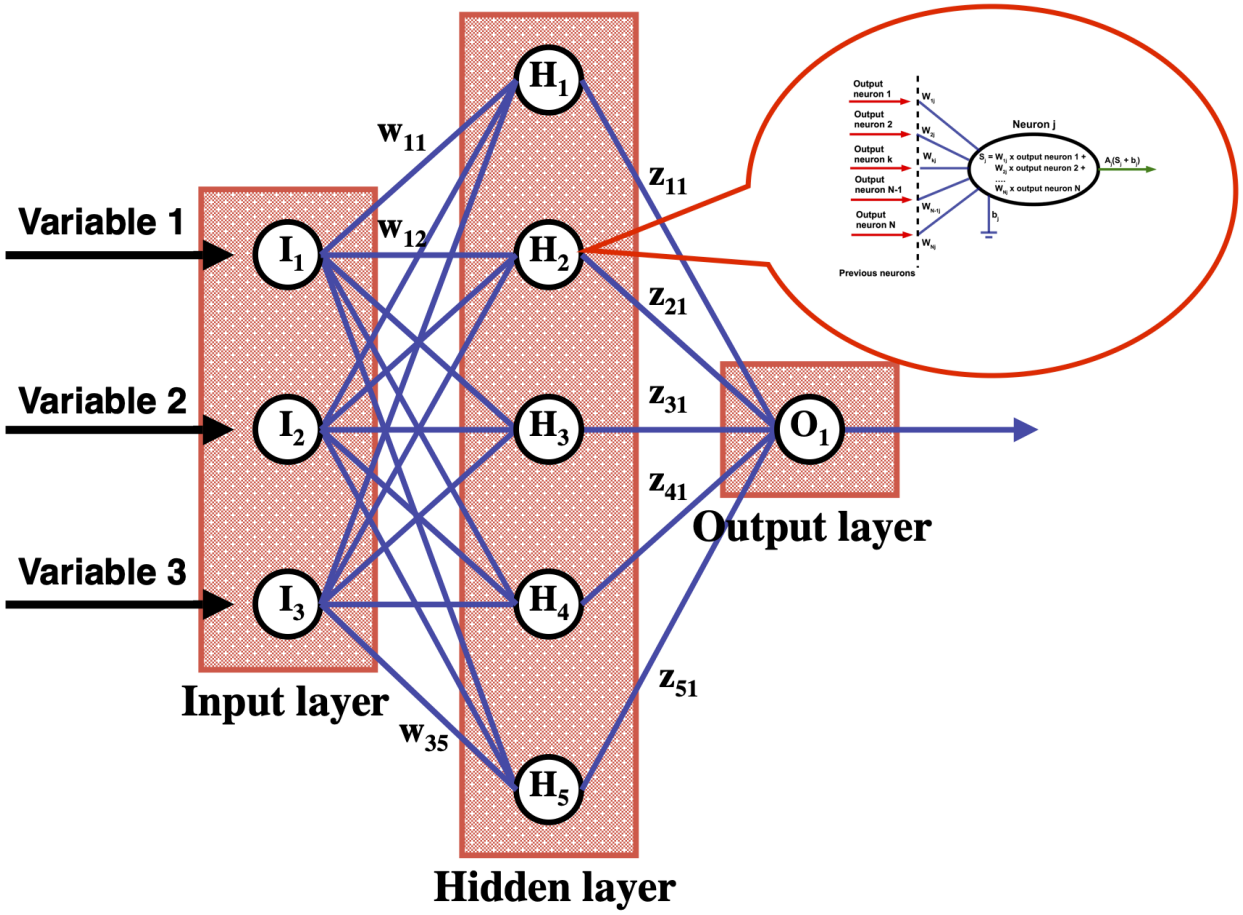
- Multilayer Perceptron (Discuss other network types later)
- Backpropagation
- Gradient Descent
- Network Optimizers
- Tensors
- Parameter Initialization
- Early stopping

Multilayer Perceptron (MLP)



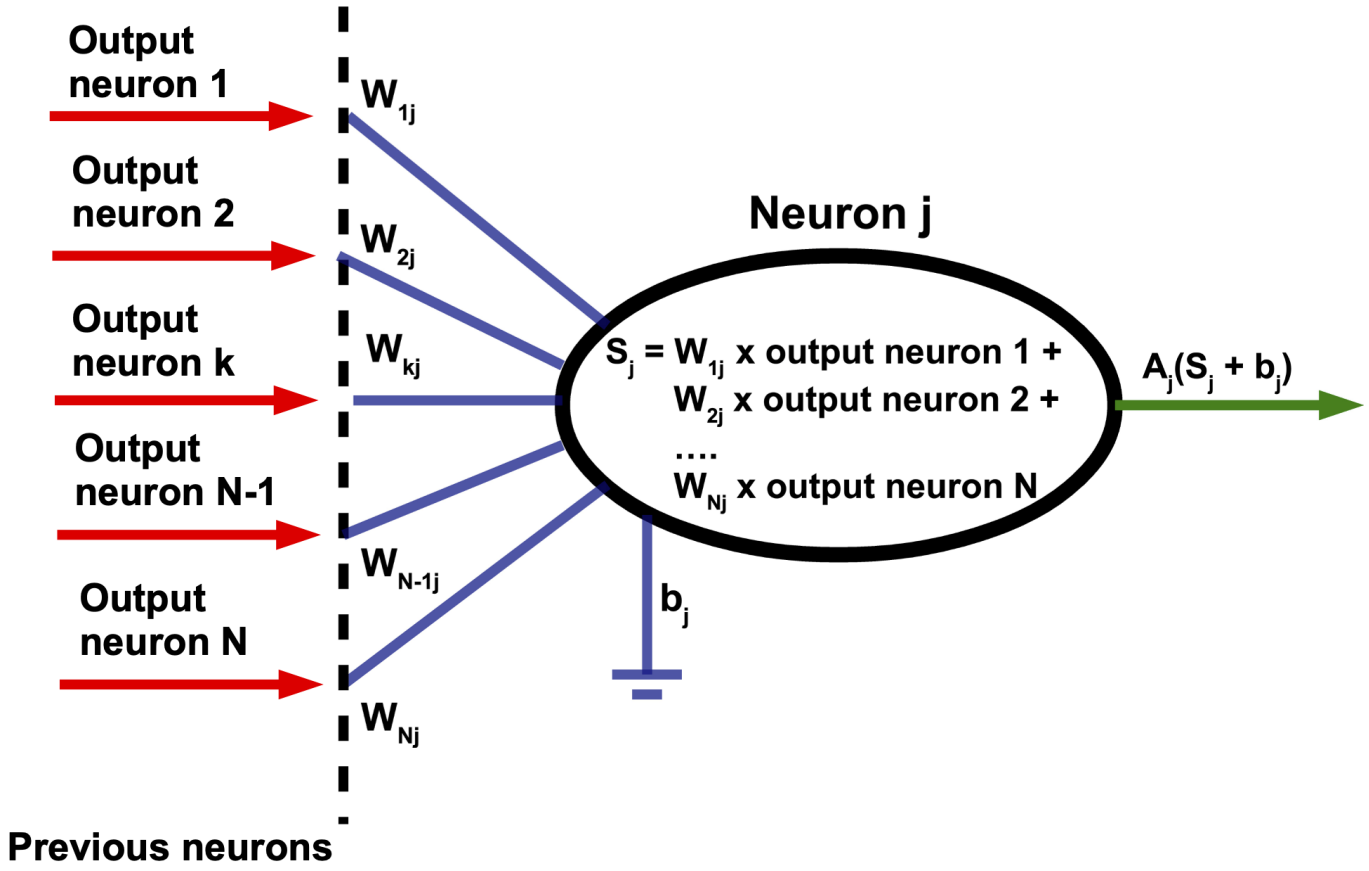
- **Dense neural network**
- **Network Architecture:** Hidden layers + Neurons
- **Learnable Parameters:** Weights and Biases

Multilayer Perceptron (MLP)



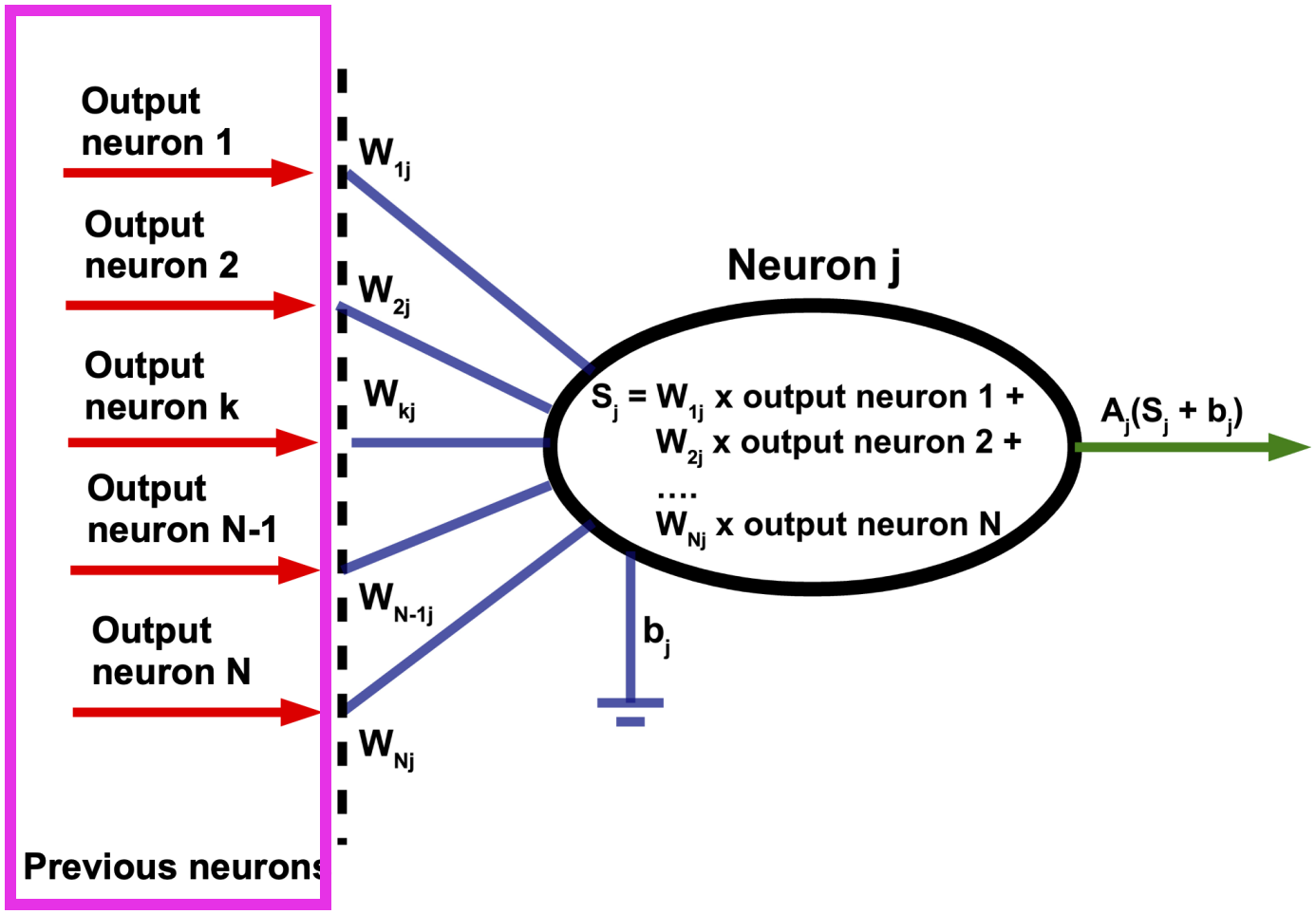
- **Dense neural network**
- **Network Architecture:** Hidden layers + Neurons
- **Learnable Parameters:** Weights and Biases

A single Neuron



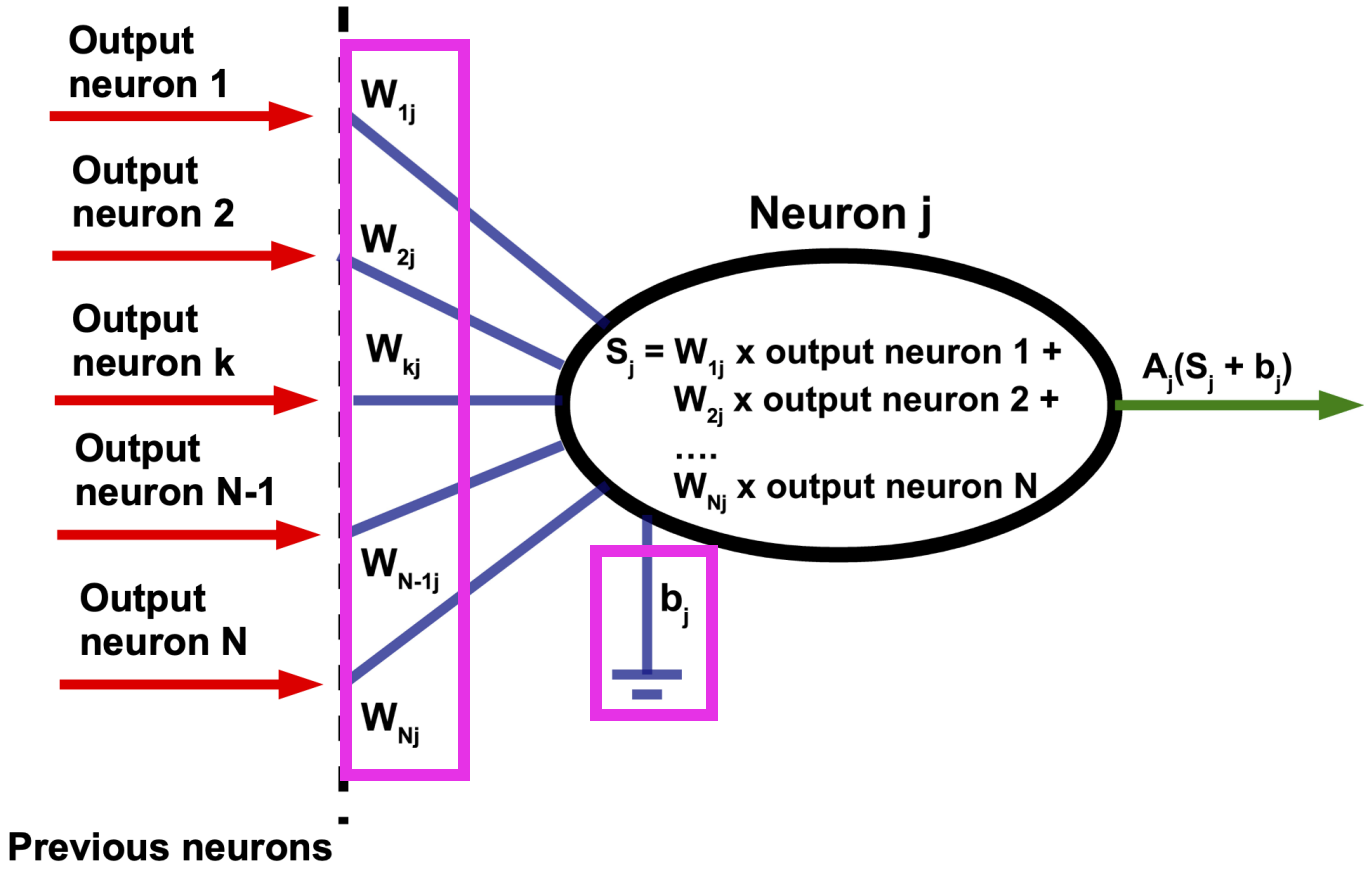
A single Neuron

Information from previous Neurons

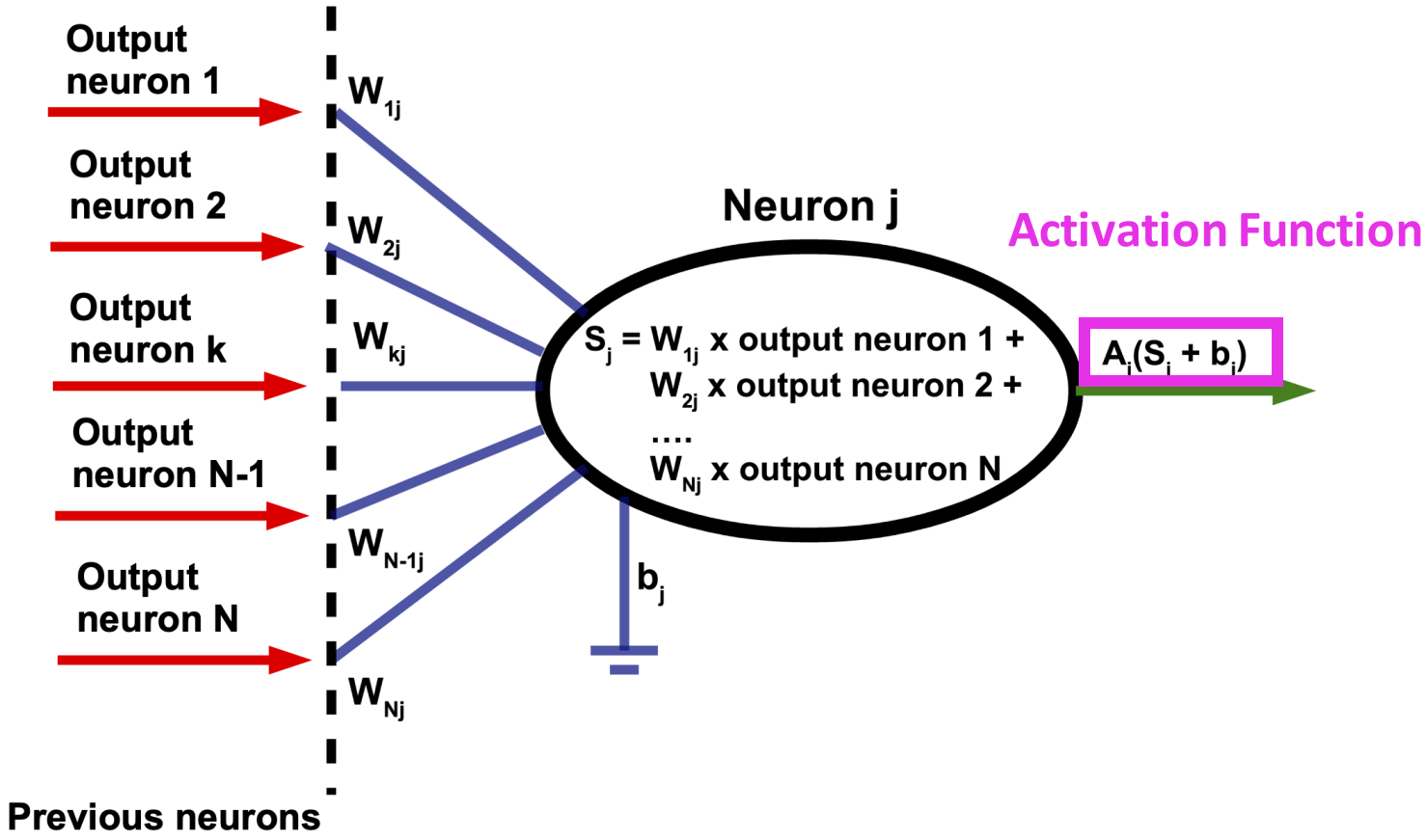


A single Neuron

Weights and Biases --> Adjusted during training



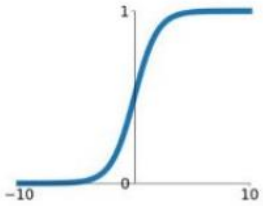
A single Neuron



Activation Functions

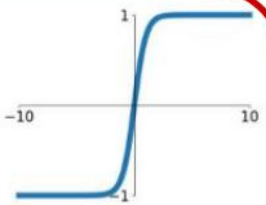
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



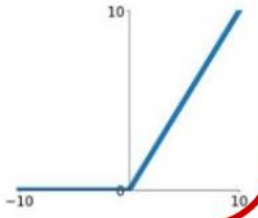
tanh

$$\tanh(x)$$



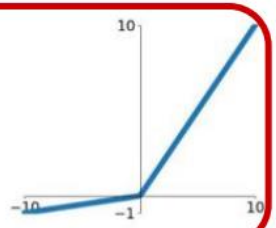
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

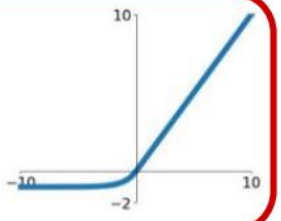


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



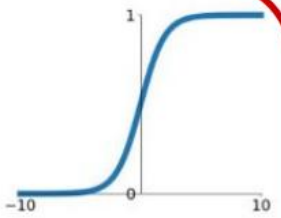
Most commonly used in modern networks as hidden layer activations

Plots taken from [Mustafa Mustafas talk at deep learning for science school 2019]

Activation Functions

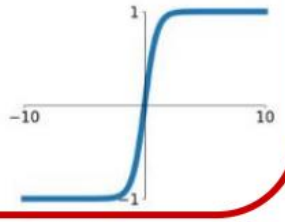
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



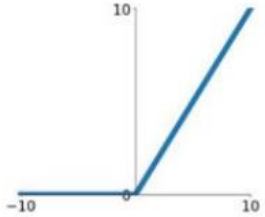
tanh

$$\tanh(x)$$



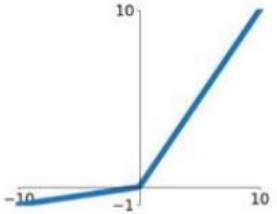
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

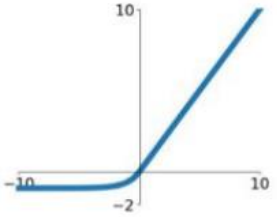


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

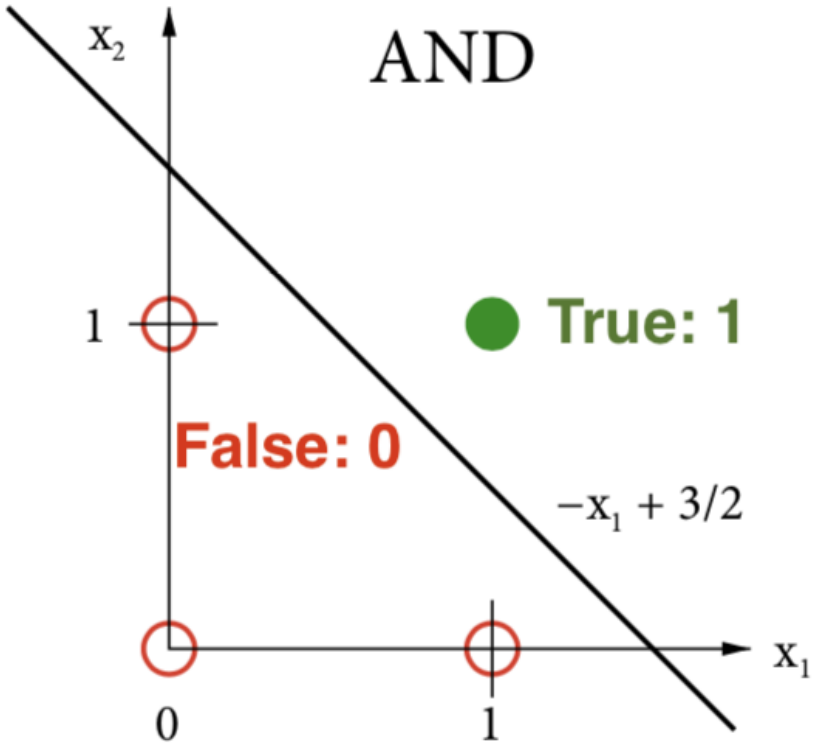
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

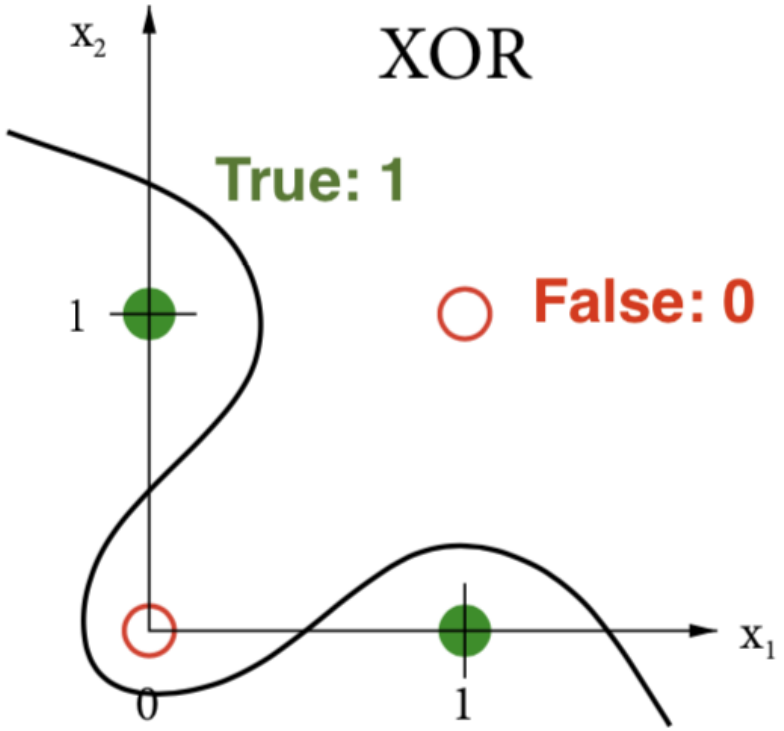


Often used for output layers

Plots taken from [Mustafa Mustafas talk at deep learning for science school 2019]

The XOR Problem

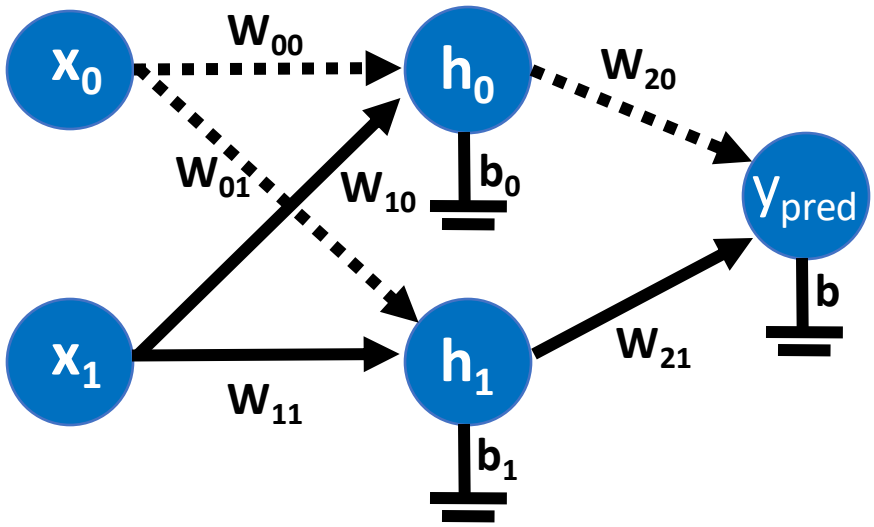




- **Left:** Easily solvable by most analytical functions
- **Right:** Not trivially solvable --> Use MLP

Input x	Expected Output y
(0.0,0.0)	0.0
(0.0,1.0)	1.0
(1.0,0.0)	1.0
(1.0,1.0)	0.0

Setting up the Neural Network



Output Layer:

$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

Output Activation: (Logistic Function)

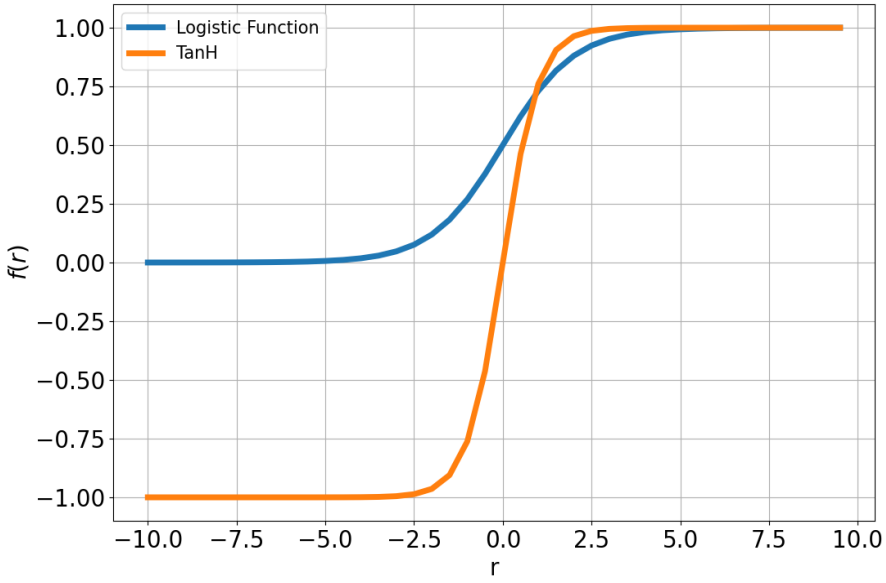
$$f_L(r) = \frac{1}{1+\exp(-r)}$$

Hidden Layer:

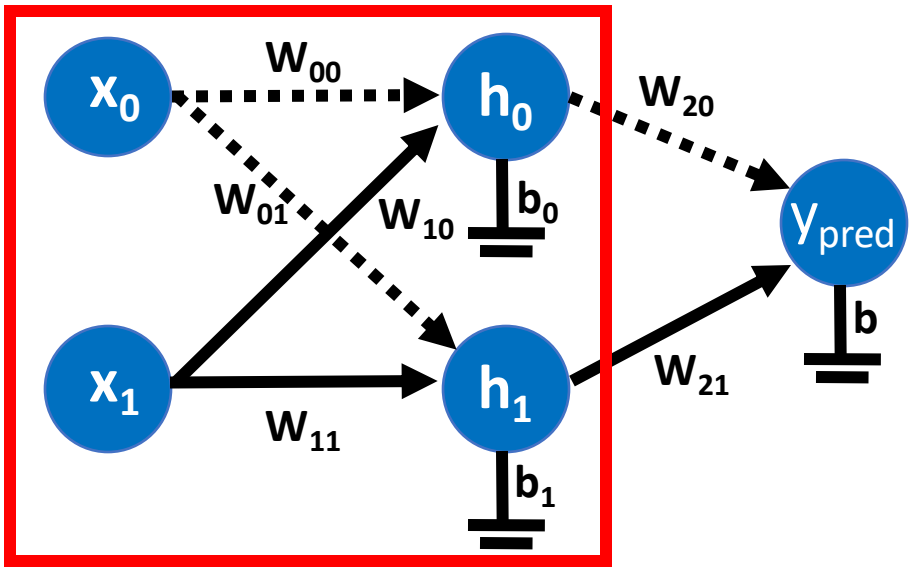
$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$
$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

Hidden Activation: (TanH)

$$f_h(r) = \tanh(r)$$



Forward Pass



Output Layer:

$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

Output Activation: (Logistic Function)

$$f_L(r) = \frac{1}{1+\exp(-r)}$$

Hidden Layer:

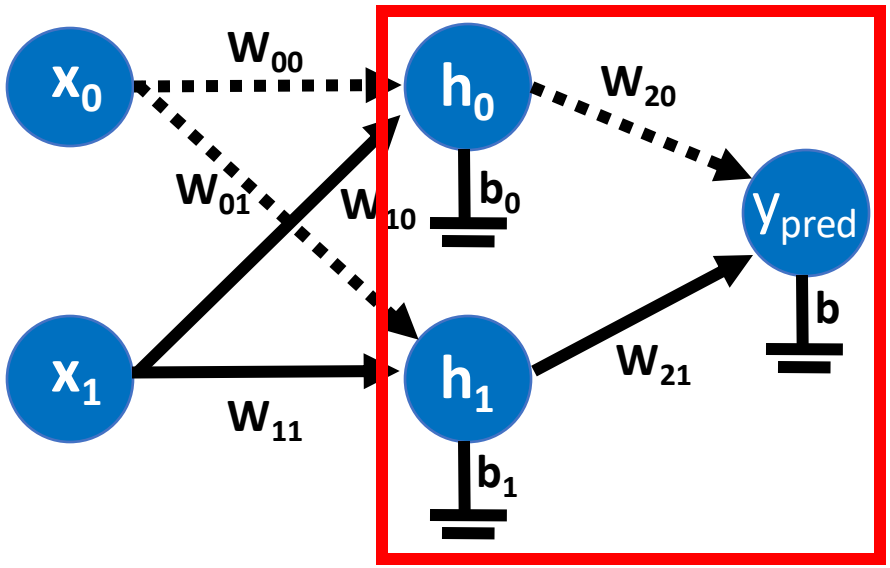
$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$
$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

Hidden Activation: (TanH)

$$f_h(r) = \tanh(r)$$

- Pass data through network
- Determine network response
- Data flow (in this cartoon) is from left to right

Forward Pass



Output Layer:

$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

Output Activation: (Logistic Function)

$$f_L(r) = \frac{1}{1+\exp(-r)}$$

Hidden Layer:

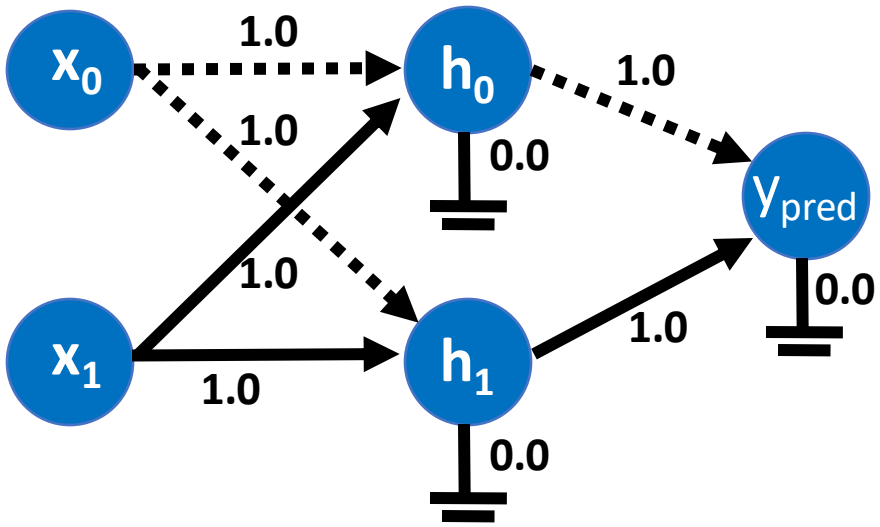
$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$
$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

Hidden Activation: (TanH)

$$f_h(r) = \tanh(r)$$

- Pass data through network
- Determine network response
- Data flow (in this cartoon) is from left to right

Finding Weights and Biases



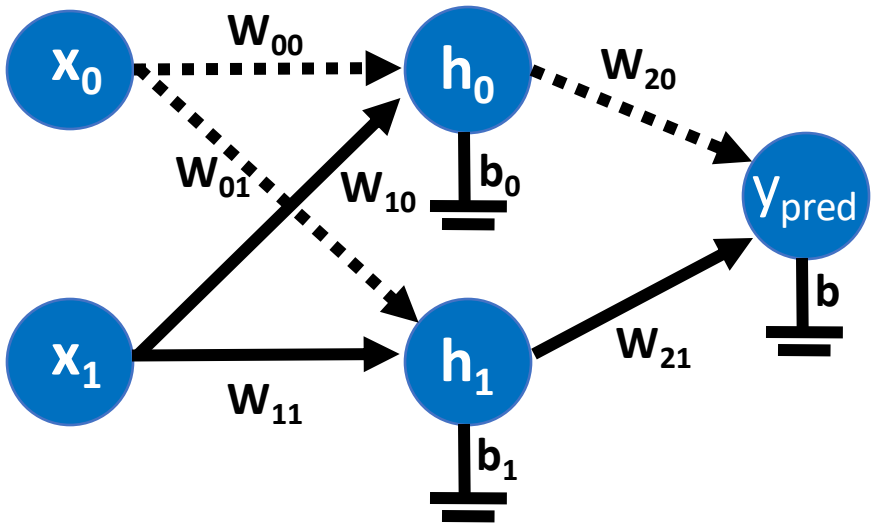
- Start with a guess
- Set all weights to 1.0 and all biases to 0.0
- Maybe we are lucky
- Use loss to measure deviation from expected output
- **Are there weights and biases that minimize the loss?**

$$\text{loss} = (\text{expected output} - \text{predicted output})^2$$

Input x	Expected Output	Predicted Output	Loss
(0.0,0.0)	0.0	0.68	0.1
(0.0,1.0)	1.0	0.5	0.25
(1.0,0.0)	1.0	0.68	0.1
(1.0,1.0)	0.0	0.72	0.52

Σ Loss = 0.98

Minimizing the Loss



Output Layer:

$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

Output Activation: (Logistic Function)

$$f_L(r) = \frac{1}{1+\exp(-r)}$$

Hidden Layer:

$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$
$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

Hidden Activation: (TanH)

$$f_h(r) = \tanh(r)$$

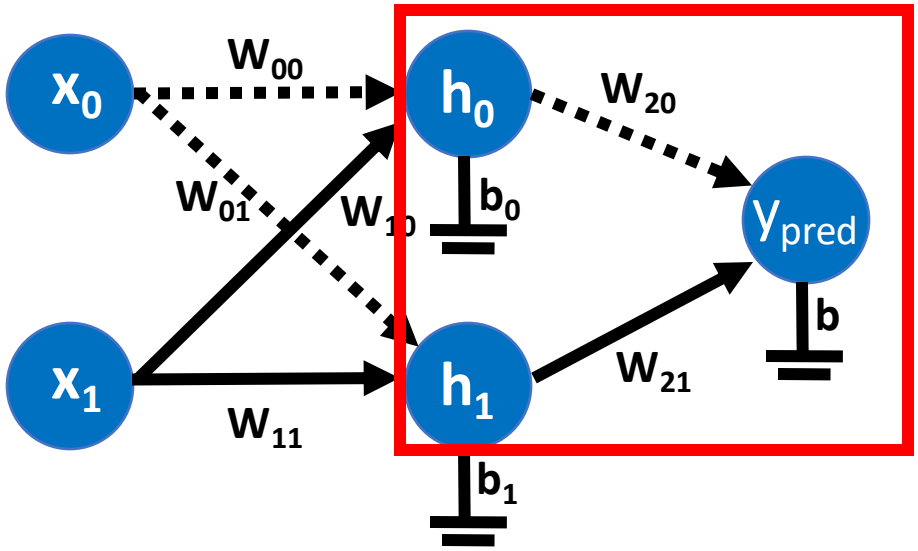
Loss:

$$\text{loss} = \frac{1}{4}\sum_{i=1}^{4}(y_i - y_{pred,i})^2$$

Try to find W_{jk} and b_k with:

$$\frac{\partial \text{loss}}{\partial W_{jk}} \approx 0, \quad \frac{\partial \text{loss}}{\partial b_k} \approx 0$$

Computing Gradients (1)



$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

$$f_L(r) = \frac{1}{1+\exp(-r)}$$

$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$

$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

$$f_h(r) = \tanh(r)$$

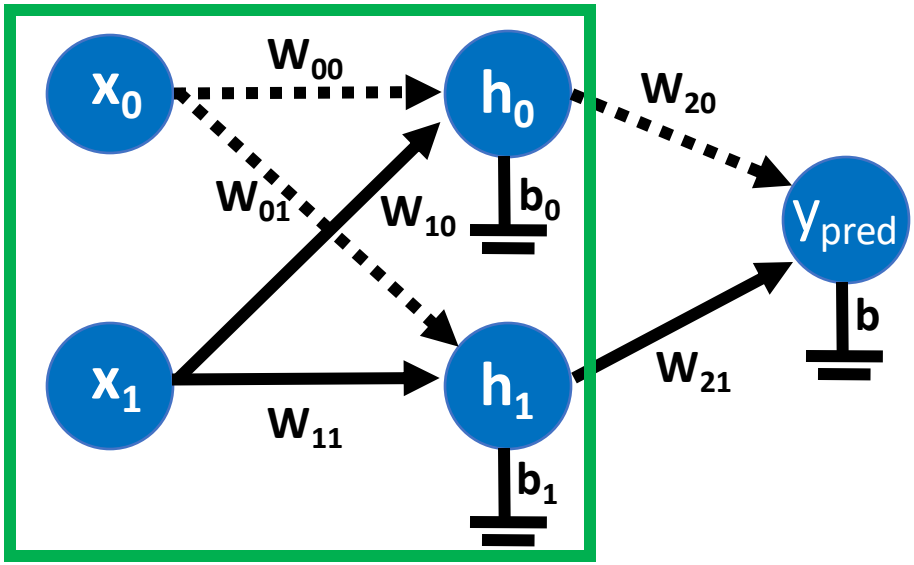
$$\text{loss} = \frac{1}{4}\sum_{i=1}^{4}(y_i - y_{pred,i})^2$$

Use chain rule to propagate loss from output layer back to hidden layer

$$\frac{\partial \text{loss}}{\partial \Theta_u} = \frac{1}{4}\sum_{i=1}^{4}\left[2(y_i - y_{pred,i}) \cdot \frac{\partial f_L(\tilde{y}_{pred,i})}{\partial \tilde{y}_{pred,i}} \cdot \frac{\partial \tilde{y}_{pred,i}}{\partial \Theta_u}\right], \Theta_u = W_{20}, W_{21}, b$$

$$\tilde{y}_{pred,i} = h_0(i) \cdot W_{20} + h_1(i) \cdot W_{21} + b$$

Computing Gradients (2)



$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

$$f_L(r) = \frac{1}{1+\exp(-r)}$$

$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$

$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

$$f_h(r) = \tanh(r)$$

$$\text{loss} = \frac{1}{4}\sum_{i=1}^{4}(y_i - y_{pred,i})^2$$

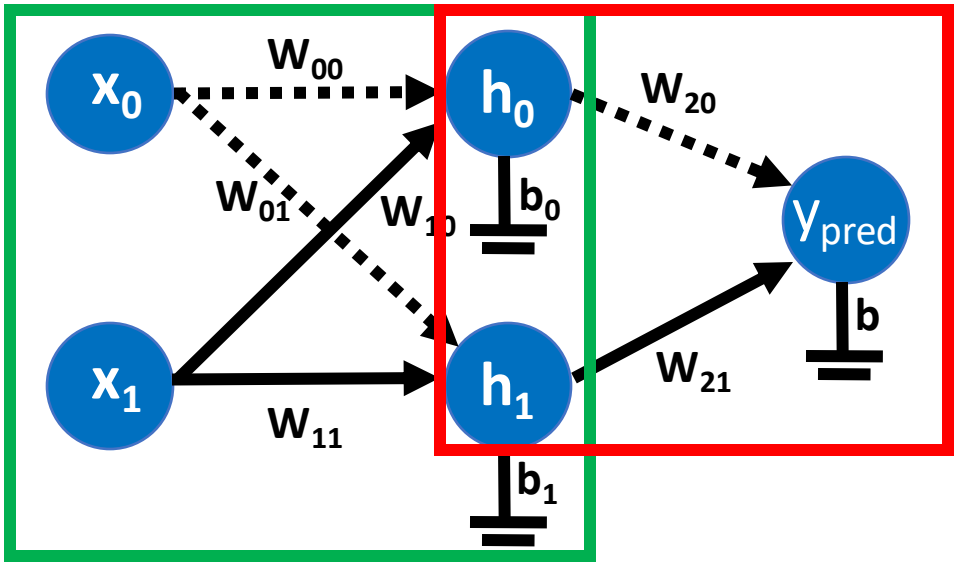
Use chain rule to propagate loss from hidden layer back to input layer

$$\frac{\partial \text{loss}}{\partial \Theta_m} = \frac{1}{4}\sum_{i=1}^{4}\left[2(y_i - y_{pred,i}) \cdot \frac{\partial f_L(\tilde{y}_{pred,i})}{\partial \tilde{y}_{pred,i}} \cdot W_{2m} \cdot \frac{\partial f_h(\tilde{h}_m(i))}{\partial \tilde{h}_m(i)} \cdot \frac{\partial \tilde{h}_m(i)}{\partial \Theta_m}\right]$$

$$\Theta_m = W_{m0}, W_{m1}, b_m$$

$$\tilde{h}_m(i) = x_0(i) \cdot W_{mn} + x_1(i) \cdot W_{mn} + b_m$$

Computing Gradients (2)



$$y_{pred} = f_L(h_0 \cdot W_{20} + h_1 \cdot W_{21} + b)$$

$$f_L(r) = \frac{1}{1+\exp(-r)}$$

$$h_0 = f_h(x_0 \cdot W_{00} + x_1 \cdot W_{01} + b_0)$$

$$h_1 = f_h(x_0 \cdot W_{10} + x_1 \cdot W_{11} + b_1)$$

$$f_h(r) = \tanh(r)$$

$$\text{loss} = \frac{1}{4}\sum_{i=1}^{4}(y_i - y_{pred,i})^2$$

Use chain rule to propagate loss from hidden layer back to input layer

$$\frac{\partial \text{loss}}{\partial \Theta_m} = \frac{1}{4}\sum_{i=1}^{4}\left[2(y_i - y_{pred,i}) \cdot \frac{\partial f_L(\tilde{y}_{pred,i})}{\partial \tilde{y}_{pred,i}} \cdot W_{2m} \cdot \frac{\partial f_h(\tilde{h}_m(i))}{\partial \tilde{h}_m(i)} \cdot \frac{\partial \tilde{h}_m(i)}{\partial \Theta_m}\right]$$

$$\Theta_m = W_{m0}, W_{m1}, b_m$$

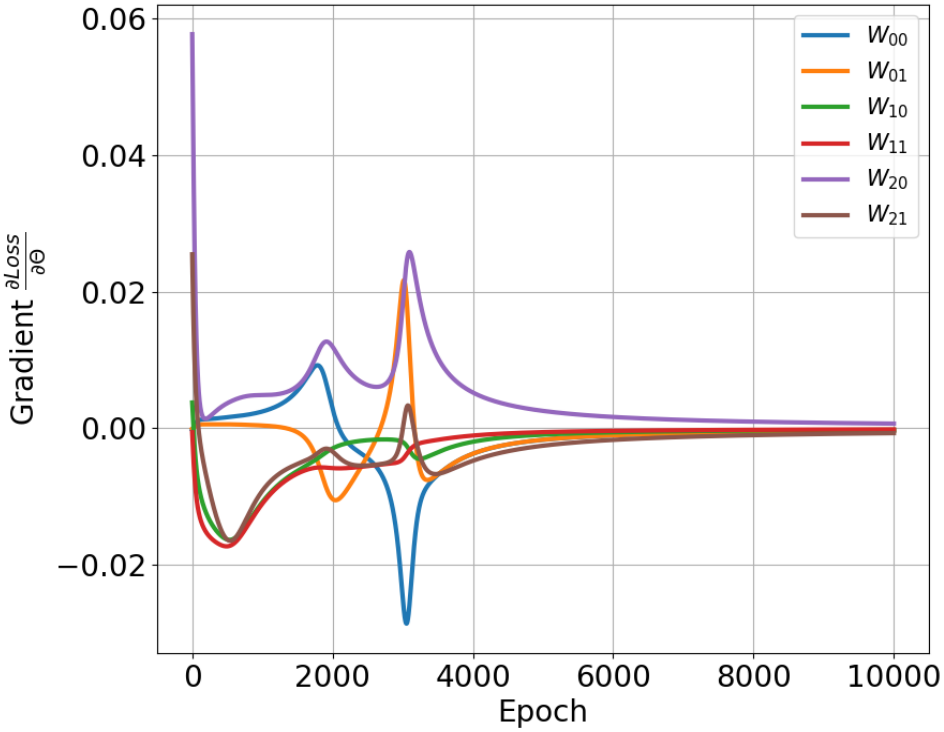
$$\tilde{h}_m(i) = x_0(i) \cdot W_{mn} + x_1(i) \cdot W_{mn} + b_m$$

Parameter Update via Stochastic Gradient Descent (SGD)

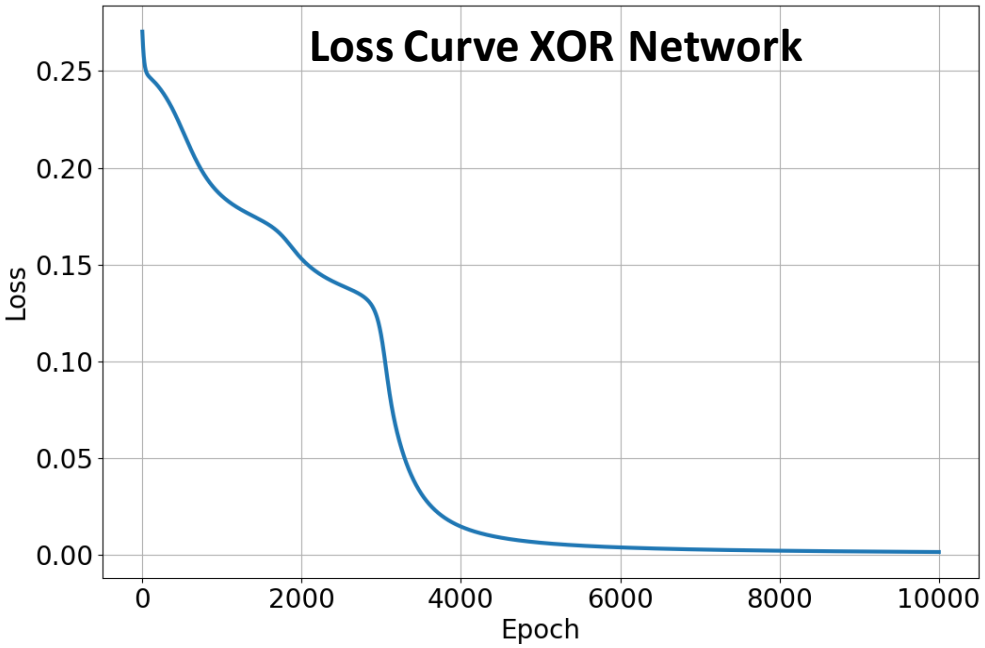
1. Propagate loss back through network (**backpropagation**) --> Get gradients
2. Use gradients to iteratively update parameters (**training**)

$$\Theta_{\text{epoch}+1} = \Theta_{\text{epoch}} - \alpha \cdot \frac{\partial \text{Loss}}{\partial \Theta_{\text{epoch}}}$$

- Step size is adjusted by learning rate α
- Large gradients <--> Large parameter updates
- Small gradients <--> Small parameter updates
- Ideally, gradients converge to 0 at the end of the training



Results for the XOR Network



- Trained XOR network for 10k epochs with learning rate $\alpha = 0.1$
- Loss curve --> Very first plot to check after training a network
- Training successful <--> Loss converges
- Network predictions look reasonable

Input x	Expected Output	Predicted Output	Loss
(0.0,0.0)	0.0	0.04	0.002
(0.0,1.0)	1.0	0.97	0.001
(1.0,0.0)	1.0	0.97	0.001
(1.0,1.0)	0.0	0.05	0.002

Σ Loss = 0.006

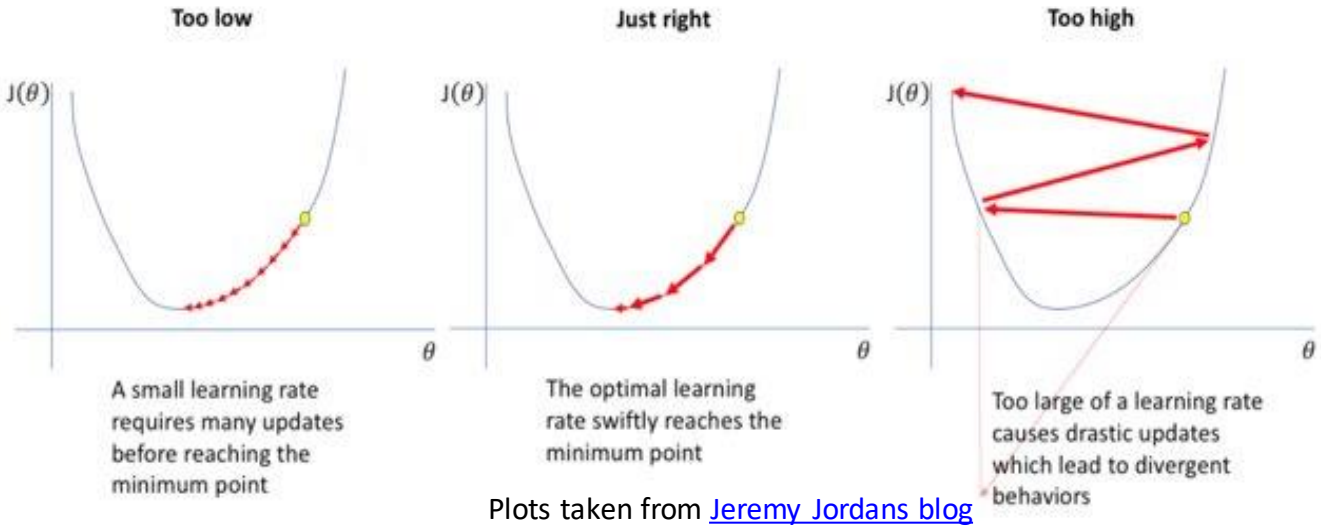
Gradient Descent and Optimizers (1)

- Common notation:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla \text{Loss}(\theta_t, x_i)$$

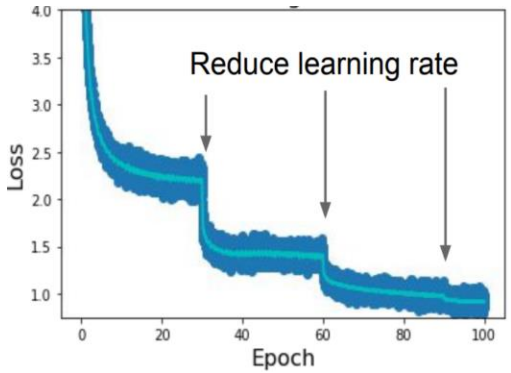
- θ_t: weights matrix, or bias vector of specific layer at training epoch t
- $\nabla = \left(\frac{\partial}{\partial \theta_{0,t}}, \frac{\partial}{\partial \theta_{1,t}}, \ldots \right)$
- Batch size $m \rightarrow$ Take samples from training data
- Learning rate η

Setting the learning rate properly



Plots taken from Jeremy Jordans blog

Decay learning rate over time



Gradient Descent and Optimizers (2)

SGD(H_t, η_t)

$$\theta_{t+1} = \theta_t - \eta_t \nabla \ell(\theta_t)$$

MOMENTUM(H_t, η_t, γ)

$$v_0 = 0$$
$$v_{t+1} = \gamma v_t + \nabla \ell(\theta_t)$$
$$\theta_{t+1} = \theta_t - \eta_t v_{t+1}$$

NESTEROV(H_t, η_t, γ)

$$v_0 = 0$$
$$v_{t+1} = \gamma v_t + \nabla \ell(\theta_t)$$
$$\theta_{t+1} = \theta_t - \eta_t \left(\gamma v_{t+1} + \nabla \ell(\theta_t)\right)$$

RMSPROP$(H_t, \eta_t, \gamma, \rho, \epsilon)$

$$v_0 = 1, m_0 = 0$$
$$v_{t+1} = \rho v_t + (1-\rho)\nabla \ell(\theta_t)^2$$
$$m_{t+1} = \gamma m_t + \frac{\eta_t}{\sqrt{v_{t+1} + \epsilon}} \nabla \ell(\theta_t)$$
$$\theta_{t+1} = \theta_t - m_{t+1}$$

ADAM$(H_t, \alpha_t, \beta_1, \beta_2, \epsilon)$

$$m_0 = 0, v_0 = 0$$
$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla \ell(\theta_t)$$
$$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla \ell(\theta_t)^2$$
$$b_{t+1} = \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}}$$
$$\theta_{t+1} = \theta_t - \alpha_t \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} b_{t+1}$$

NADAM$(H_t, \alpha_t, \beta_1, \beta_2, \epsilon)$

$$m_0 = 0, v_0 = 0$$
$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla \ell(\theta_t)$$
$$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla \ell(\theta_t)^2$$
$$b_{t+1} = \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}}$$
$$\theta_{t+1} = \theta_t - \alpha_t \frac{\beta_1 m_{t+1} + (1-\beta_1)\nabla \ell(\theta_t)}{\sqrt{v_{t+1}} + \epsilon} b_{t+1}$$

Taken from On Empirical Comparisons of Optimizers for Deep Learning

Tensors

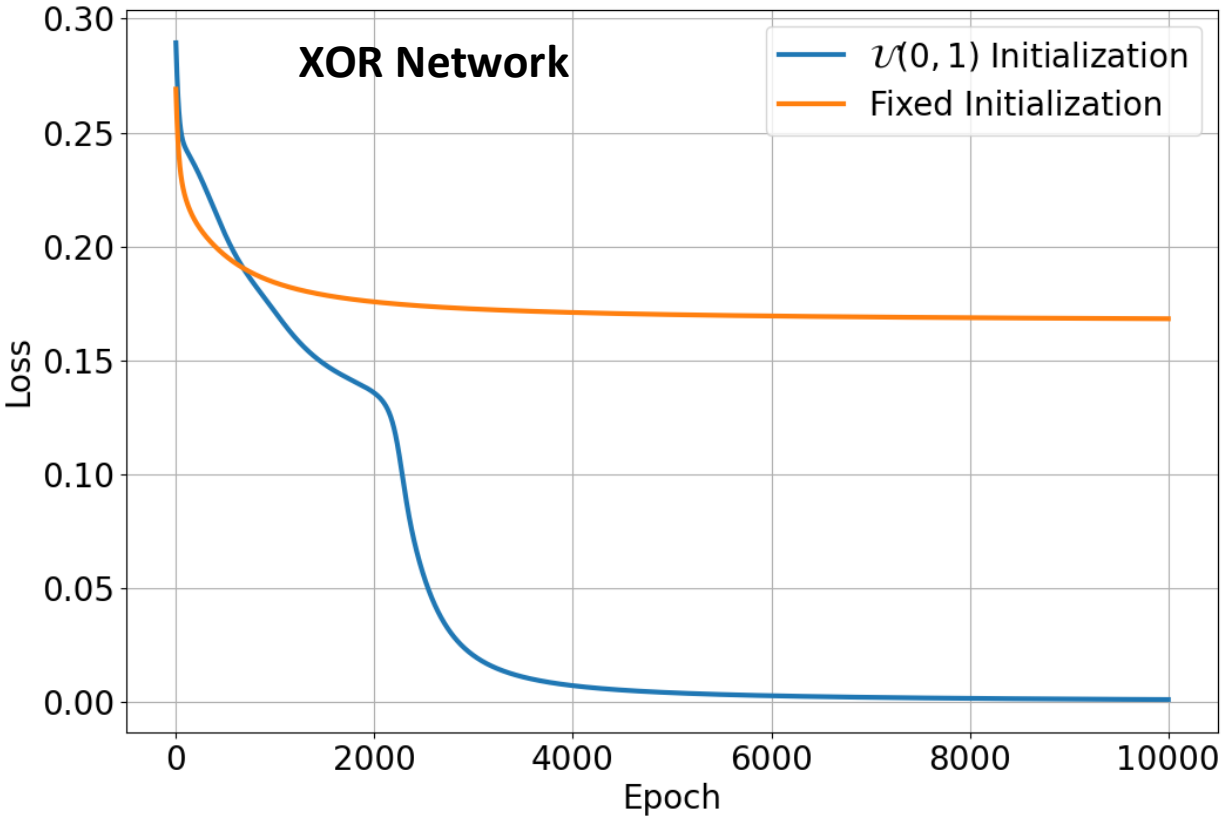
Rank	Shape	Dimension number	Example	
0	[]	0-D	A 0-D tensor. A scalar.	**a single number**
1	[D0]	1-D	A 1-D tensor with shape [5].	**a 5-dim vector**
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].	**a 3x4 matrix**
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].	**a cube**
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].	**I am lost…**

Table from tensorflow

- Forward / backward pass of data through network is expressed via tensor operations
- Weight matrix W connecting layers h and h+1
- Bias vector $\vec{b}_{h+1}$ from layer h+1
- Response from previous layer h: $\vec{S}_h$
- Get response in adjacent layer: $\vec{S}_{h+1} = W \cdot \vec{S}_h + \vec{b}_{h+1}$

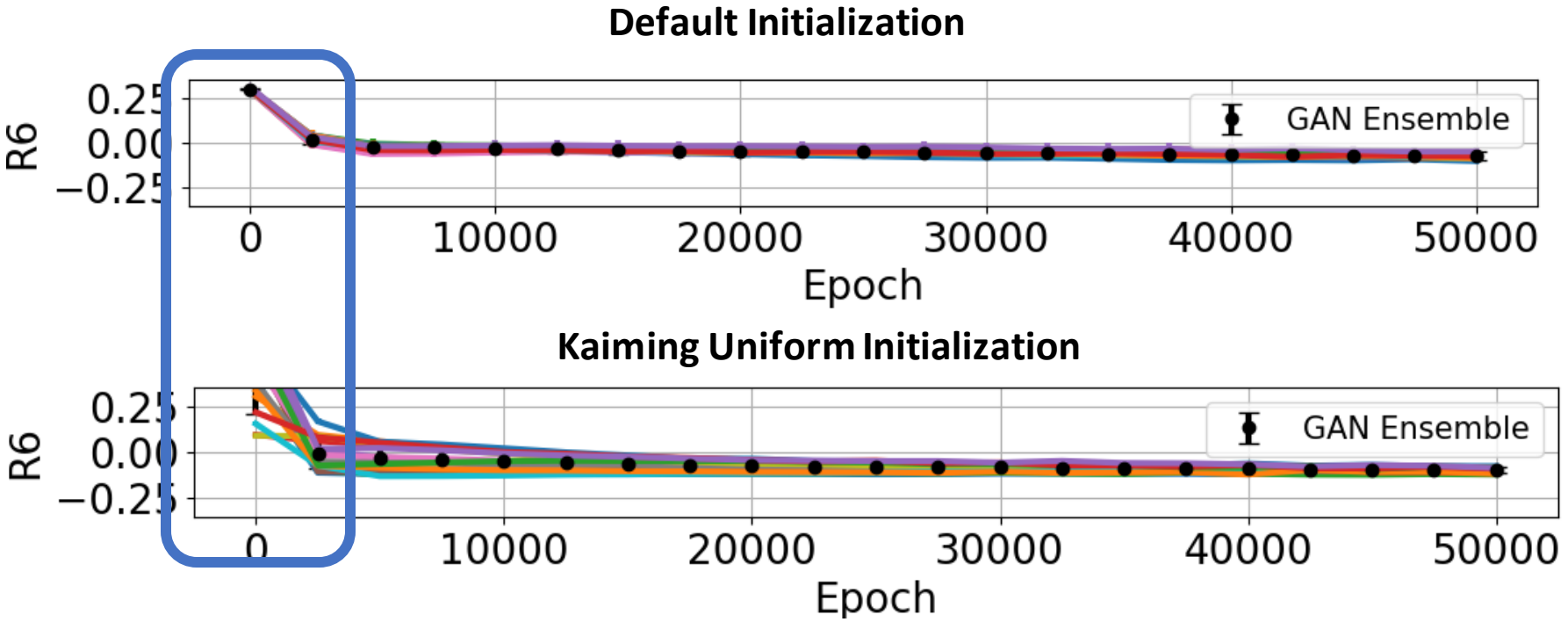
Parameter Initialization (1)

- Defines initial state of network
- Wrong initialization affects training
  - Network starts far off any optimum --> No convergence
  - Network settles in one optimum --> No further learning
- Different initialization types available
- May depend on activation function
- Typical initialization for bias: 0, Uniform, Normal



Activation Function	Suggested Weight Initialization
Linear, Sigmoid, Tanh	Xavier Uniform / Normal
ReLU, Leaky ReLU	Kaiming Uniform / Normal

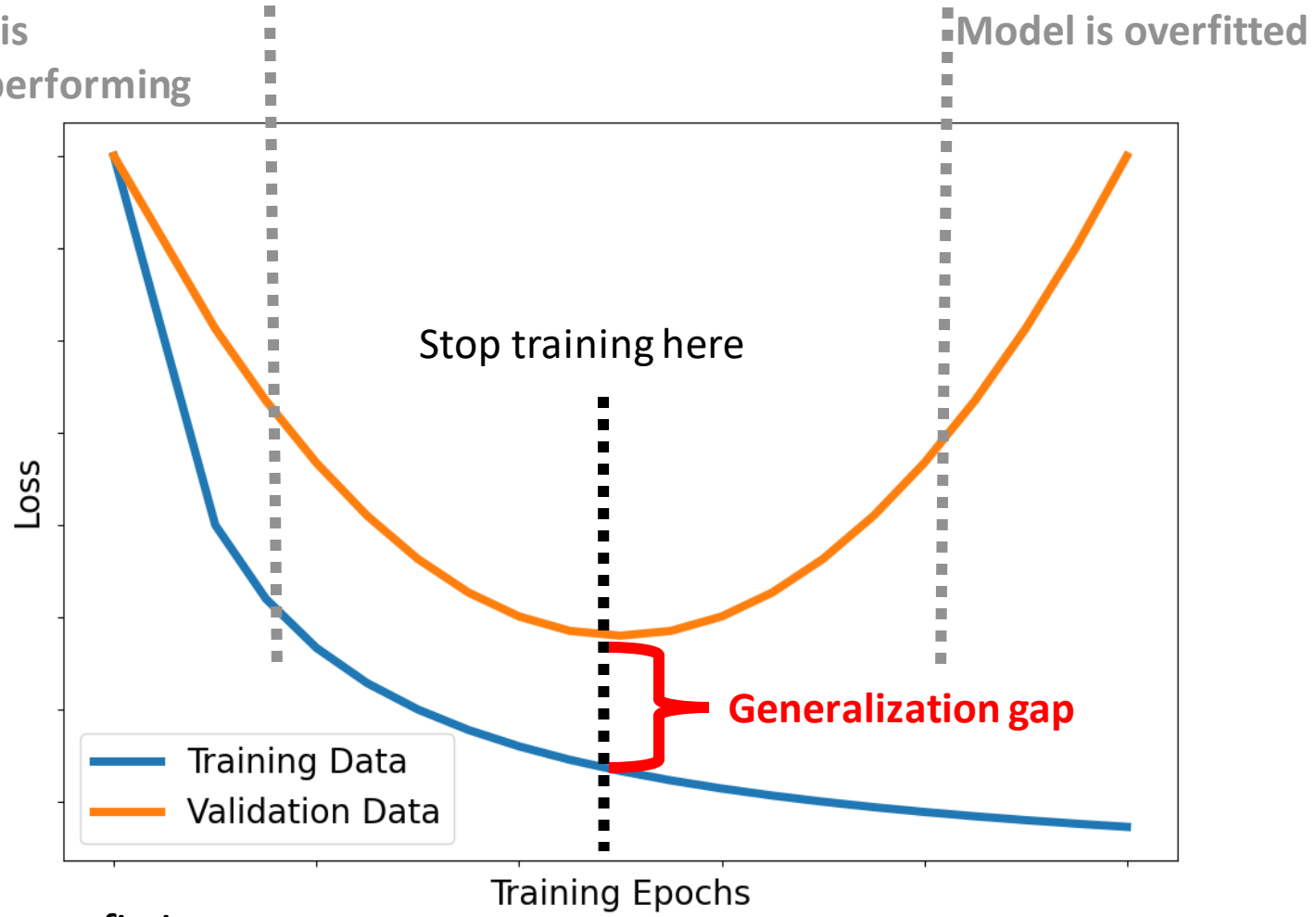
Parameter Initialization (2)



- Ran analysis with network that used Leaky ReLU in activation layers
- At first, did not check default weight initialization **(provided by software that I was using)**
- Adjusted initialization according to activation function

Always check the parameter initialization in your network!

Early Stopping

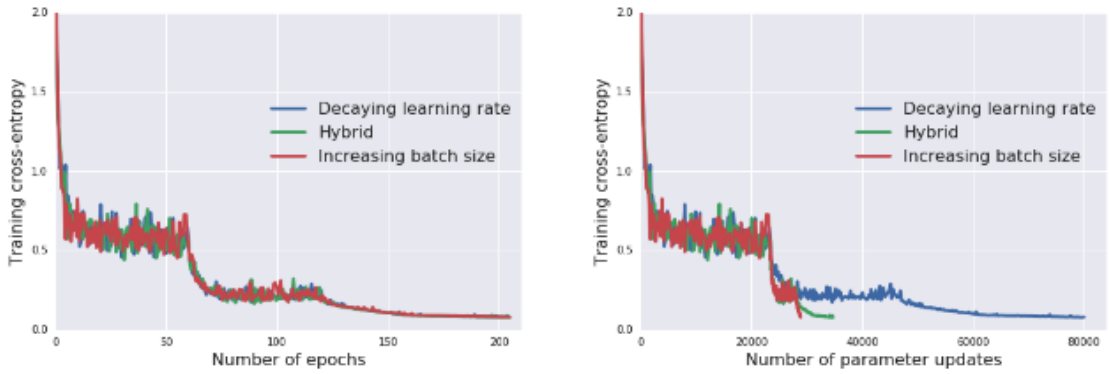


- Avoid overfitting
- Terminate training when generalization gap is minimal
- Provided by most software packages, e.g. scikit-learn early stopping

The Batch Size

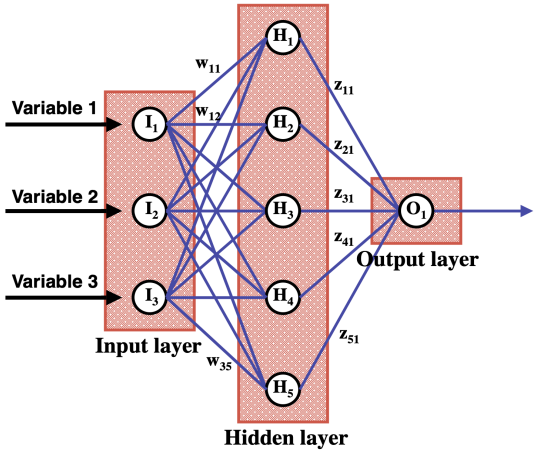
$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{m} \cdot \sum_{i=1}^{m} \nabla \text{Loss}(\theta_t, x_i)$$

- Determines how many data samples are seen by model during gradient computation
- Batch size too small --> Large bias and variance in gradients
- Batch size too large --> Computational cost / memory issues on GPU
- Often recommended: m ~ 16 , 32
- However: Some models benefit from larger batch sizes, e.g. GANs as discussed in this paper
- This paper suggest to keep the learning rate constant and increase the batch size



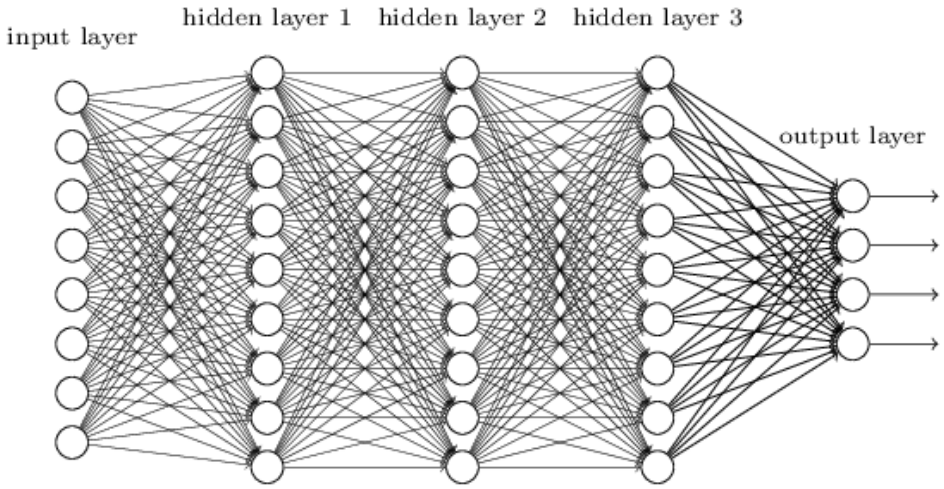
Now what is Deep Learning ?

Machine Learning



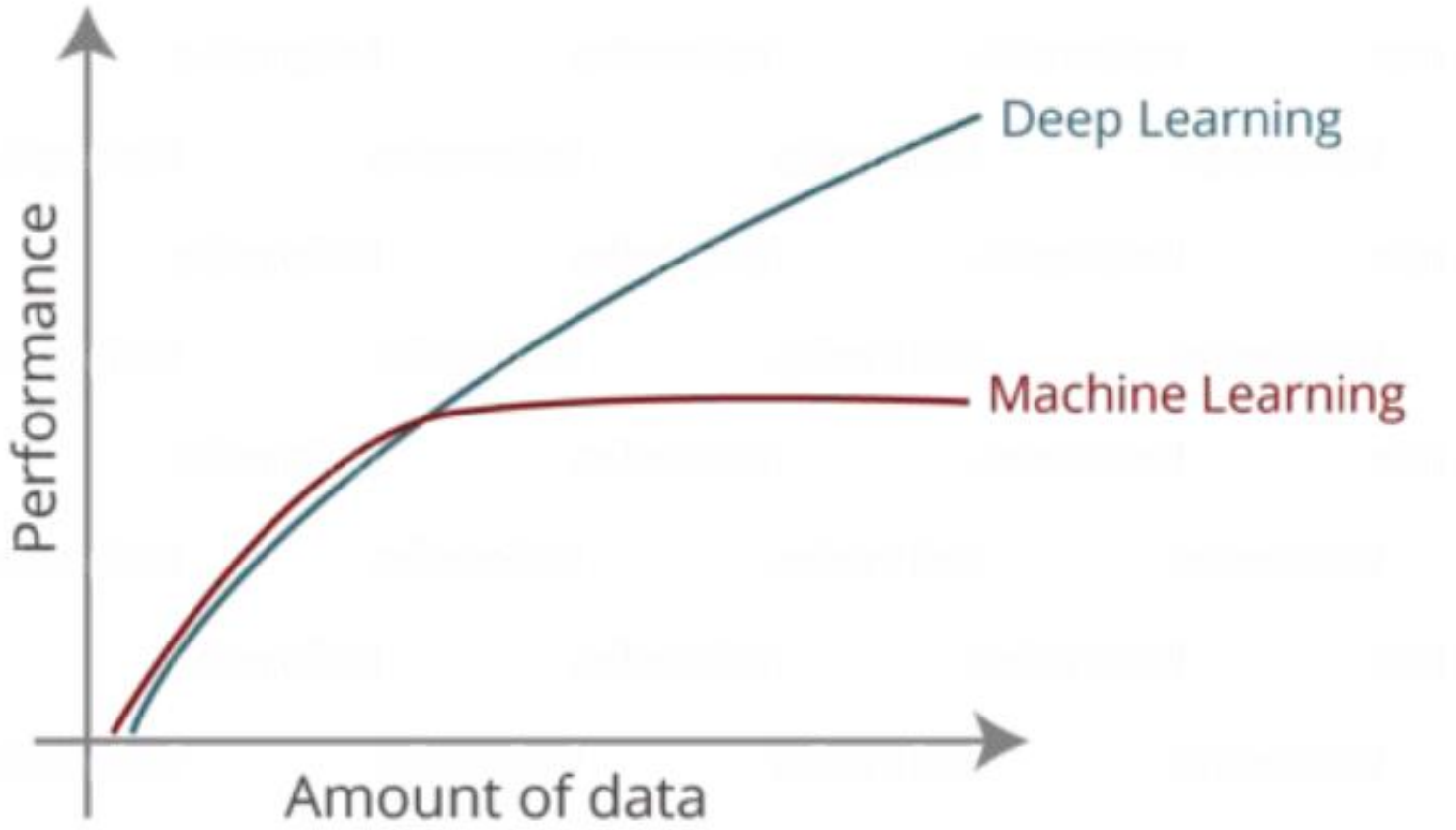
- Variety of algorithms
- Multilayer perceptrons < 3 hidden layers
- Decision trees
- Linear classifier
- ...

Deep Learning



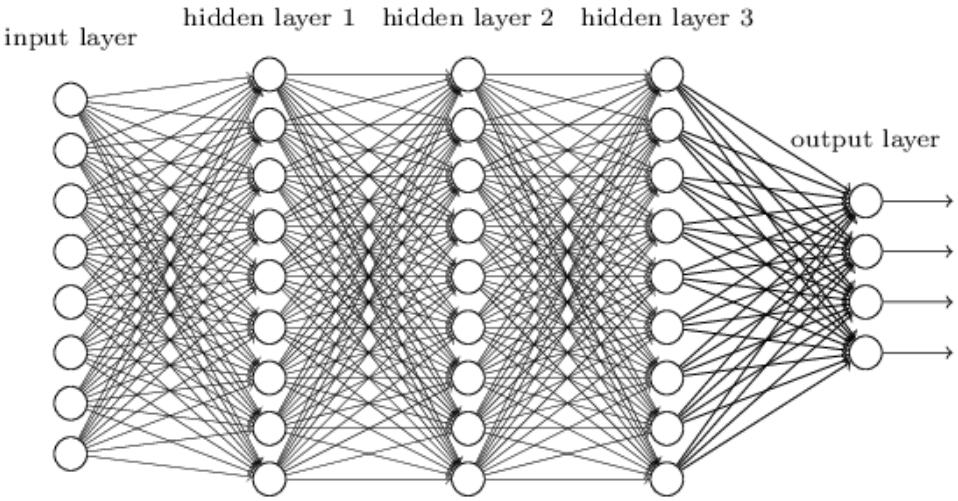
- Large neural networks
- Multilayer perceptrons >= 3 hidden layers
- Convolutional neural networks (computer vision)
- Graph neural networks
- Language models (Chat GPT)
- ....

Why Deep Learning ?



Plot taken from [Mustafa Mustafas talk at deep learning for science school 2019]

Challenges in Deep Learning



Need gradients for weight updates

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla \text{Loss}(\theta_t, x_i)$$

No gradients, no updates

$$\nabla \text{Loss} = 0 \Rightarrow \theta_{t+1} = \theta_t$$

- Computationally intensive --> Many algebraic operations --> **Utilize GPUs**
- Every additional layer adds a factor to the loss derivative **(chain rule!)**
- Vanishing gradient problem --> Zero gradients --> No weight updates
- Overfitting --> So many parameters
- Larger models (e.g. Chat GPT) require distributed training across multiple GPUs

Summary & Outlook

- Machine learning workflow
  - Same for nearly all tasks **(classification, regression,…)**
  - Used PID on fake data as an example
  - Discussed performance evaluation metrics
  - More examples in **"Machine Learning for Nuclear Physics: Lecture 3"**, Thu. 06/06/2024, Torri Jeske
- Neural networks
  - Components of multilayer perceptron
  - Backpropagation and gradient descent
  - Weight initialization, learning rate, batch size
  - Overfitting
- Deep learning
  - Model complexity
  - Challenges in training