Moderne Methoden der Datenanalyse

The NN training

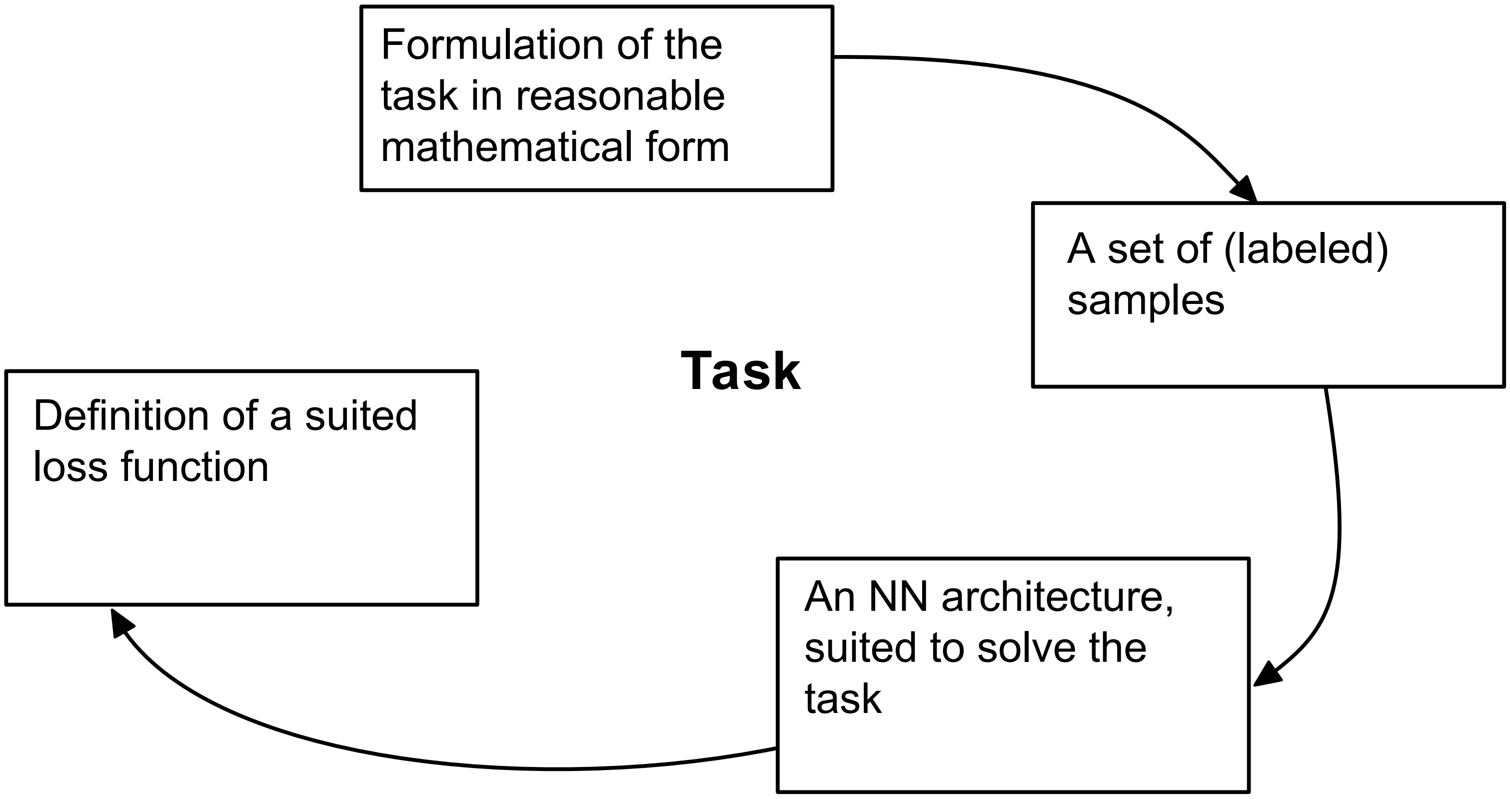
Roger Wolf

14. July 2022

Content of this lecture

- Preparation for training and practical training aspects.
- Challenges during training and application and how to cope with them.
- Assessment of the training.

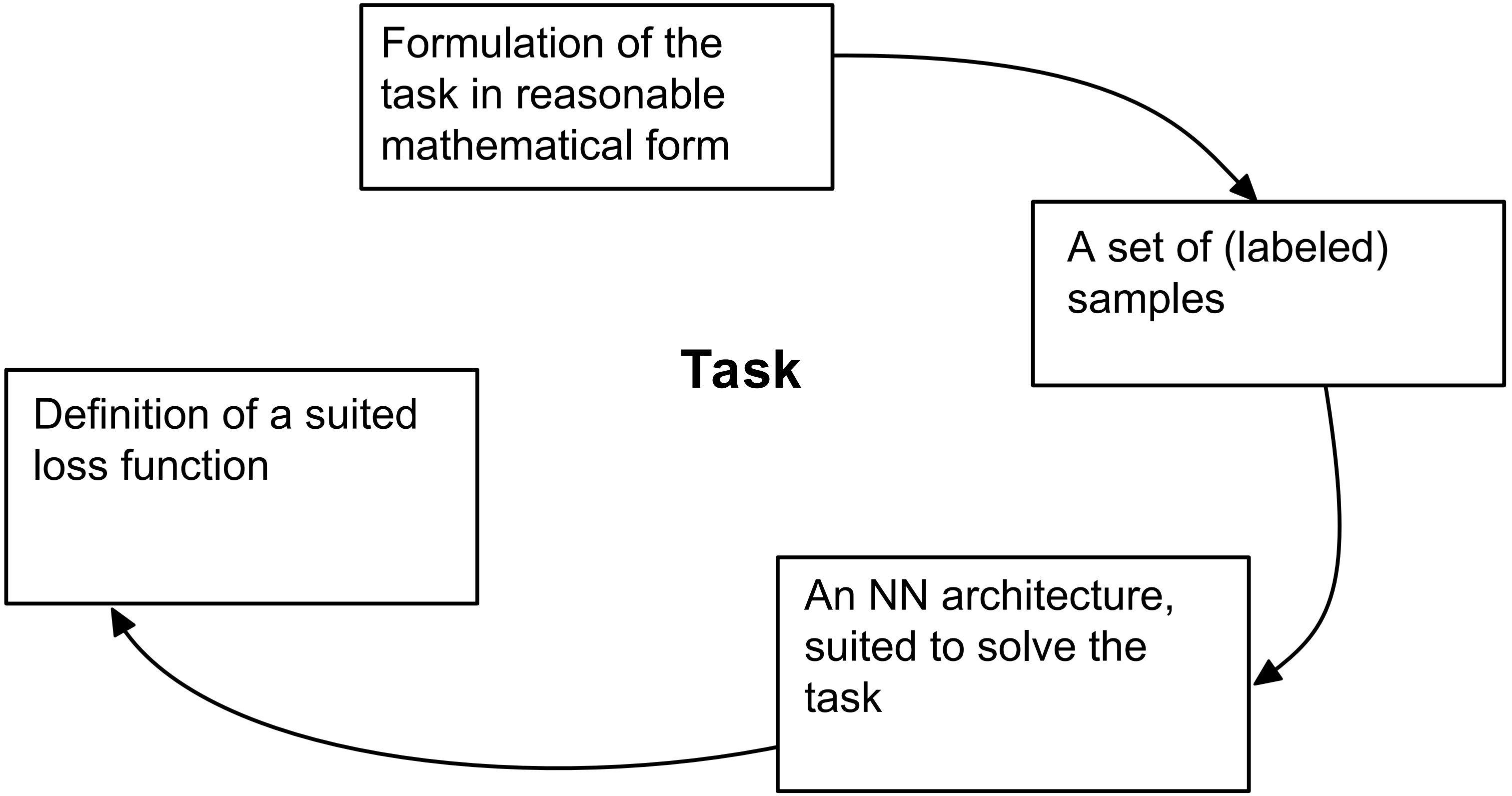
The NN Task


Formulation of the task in reasonable mathematical form
A set of (labeled) samples
Task
Definition of a suited loss function
An NN architecture, suited to solve the task

The NN Task

Label for sample (ℓ) :

$$y_j(x^{(\ell)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$



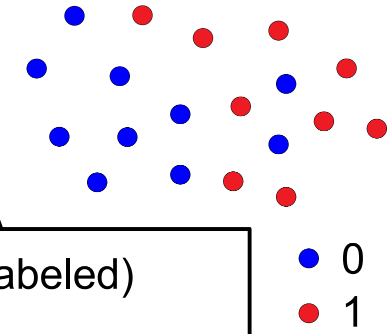
The NN Task

Label for sample (ℓ) :

$$y_j(x^{(\ell)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

Formulation of the task in reasonable mathematical form

A set of (labeled) samples



Task

Definition of a suited loss function

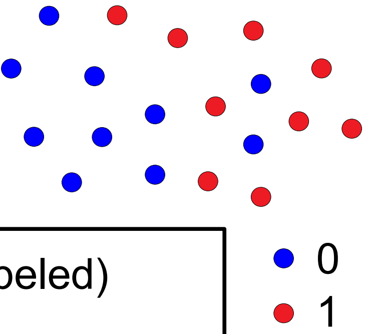
An NN architecture, suited to solve the task

The NN Task

Label for sample (ℓ) :

$$y_j(x^{(\ell)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

Formulation of the task in reasonable mathematical form

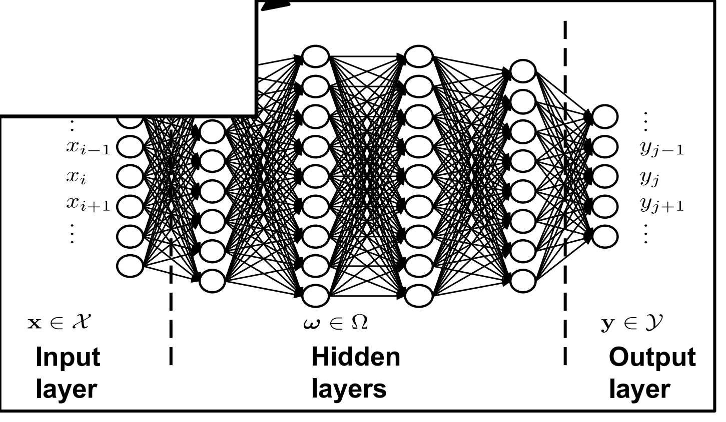


A set of (labeled) samples

Task

Definition of a suited loss function

An NN architecture, suited to solve the task

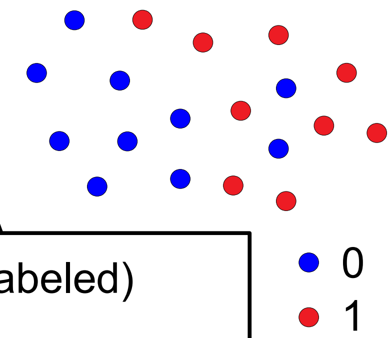


The NN Task

Label for sample (ℓ) :

$$y_j(x^{(\ell)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

Formulation of the task in reasonable mathematical form

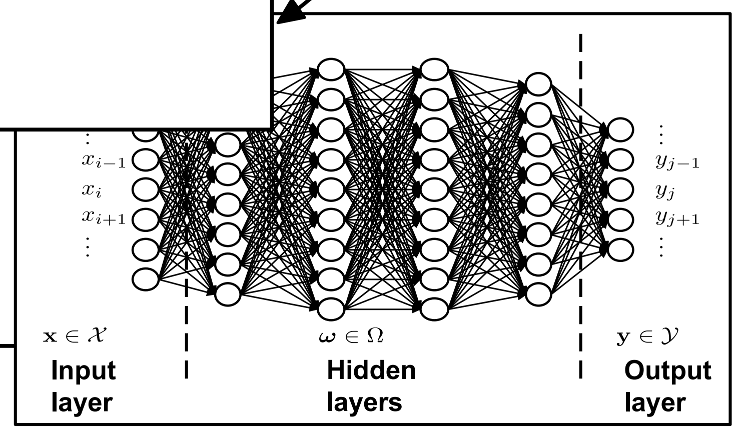


A set of (labeled) samples

Task

Definition of a suited loss function

An NN architecture, suited to solve the task



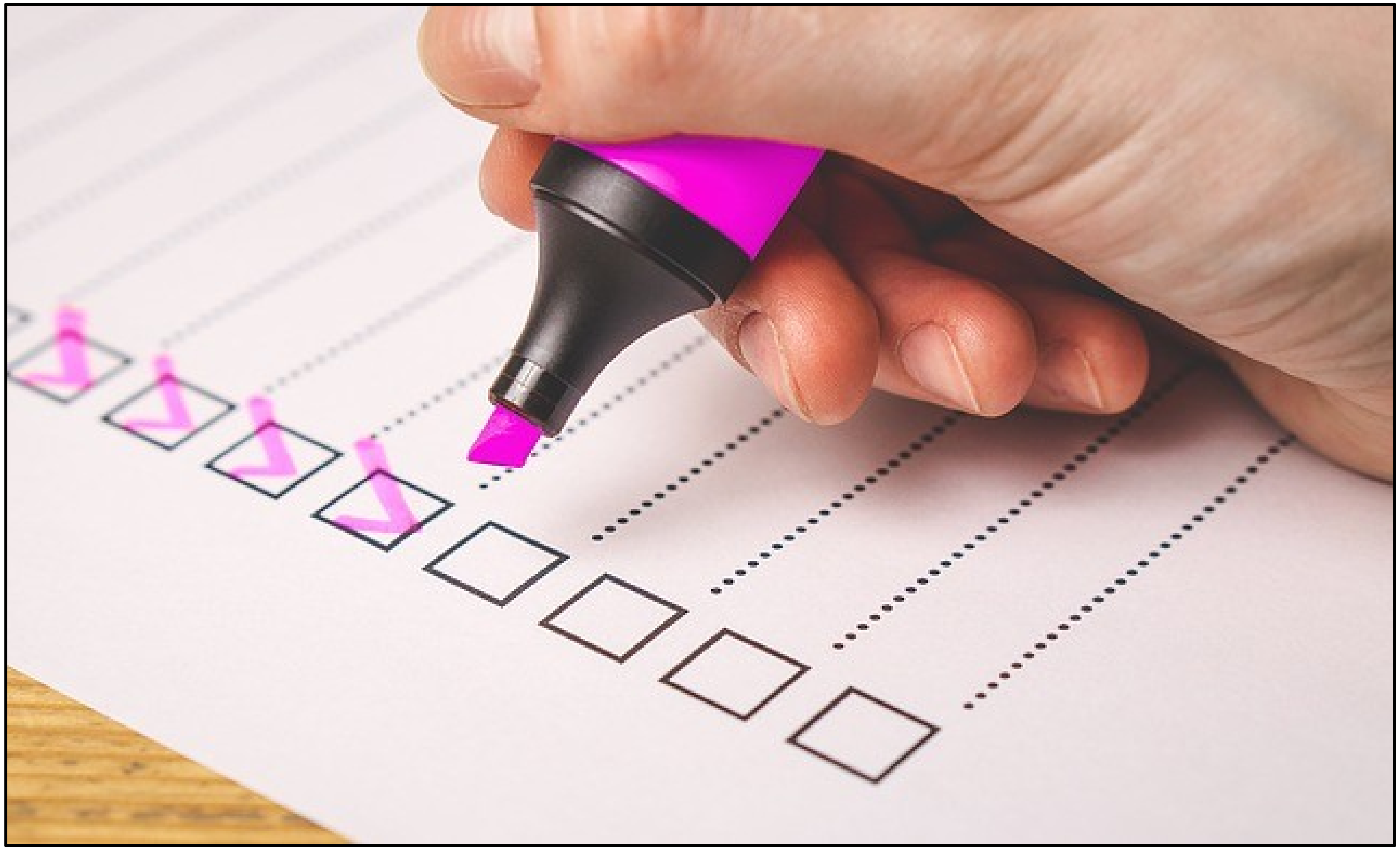
$$L(\{y_j^{(\ell)}\}, \{\hat{y}_j^{(\ell)}\}) = -\sum_{j=0}^{1} y_j^{(\ell)} \log\left(\hat{y}_j^{(\ell)}\right)$$

n : Number of classes

$y_j^{(\ell)}$: Truth label for sample (ℓ)

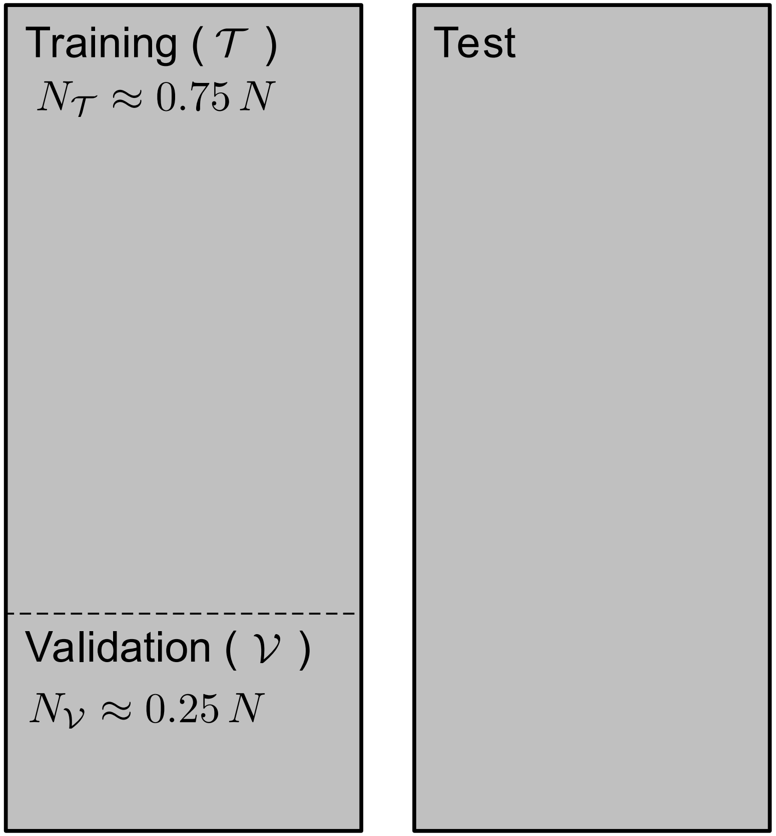
$\hat{y}_j^{(\ell)}$: NN prediction for sample (ℓ)

Preparation for training



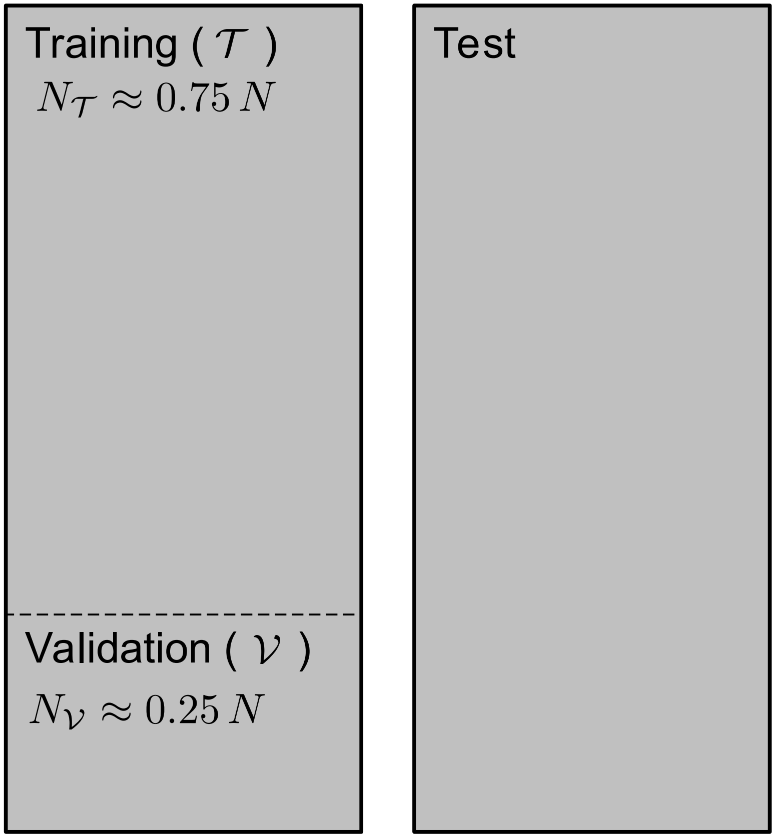
Preparation for training

- **Test dataset**:
  The data that the NN will be applied to.
- **Training dataset ($\mathcal{T}$)**:
  The data that the NN will be trained on.
- **Validation dataset ($\mathcal{V}$)**:
  The data that the NN will be validated on during training.



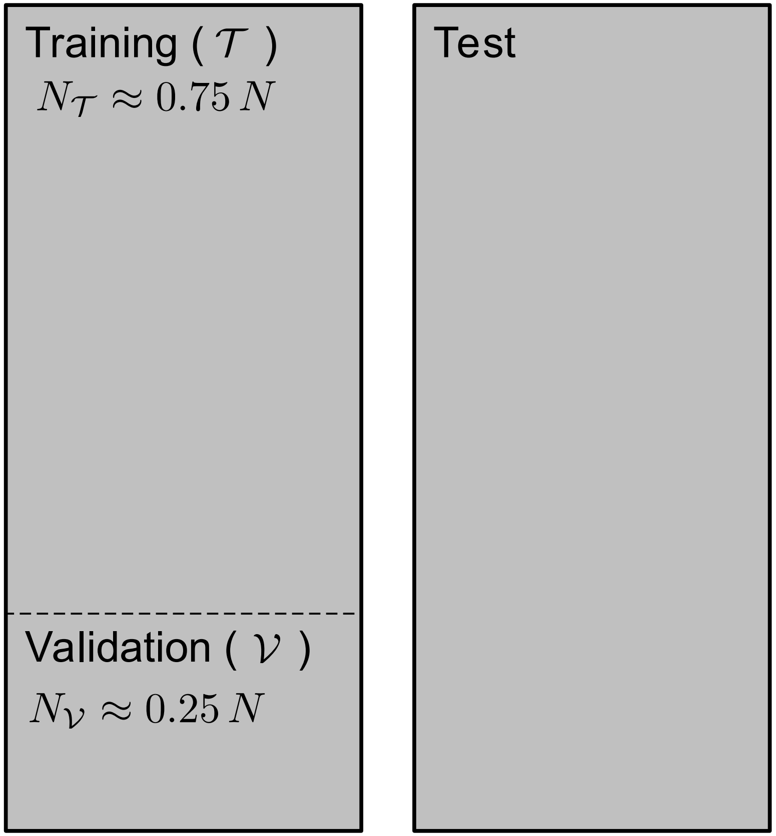
K-fold cross validation

- In particle physics we use the data of our **background model for training**.



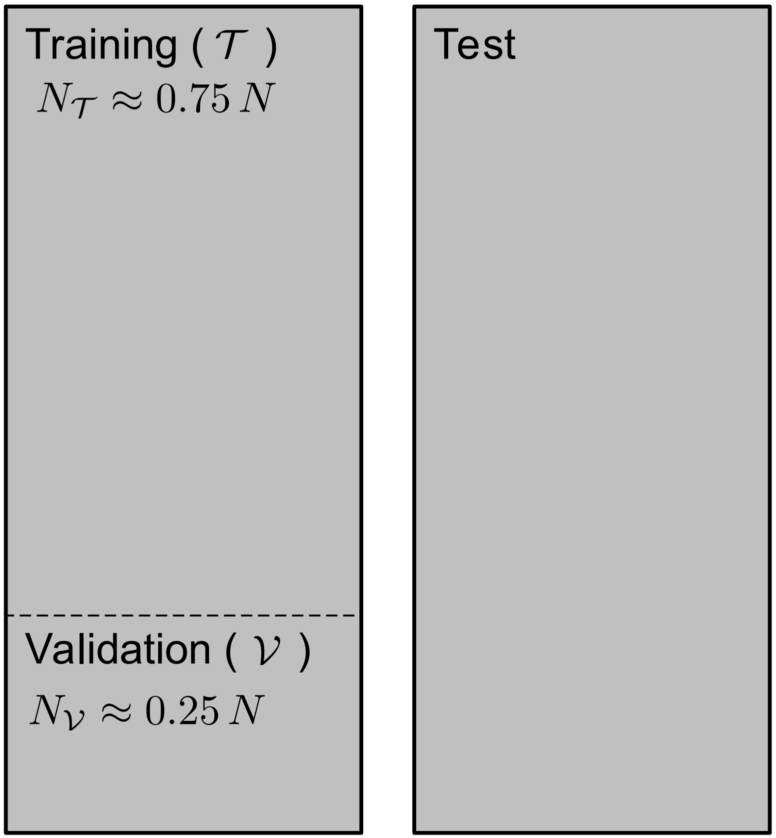
K-fold cross validation

- In particle physics we use the data of our **background model for training**.
- Splitting those in stat. independent training and test datasets may imply a significant loss of sample size for signal extraction.



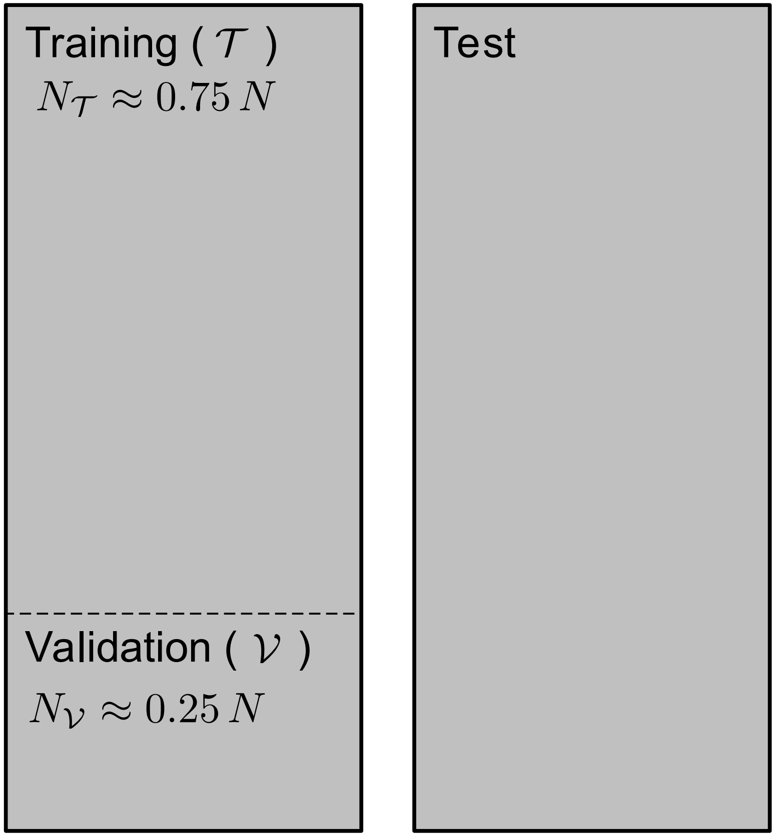
K-fold cross validation

- In particle physics we use the data of our **background model for training**.
- Splitting those in stat. independent training and test datasets may imply a significant loss of sample size for signal extraction.
- A possibility to keep stat. independent datasets but still use full sample sizes for signal extraction is **k-fold cross validation**:



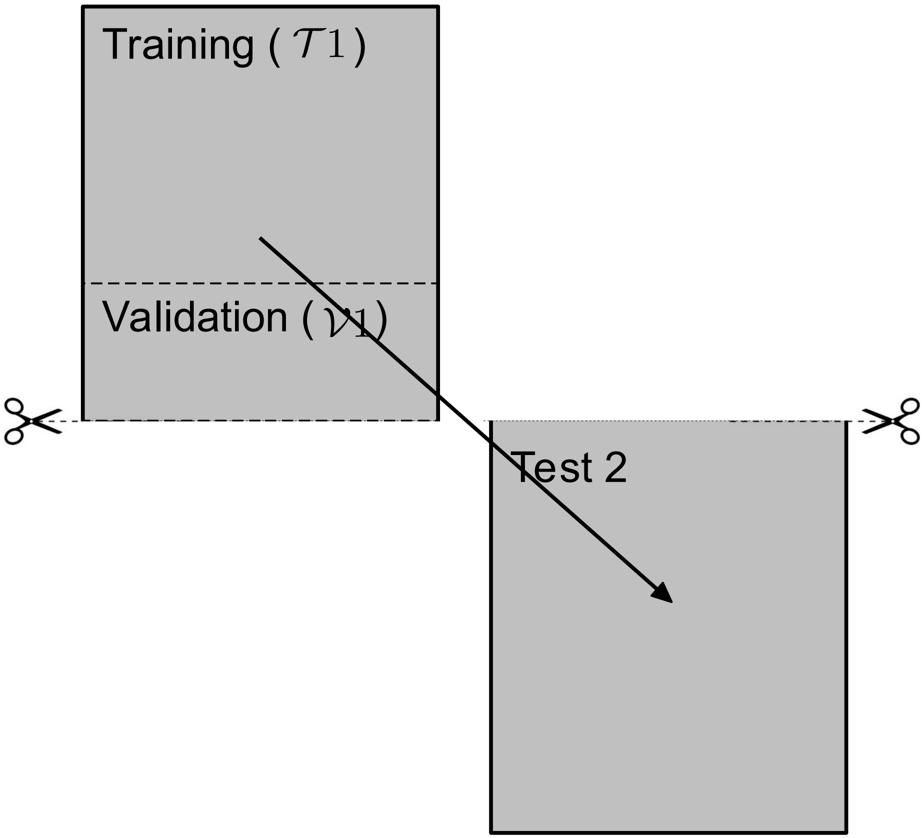
K-fold cross validation

- In particle physics we use the data of our **background model for training**.
- Splitting those in stat. independent training and test datasets may imply a significant loss of sample size for signal extraction.
- A possibility to keep stat. independent datasets but still use full sample sizes for signal extraction is **k-fold cross validation**:
- The training and test datasets are split in k parts. The training is performed k times. Finally the results of the k trainings are added.



K-fold cross validation

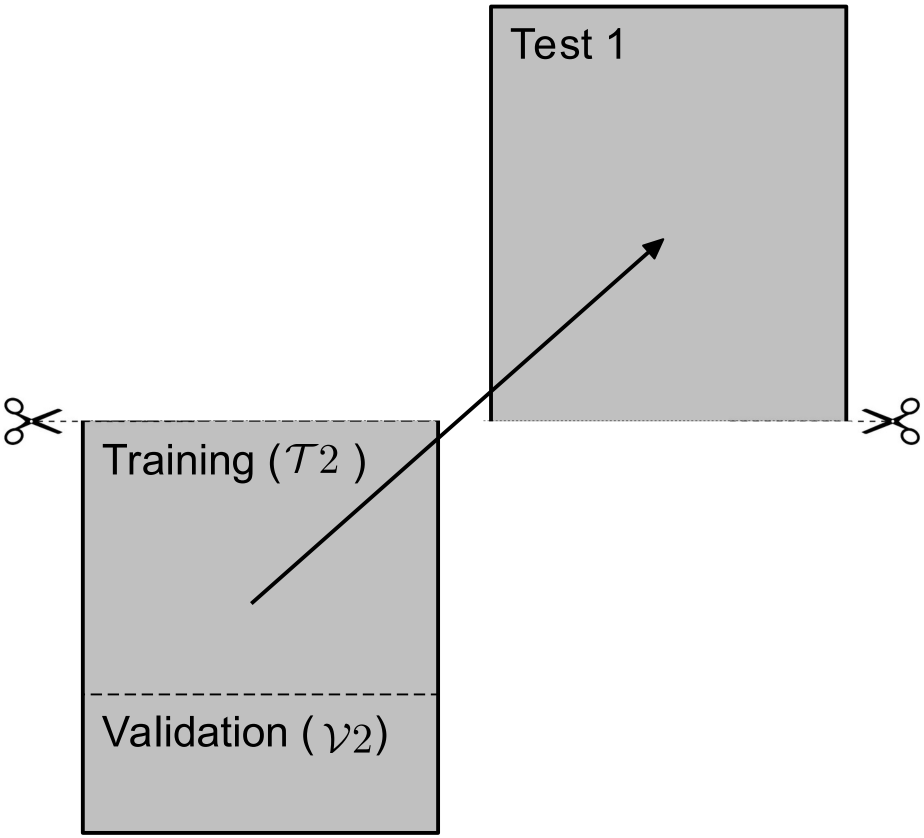
- In particle physics we use the data of our **background model for training**.
- Splitting those in stat. independent training and test datasets may imply a significant loss of sample size for signal extraction.
- A possibility to keep stat. independent datasets but still use full sample sizes for signal extraction is **k-fold cross validation**:
- The training and test datasets are split in k parts. The training is performed k times. Finally the results of the k trainings are added.



(Here shown for *2-fold cross validation*)

K-fold cross validation

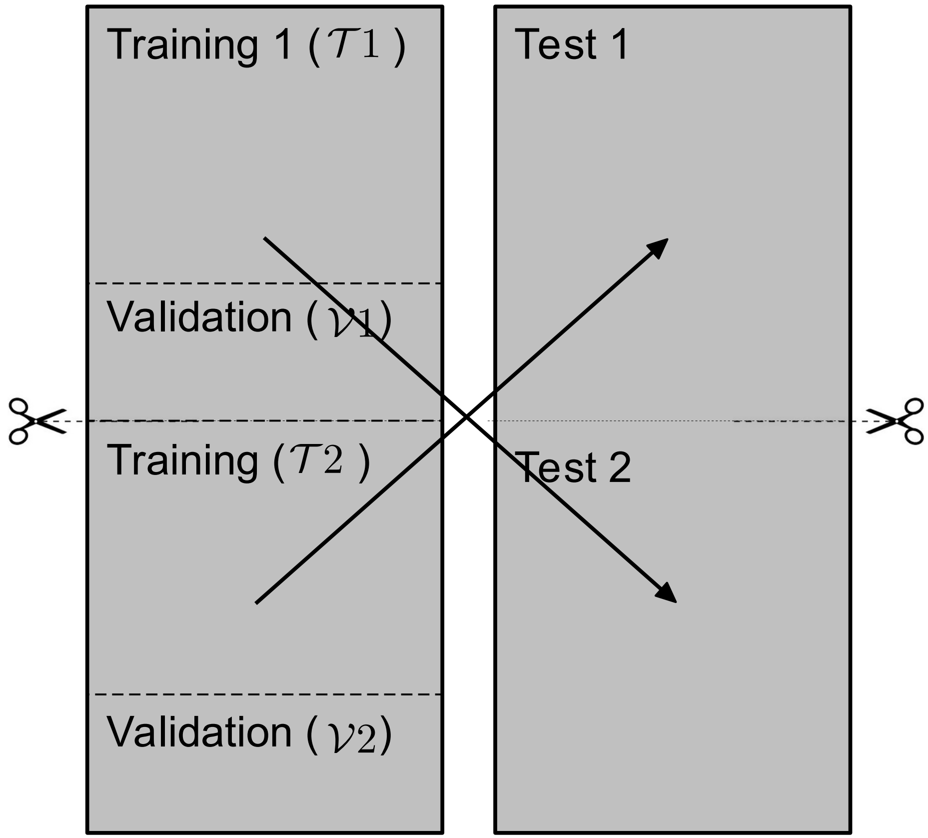
- In particle physics we use the data of our **background model for training**.
- Splitting those in stat. independent training and test datasets may imply a significant loss of sample size for signal extraction.
- A possibility to keep stat. independent datasets but still use full sample sizes for signal extraction is **k-fold cross validation**:
- The training and test datasets are split in k parts. The training is performed k times. Finally the results of the k trainings are added.



(Here shown for *2-fold cross validation*)

K-fold cross validation

- In particle physics we use the data of our **background model for training**.
- Splitting those in stat. independent training and test datasets may imply a significant loss of sample size for signal extraction.
- A possibility to keep stat. independent datasets but still use full sample sizes for signal extraction is **k-fold cross validation**:
- The training and test datasets are split in k parts. The training is performed k times. Finally the results of the k trainings are added.



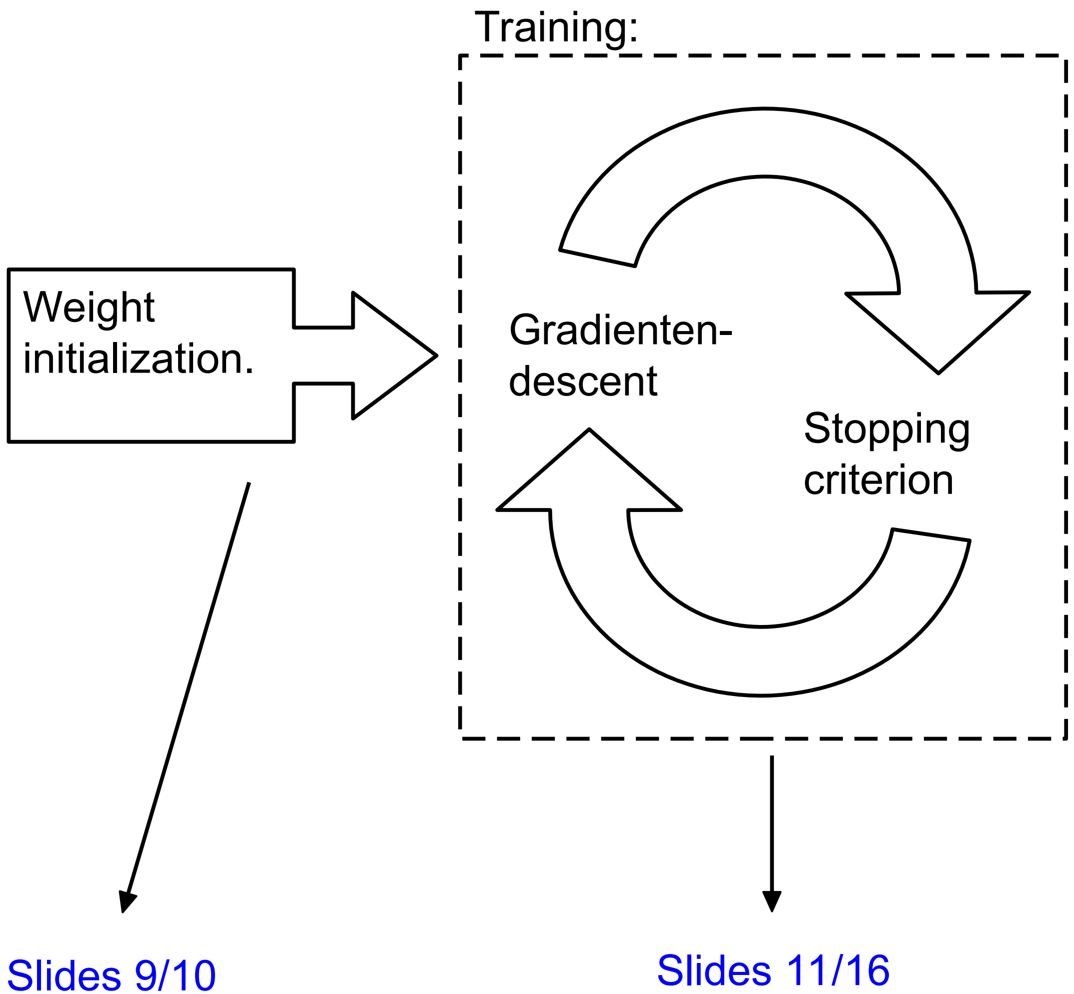
(Here shown for *2-fold cross validation*)

Training procedure

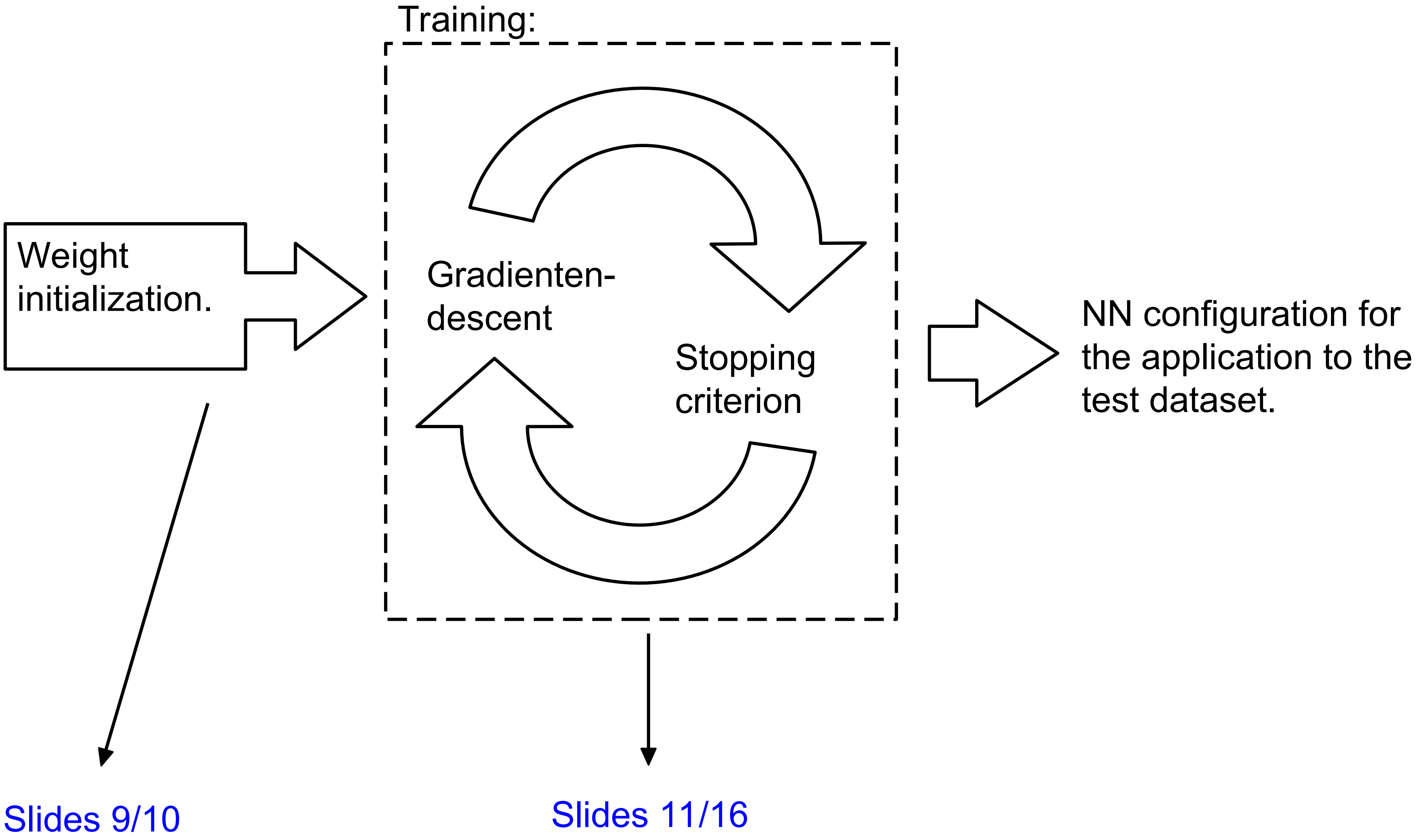
Weight initialization.

Slides 9/10

Training procedure

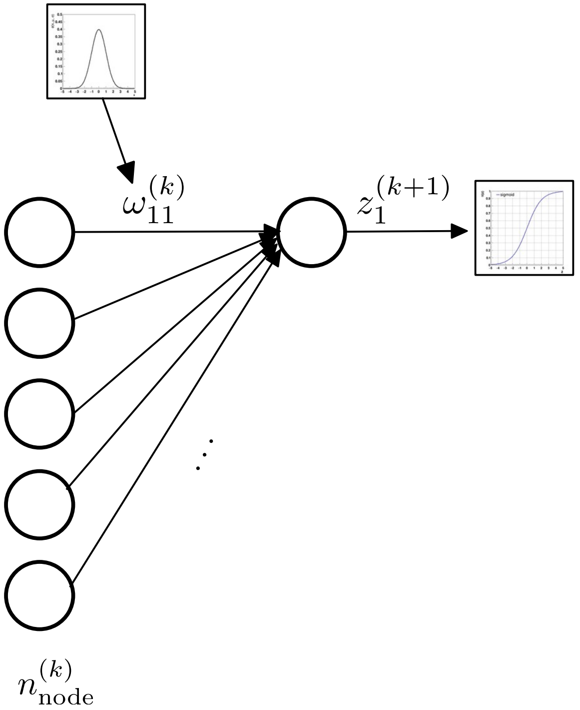


Training procedure



Initialization of trainable parameters (TPs)

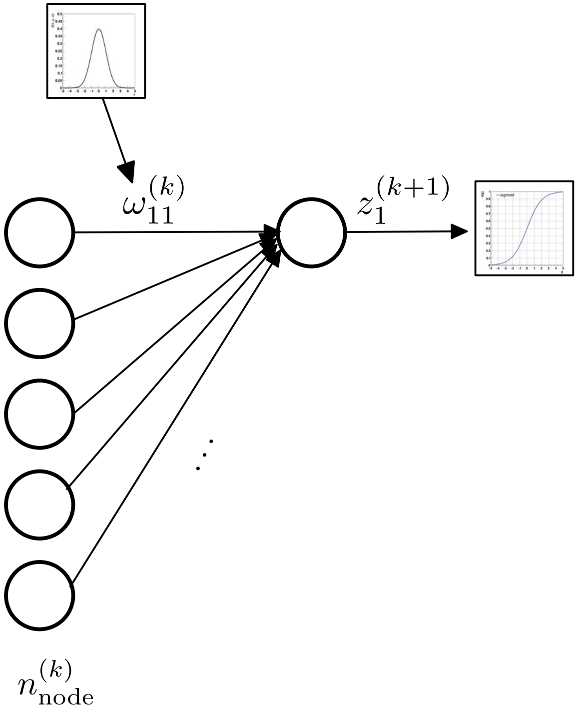
- Thresholds/biases are usually initialized to 0.
- The initialization of the weights happens randomly following a normal or uniform distribution.
- Naive ansatz:



Here (k) stands for layer k.

Initialization of trainable parameters (TPs)

- Thresholds/biases are usually initialized to 0.
- The initialization of the weights happens randomly following a normal or uniform distribution.
- Naive ansatz:
  - Assume all weights to be initialized as standard normal distributed:
    $\mu(\omega_{ij}^{(k)}) = 0,\ \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$
  - Then $z_i^{(k+1)}$ is also normal distributed, with what variance?



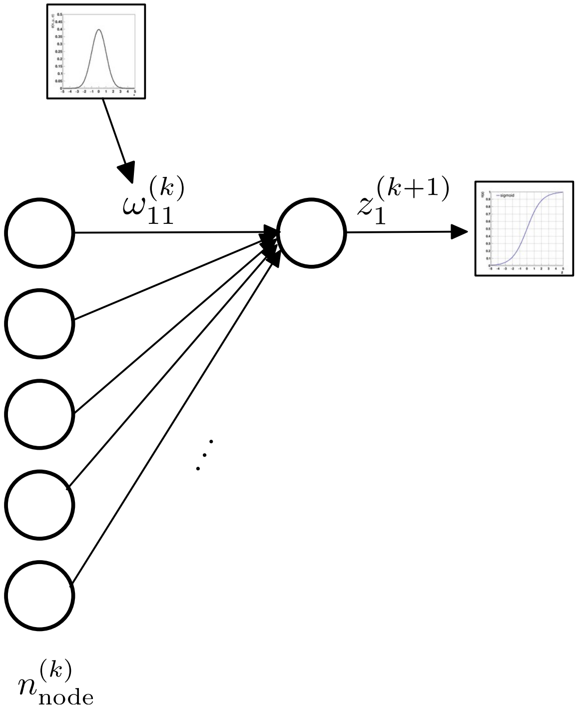
Here (k) stands for layer k.

Initialization of trainable parameters (TPs)

- Thresholds/biases are usually initialized to 0.

- The initialization of the weights happens randomly following a normal or uniform distribution.

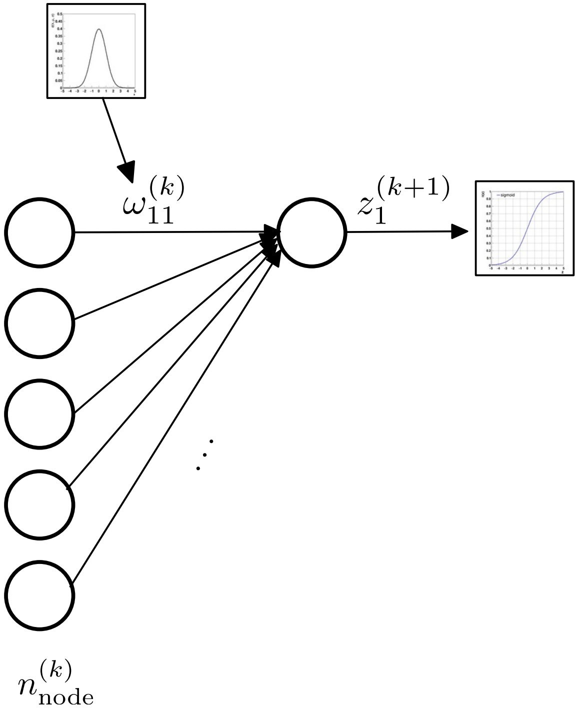
- Naive ansatz:
  - Assume all weights to be initialized as standard normal distributed:
    $\mu(\omega_{ij}^{(k)}) = 0,\, \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$
  - Then $z_i^{(k+1)}$ is also normal distributed, with what variance? – $\sigma(z_i^{(k+1)})^2 = n_{\mathrm{node}}^{(k)}$



Here (k) stands for layer k.

Initialization of trainable parameters (TPs)

- Thresholds/biases are usually initialized to 0.
- The initialization of the weights happens randomly following a normal or uniform distribution.
- Naive ansatz:
  - Assume all weights to be initialized as standard normal distributed:
    $\mu(\omega_{ij}^{(k)}) = 0,\ \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$
  - Then $z_i^{(k+1)}$ is also normal distributed, with what variance? – $\sigma(z_i^{(k+1)})^2 = n_{\mathrm{node}}^{(k)}$
  - i.e. increased probability for $|z_i^{(k+1)}| \gg 0$.



Here (k) stands for layer k.

Initialization of trainable parameters (TPs)

- Thresholds/biases are usually initialized to 0.

- The initialization of the weights happens randomly following a normal or uniform distribution.

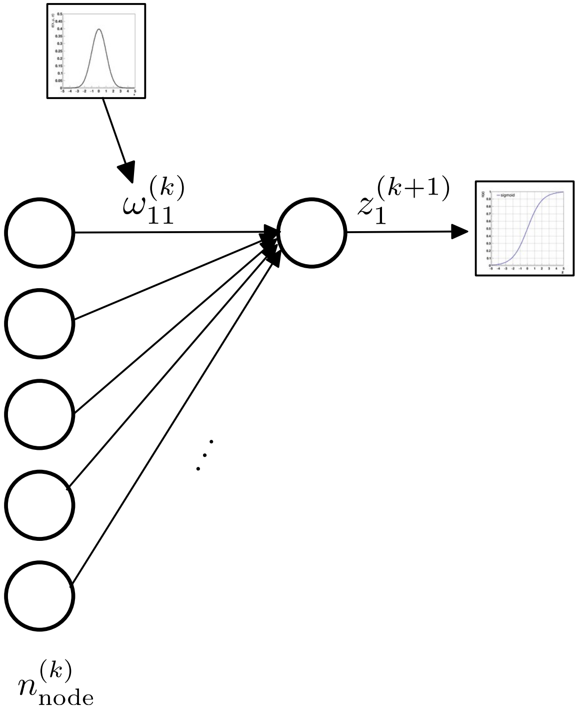
- Naive ansatz:

  - Assume all weights to be initialized as standard normal distributed:
    $\mu(\omega_{ij}^{(k)}) = 0,\, \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$

  - Then $z_i^{(k+1)}$ is also normal distributed, with what variance? – $\sigma(z_i^{(k+1)})^2 = n_{\mathrm{node}}^{(k)}$

  - i.e. increased probability for $|z_i^{(k+1)}| \gg 0$.

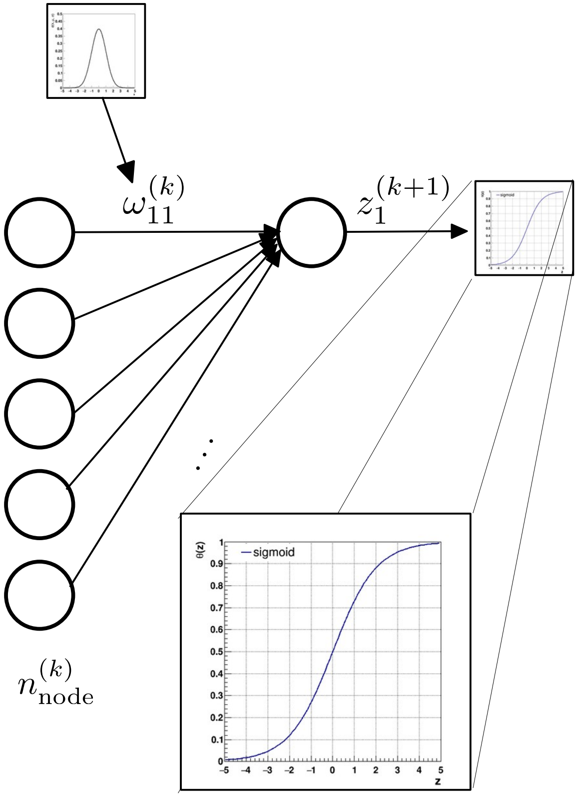
  - What is the consequence for $y_i^{(k+1)}$?



Here (k) stands for layer k.

Initialization of trainable parameters (TPs)

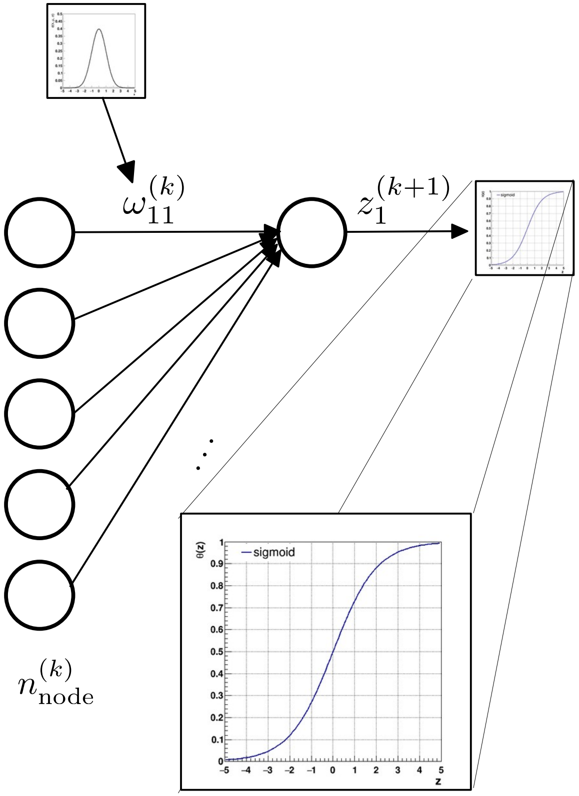
- Thresholds/biases are usually initialized to 0.
- The initialization of the weights happens randomly following a normal or uniform distribution.
- Naive ansatz:
  - Assume all weights to be initialized as standard normal distributed:
    $\mu(\omega_{ij}^{(k)}) = 0,\ \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$
  - Then $z_i^{(k+1)}$ is also normal distributed, with what variance? – $\sigma(z_i^{(k+1)})^2 = n_{\mathrm{node}}^{(k)}$
  - i.e. increased probability for $|z_i^{(k+1)}| \gg 0$.
  - What is the consequence for $y_i^{(k+1)}$? – $y_i^{(k+1)} \to 0, 1$



Here (k) stands for layer k.

Initialization of trainable parameters (TPs)

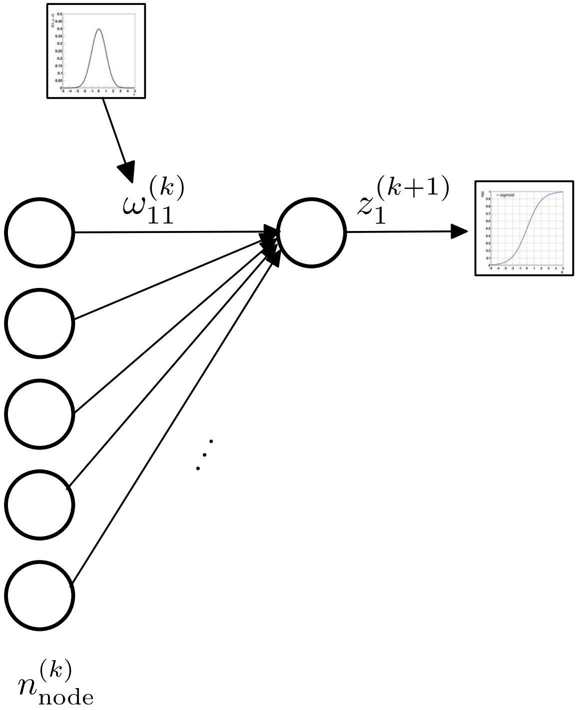
- Thresholds/biases are usually initialized to 0.
- The initialization of the weights happens randomly following a normal or uniform distribution.
- Naive ansatz:
  - Assume all weights to be initialized as standard normal distributed:
    $\mu(\omega_{ij}^{(k)}) = 0,\ \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$
  - Then $z_i^{(k+1)}$ is also normal distributed, with what variance? – $\sigma(z_i^{(k+1)})^2 = n_{\mathrm{node}}^{(k)}$
  - i.e. increased probability for $|z_i^{(k+1)}| \gg 0$.
  - What is the consequence for $y_i^{(k+1)}$? – $y_i^{(k+1)} \to 0, 1$
  - i.e. nodes in subsequent layers will not contribute any more to the information gain.



Here (k) stands for layer k.

Glorot initialization

- This situation can be prevented when initializing the weights in the following way:
  - Initialize weights according to:
    $\mu(\omega_{ij}^{(k)}) = 0,\ \sigma(\omega_{ij}^{(k)}) = 1 \quad \forall\, i,\, j,\, k$
  - Scale all weights according to:
    $$\omega_{ij}^{(k)} \rightarrow \omega_{ij}^{(k)\prime} = \frac{1}{\sqrt{n_{\mathrm{node}}^{(k)}}}\omega_{ij}^{(k)}$$
  - This leads to: $\sigma(z_i^{(k+1)})^2 = 1$
- This method of initialization is called **Glorot** or **Xavier initialization**.



Here (k) stands for layer k.

ML applications of gradient descent

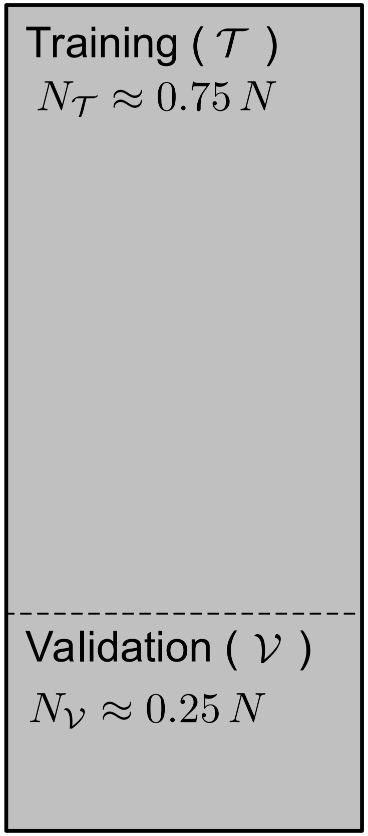
Gradienten decent in practice

- We will discuss three practical flavors of gradient descent (GD):
  - *Batch gradient descent*, (BGD).
  - *Stochastic gradient descent* (SGD).
  - *Mini-batch gradient descent* (mBGD).

Batch gradient descent (BGD)

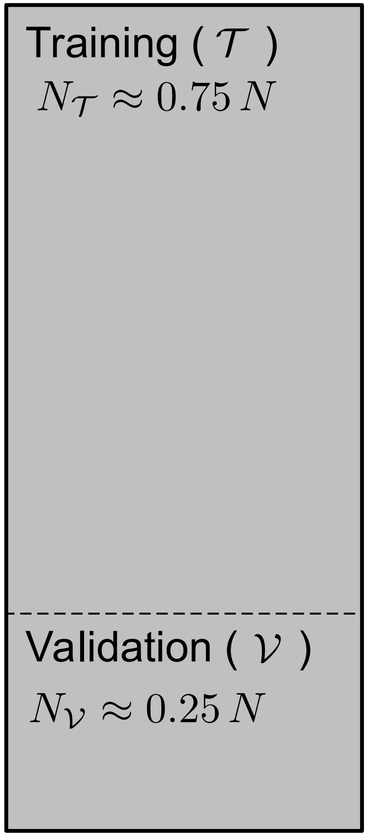
- Evaluate $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ on $\mathcal{T}$ ($N = N_{\mathcal{T}}$).

- After weight actualization validate $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ on $\mathcal{V}$ ($N = N_{\mathcal{V}}$).



Stochastic gradient descent (SGD)

- Evaluate $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ on a **single sample** of $\mathcal{T}$ ($N=1$).
- After evaluation permute $\mathcal{T}$ randomly.
- After weight actualization validate $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ on $\mathcal{V}$ ($N=N_{\mathcal{V}}$).

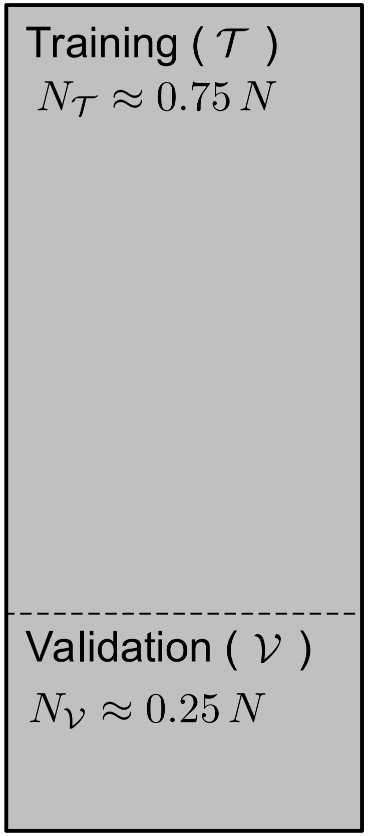


Mini-batch gradient descent (mBGD)

- Evaluate $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ on a **mini-batch** drawn from $\mathcal{T}$ ($N = N_{\mathrm{batch}} < N_{\mathcal{T}}$).

- After evaluation permute $\mathcal{T}$ randomly.

- After weight actualization validate $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ on $\mathcal{V}$ ($N = N_{\mathcal{V}}$).



Discussion of gradient descent

- Each time $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ is evaluated on $\mathcal{V}$ we call **epoch**.

Discussion of gradient descent

- Each time $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ is evaluated on $\mathcal{V}$ we call **epoch**.

- This usually happens after $\mathcal{T}$ could have been sampled (in principle) once completely. But it is also possible to define an epoch by fixed size of gradient descent steps.

Discussion of gradient descent

- Each time $\hat{R}[y(\mathbf{x}^{(\ell)}), \hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})]$ is evaluated on $\mathcal{V}$ we call **epoch**.

- This usually happens after $\mathcal{T}$ could have been sampled (in principle) once completely. But it is also possible to define an epoch by fixed size of gradient descent steps.

- SGD und mBGD are **classical boostrap methods**. **NB**: they can be applied on growing datasets. The nowadays nearly exclusively used method is the mBGD. Batch sizes vary depending on what you can afford hardware-wise.

Challenges during training

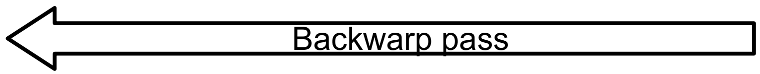
Challenges during training

- We will next discuss the following challenges and how to cope with them during training:
  - Exploding/vanishing gadients.
  - Batch normalization.
  - Generalization property of the NN training.
  - Overtraining and regularization methods.

Exploding/Vanishing gradient

- $\nabla_{\boldsymbol{\omega}^{(k)}} L$ is the product of derivatives of consecutive NN layers (see *backward pass*).

$$\frac{\mathrm{d}L}{\mathrm{d}\omega_{ij}^{(1)}} = \underbrace{\sum_i \frac{\mathrm{d}z_j^{(1)}}{\mathrm{d}\omega_{ij}^{(1)}} \cdot \frac{\mathrm{d}y_1^{(1)}}{\mathrm{d}z_j^{(1)}}}_{\text{Layer 1}} \cdot \underbrace{\sum_i \frac{\mathrm{d}z_j^{(2)}}{\mathrm{d}\omega_{ij}^{(2)}} \cdot \frac{\mathrm{d}y_j^{(2)}}{\mathrm{d}z_j^{(2)}}}_{\text{Layer 2}} \cdot \ldots \cdot \underbrace{\sum_i \frac{\mathrm{d}z_j^{(k)}}{\mathrm{d}\omega_{ij}^{(k)}} \cdot \frac{\mathrm{d}y_i^{(k)}}{\mathrm{d}z_i^{(k)}}}_{\text{Layer } k} \cdot \frac{\mathrm{d}L}{\mathrm{d}y_i^{(k)}}$$



- For an NN with many layers this product can become **very long**!

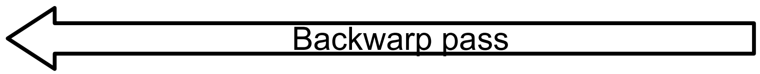
- It may happen that each factor contributes with < 1, i.e. $\nabla_{\boldsymbol{\omega}^{(k)}} L \to 0$, the $\omega_{ij}^{(k)}$ in the first layers are never really updated in such a case ($\to$ **vanishing gradient**).

Here (k) stands for layer k.

Exploding/Vanishing gradient

- $\nabla_{\boldsymbol{\omega}^{(k)}} L$ is the product of derivatives of consecutive NN layers (see *backward pass*).

$$\frac{\mathrm{d}L}{\mathrm{d}\omega_{ij}^{(1)}} = \underbrace{\sum_i \frac{\mathrm{d}z_j^{(1)}}{\mathrm{d}\omega_{ij}^{(1)}} \cdot \frac{\mathrm{d}y_1^{(1)}}{\mathrm{d}z_j^{(1)}}}_{\text{Layer 1}} \cdot \underbrace{\sum_i \frac{\mathrm{d}z_j^{(2)}}{\mathrm{d}\omega_{ij}^{(2)}} \cdot \frac{\mathrm{d}y_j^{(2)}}{\mathrm{d}z_j^{(2)}}}_{\text{Layer 2}} \cdot \ldots \cdot \underbrace{\sum_i \frac{\mathrm{d}z_j^{(k)}}{\mathrm{d}\omega_{ij}^{(k)}} \cdot \frac{\mathrm{d}y_i^{(k)}}{\mathrm{d}z_i^{(k)}}}_{\text{Layer } k} \cdot \frac{\mathrm{d}L}{\mathrm{d}y_i^{(k)}}$$



- For an NN with many layers this product can become **very long**!

- It may happen that each factor contributes with < 1, i.e. $\nabla_{\boldsymbol{\omega}^{(k)}} L \to 0$, the $\omega_{ij}^{(k)}$ in the first layers are never really updated in such a case ($\to$ **vanishing gradient**).

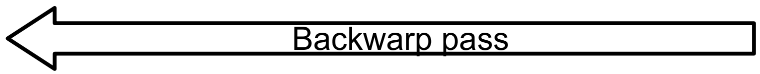
- It may happen that each factor contributes with $\gg 1$, i.e. $\nabla_{\boldsymbol{\omega}^{(k)}} L \to \infty$, we obtain erratic jumps in the updates of the $\omega_{ij}^{(k)}$ ($\to$ **exploding gradient**).

Here (k) stands for layer k.

Exploding/Vanishing gradient

- $\nabla_{\boldsymbol{\omega}^{(k)}} L$ is the product of derivatives of consecutive NN layers (see *backward pass*).

$$\frac{\mathrm{d}L}{\mathrm{d}\omega_{ij}^{(1)}} = \underbrace{\sum_i \frac{\mathrm{d}z_j^{(1)}}{\mathrm{d}\omega_{ij}^{(1)}} \cdot \frac{\mathrm{d}y_1^{(1)}}{\mathrm{d}z_j^{(1)}}}_{\text{Layer 1}} \cdot \underbrace{\sum_i \frac{\mathrm{d}z_j^{(2)}}{\mathrm{d}\omega_{ij}^{(2)}} \cdot \frac{\mathrm{d}y_j^{(2)}}{\mathrm{d}z_j^{(2)}}}_{\text{Layer 2}} \cdot \ldots \cdot \underbrace{\sum_i \frac{\mathrm{d}z_j^{(k)}}{\mathrm{d}\omega_{ij}^{(k)}} \cdot \frac{\mathrm{d}y_i^{(k)}}{\mathrm{d}z_i^{(k)}}}_{\text{Layer } k} \cdot \frac{\mathrm{d}L}{\mathrm{d}y_i^{(k)}}$$



- For an NN with many layers this product can become **very long**!

- It may happen that each factor contributes with < 1, i.e. $\nabla_{\boldsymbol{\omega}^{(k)}} L \to 0$, the $\omega_{ij}^{(k)}$ in the first layers are never really updated in such a case (→ **vanishing gradient**).

- It may happen that each factor contributes with $\gg 1$, i.e. $\nabla_{\boldsymbol{\omega}^{(k)}} L \to \infty$, we obtain erratic jumps in the updates of the $\omega_{ij}^{(k)}$ (→ **exploding gradient**).

- **NB**: this complex is generally discussed as *unstable gradient problem*.

Here (k) stands for layer k.

Initialization and *feature* standardization

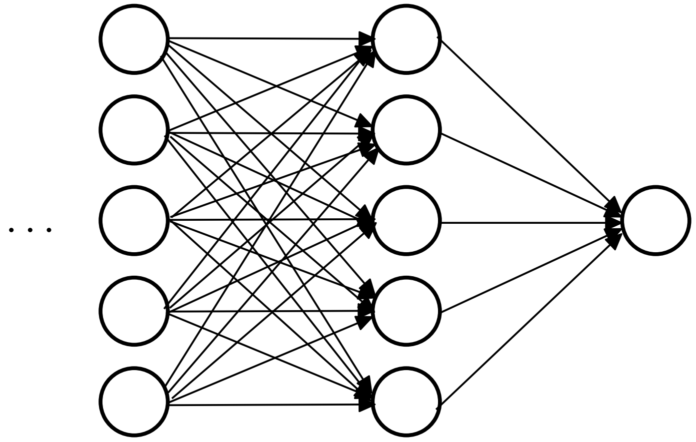
- Two popular ways to address *unstable gradients* are **Glorot initialization** (see slide 10) and **standardization** (of the input features):

$$x_i \to x_i' = \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$$

  - *Input features* with arbitrary potentially strongly varying scales are mapped onto a standard scale.

Batch normalization

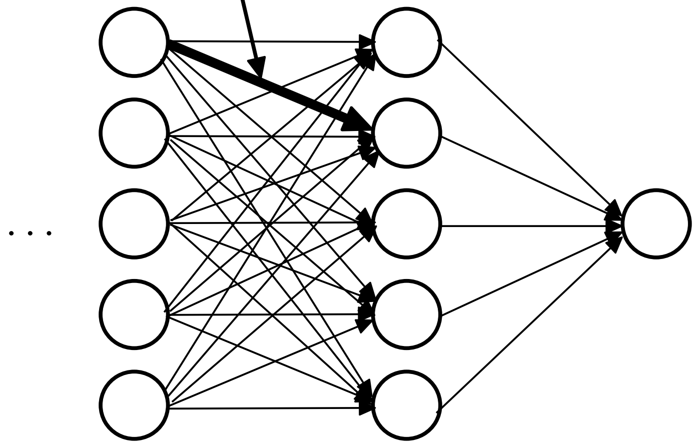
- Standardization of the outputs of individual NN layers (→ **batch normalization**) helps preventing issues with strongly unequalized weights *also during the training*.



Batch normalization

- Standardization of the outputs of individual NN layers (→ **batch normalization**) helps preventing issues with strongly unequalized weights *also during the training*.

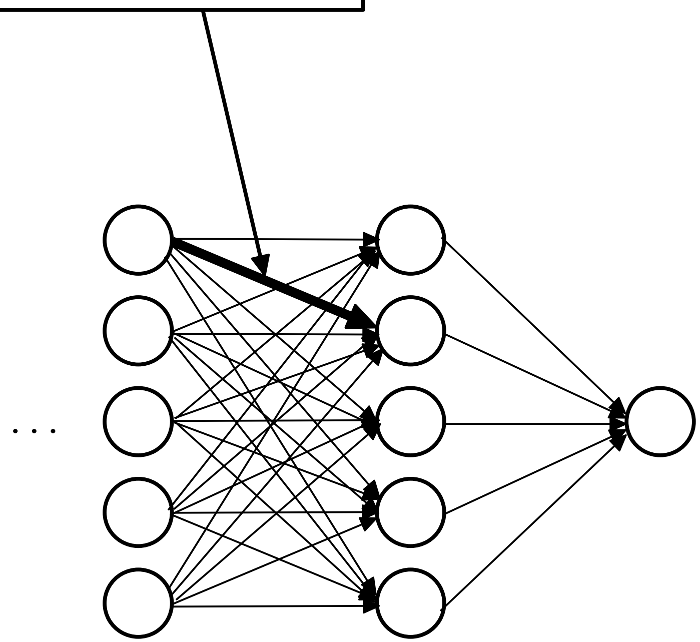
E.g. prevent this weight from becoming much larger than the others.



Batch normalization

- Standardization of the outputs of individual NN layers (→ **batch normalization**) helps preventing issues with strongly unequalized weights *also during the training*.

E.g. prevent this weight from becoming much larger than the others.



- **Procedure**:
  - Standardize outputs of layer (k):

$$y_i^{(k)} \rightarrow y_i^{(k)\prime} = \frac{y_i^{(k)} - \mu_{y_i^{(k)}}}{\sigma_{y_i^{(k)}}}$$

  - Scale and shift the result into an abitrary parameter space:

$$y_i^{(k)\prime} \rightarrow y_i^{(k)\prime\prime} = (g_{ik} \cdot y_i^{(k)\prime}) + b_{ik}$$

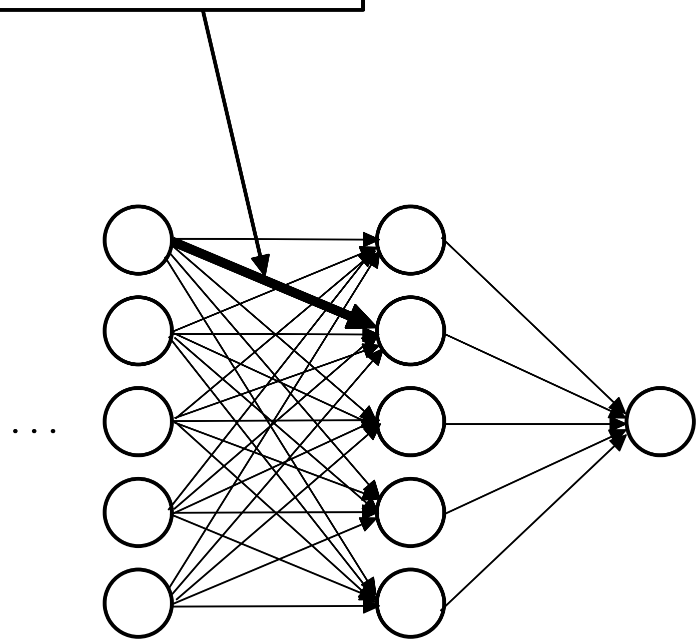
$g_{ik},\, b_{ik}$: abitrary TPs.

g_{ik} and b_{ik} give the NN the possibility to apply the $\{\omega_{ij}^{(k)}\}$ always on the same sub-manifold in parameter space.

Batch normalization

- Standardization of the outputs of individual NN layers (→ **batch normalization**) helps preventing issues with strongly unequalized weights *also during the training*.

E.g. prevent this weight from becoming much larger than the others.



- **Procedure**:
  - Standardize outputs of layer (k):

$$y_i^{(k)} \to y_i^{(k)\prime} = \frac{y_i^{(k)} - \mu_{y_i^{(k)}}}{\sigma_{y_i^{(k)}}}$$

  - Scale and shift the result into an abitrary parameter space:

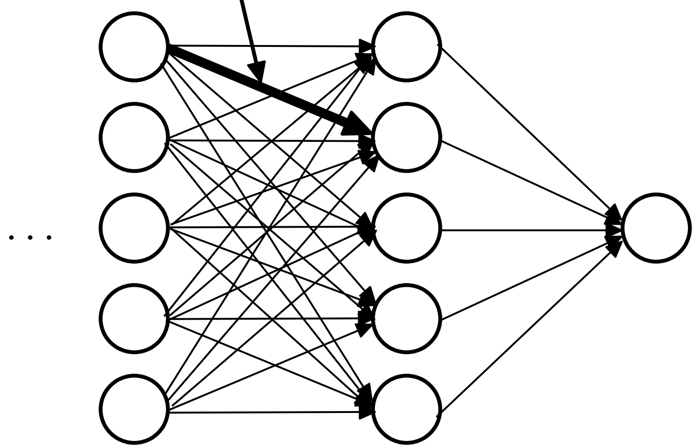
$$y_i^{(k)\prime} \to y_i^{(k)\prime\prime} = (g_{ik} \cdot y_i^{(k)\prime}) + b_{ik}$$
$$g_{ik},\, b_{ik}\text{: abitrary TPs.}$$

  - How would you determine the $\mu_{y_i^{(k)}}$ & $\sigma_{y_i^{(k)}}$?

Batch normalization

- Standardization of the outputs of individual NN layers (→ **batch normalization**) helps preventing issues with strongly unequalized weights <u>*also during the training*</u>.

E.g. prevent this weight from becoming much larger than the others.



- **Procedure**:
  - Standardize outputs of layer (k):

$$y_i^{(k)} \to y_i^{(k)\prime} = \frac{y_i^{(k)} - \mu_{y_i^{(k)}}}{\sigma_{y_i^{(k)}}}$$

  - Scale and shift the result into an abitrary parameter space:

$$y_i^{(k)\prime} \to y_i^{(k)\prime\prime} = (g_{ik} \cdot y_i^{(k)\prime}) + b_{ik}$$

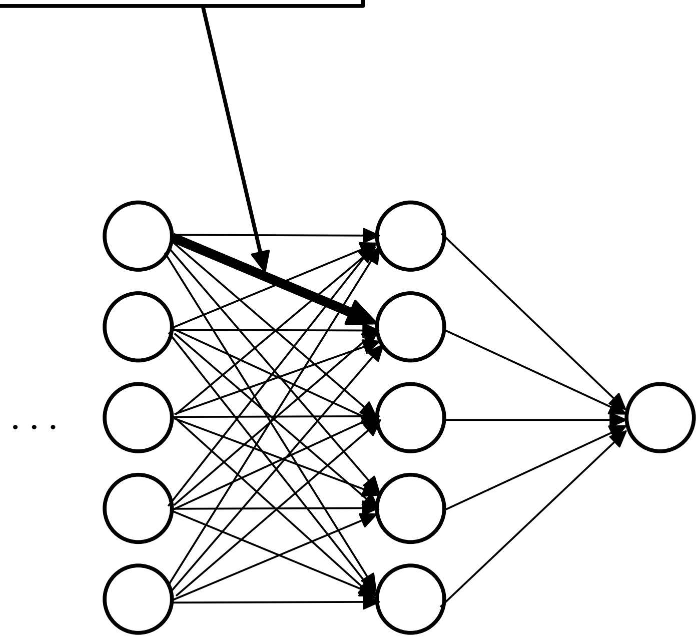
$$g_{ik},\, b_{ik}\text{: abitrary TPs.}$$

  - How would you determine the $\mu_{y_i^{(k)}}$ & $\sigma_{y_i^{(k)}}$? – from the mini-batch during training.

Batch normalization

- Standardization of the outputs of individual NN layers ($\rightarrow$ **batch normalization**) helps preventing issues with strongly unequalized weights *also during the training*.

E.g. prevent this weight from becoming much larger than the others.



- **Procedure**:
  - Standardize outputs of layer (k):

$$y_i^{(k)} \rightarrow y_i^{(k)\prime} = \frac{y_i^{(k)} - \mu_{y_i^{(k)}}}{\sigma_{y_i^{(k)}}}$$

  - Scale and shift the result into an abitrary parameter space:

$$y_i^{(k)\prime} \rightarrow y_i^{(k)\prime\prime} = (g_{ik} \cdot y_i^{(k)\prime}) + b_{ik}$$

$g_{ik},\, b_{ik}$: abitrary TPs.

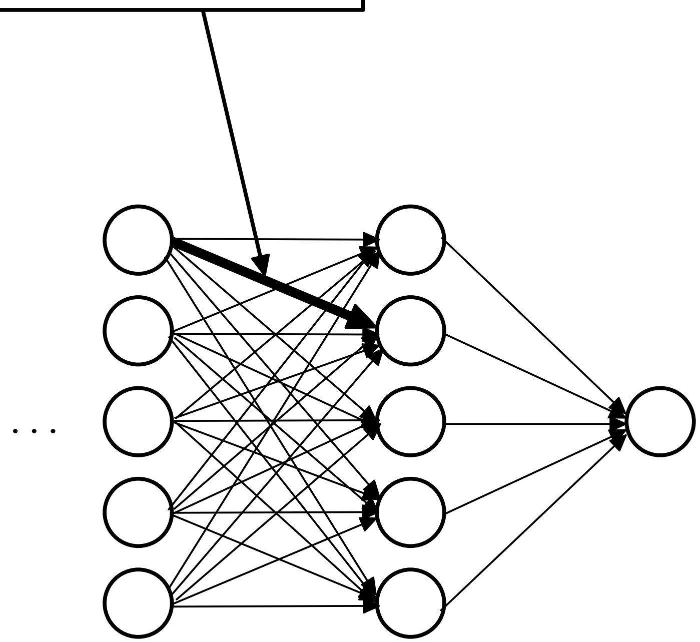
  - How would you determine the $\mu_{y_i^{(k)}}$ & $\sigma_{y_i^{(k)}}$? – from the mini-batch during training.

  - How would you determine the $\mu_{y_i^{(k)}}$ & $\sigma_{y_i^{(k)}}$ when applying the NN to the test data?

Batch normalization

- Standardization of the outputs of individual NN layers (→ **batch normalization**) helps preventing issues with strongly unequalized weights *also during the training*.

E.g. prevent this weight from becoming much larger than the others.



- **Procedure**:
  - Standardize outputs of layer (k):

$$y_i^{(k)} \rightarrow y_i^{(k)\prime} = \frac{y_i^{(k)} - \mu_{y_i^{(k)}}}{\sigma_{y_i^{(k)}}}$$

  - Scale and shift the result into an abitrary parameter space:

$$y_i^{(k)\prime} \rightarrow y_i^{(k)\prime\prime} = (g_{ik} \cdot y_i^{(k)\prime}) + b_{ik}$$

$$g_{ik},\, b_{ik}\text{: abitrary TPs.}$$

  - How would you determine the $\mu_{y_i^{(k)}}$ & $\sigma_{y_i^{(k)}}$? – from the mini-batch during training.
  - How would you determine the $\mu_{y_i^{(k)}}$ & $\sigma_{y_i^{(k)}}$ when applying the NN to the test data? – use the values calculated on all $\mathcal{T}$.

What does the MLP learn?!?

- **Reminder**: In general we have to assume that the underlying truth to a classification or regression problem is unknown.

What does the MLP learn?!?

- **Reminder**: In general we have to assume that the underlying truth to a classification or regression problem is unknown.

- Fundamental issue **generalization**:

What does the MLP learn?!?

- **Reminder**: In general we have to assume that the underlying truth to a classification or regression problem is unknown.

- Fundamental issue **generalization**:

  - Truth unknown, the training data consist only of samples from the ground truth.

What does the MLP learn?!?

- **Reminder**: In general we have to assume that the underlying truth to a classification or regression problem is unknown.

- Fundamental issue **generalization**:
  - Truth unknown, the training data consist only of samples from the ground truth.
  - The sample is subject to fluctuations (→ variance).

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

What does the MLP learn?!?

- **Reminder**: In general we have to assume that the underlying truth to a classification or regression problem is unknown.

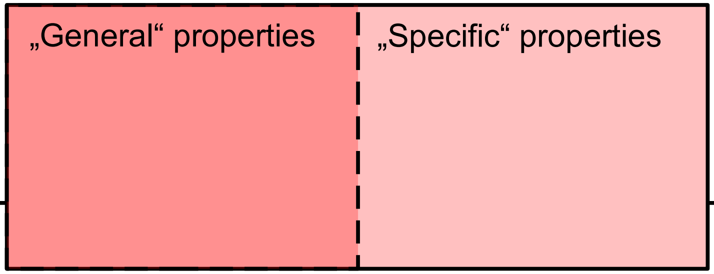
- Fundamental issue **generalization**:
  - Truth unknown, the training data consist only of samples from the ground truth.
  - The sample is subject to fluctuations (→ variance).
  - In addition training data could look fundamentally different from test data (→ bias).

What does the MLP learn?!?

- **Reminder**: In general we have to assume that the underlying truth to a classification or regression problem is unknown.

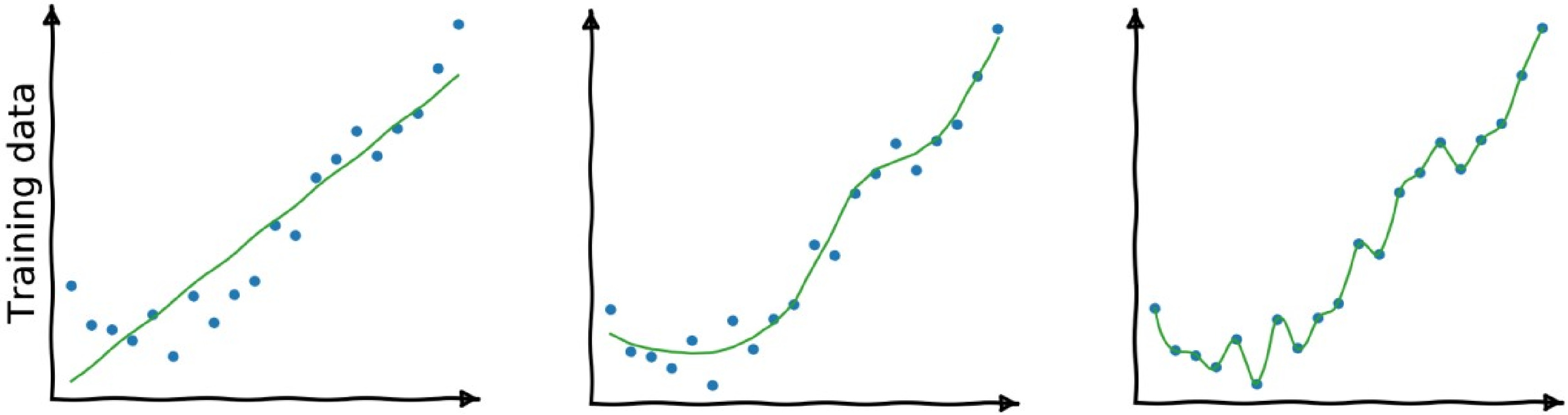
- Fundamental issue **generalization**:
  - Truth unknown, the training data consist only of samples from the ground truth.
  - The sample is subject to fluctuations (→ variance).
  - In addition training data could look fundamentally different from test data (→ bias).

Training dataset:

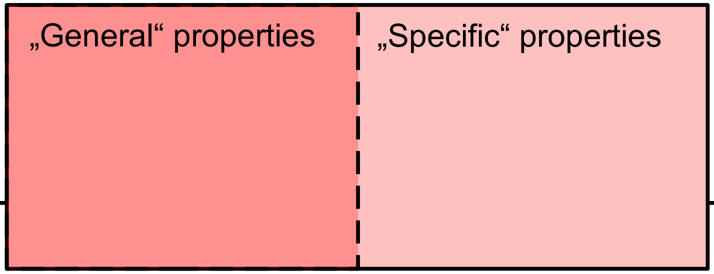


Identification of general properties

- (How) can one distinguish general from specific properties of the training dataset?
- **Example**: the training dataset is indicated by the blue points.



Training dataset:



Obvious connection to the issue of **overfitting** → overtraining.

Training and validation

- The issue of generalization of the NN model after training, nowadays is addressed through the use of the validation dataset $\mathcal{V}$ (see slide 5):

Training and validation

- The issue of generalization of the NN model after training, nowadays is addressed through the use of the validation dataset $\mathcal{V}$ (see slide 5):

- If the NN model mostly describes unbiased general properties of the ground truth, we can expect that it will also describe $\mathcal{V}$ „reasonably well“.

Training and validation

- The issue of generalization of the NN model after training, nowadays is addressed through the use of the validation dataset $\mathcal{V}$ (see slide 5):

- If the NN model mostly describes unbiased general properties of the ground truth, we can expect that it will also describe $\mathcal{V}$ „reasonably well“.

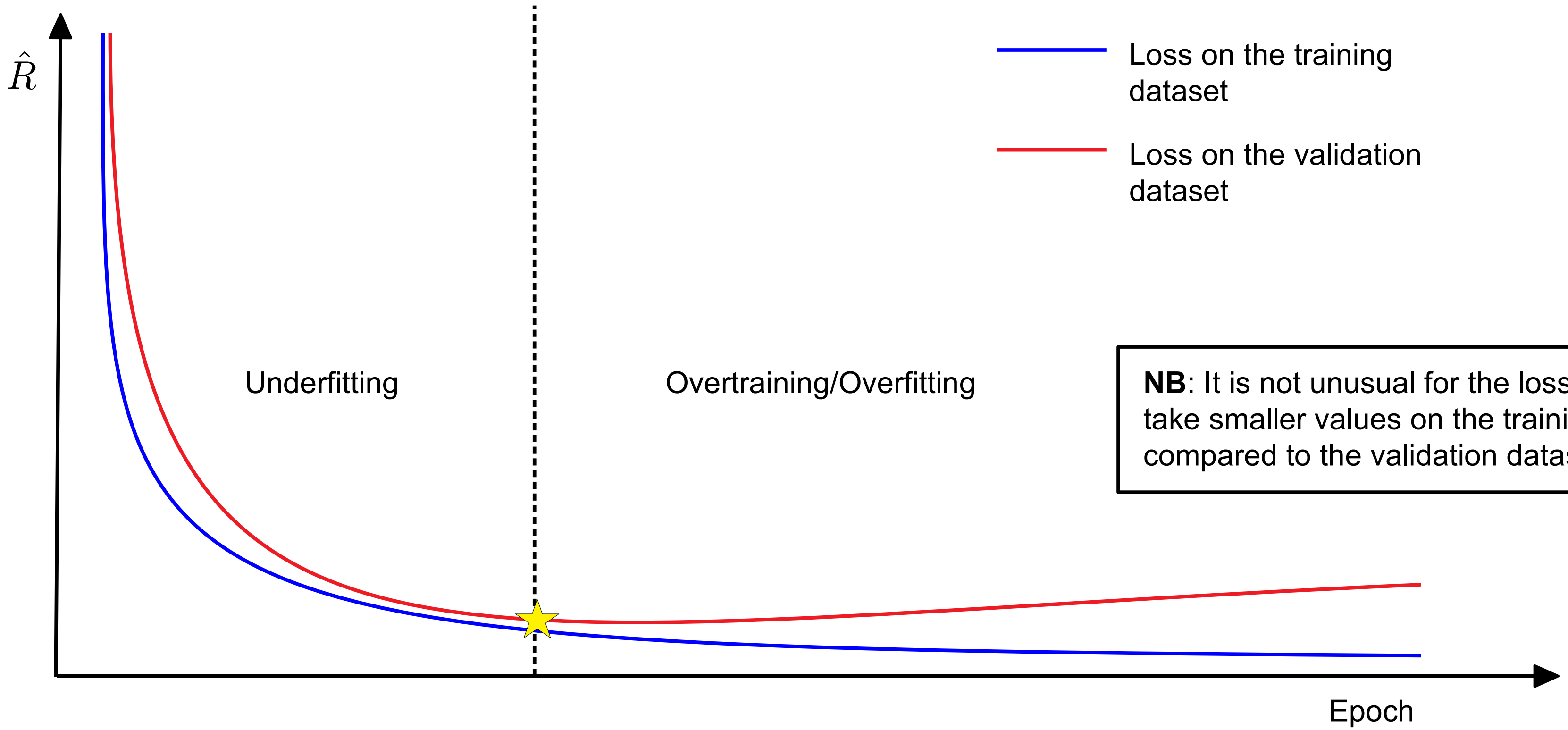
- An obvious way to check the consistency of the training is via the empirical risk function and thus the training objective itself. But it‘s not the only way…

Training and validation

- The issue of generalization of the NN model after training, nowadays is addressed through the use of the validation dataset $\mathcal{V}$ (see slide 5):
- If the NN model mostly describes unbiased general properties of the ground truth, we can expect that it will also describe $\mathcal{V}$ „reasonably well".
- An obvious way to check the consistency of the training is via the empirical risk function and thus the training objective itself. But it's not the only way…
- **NB**: In the past people evaluated $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ on the training and validation datasets and quantified their consistency with help of a Kolmogorow-Smirnow test.

Learning curve

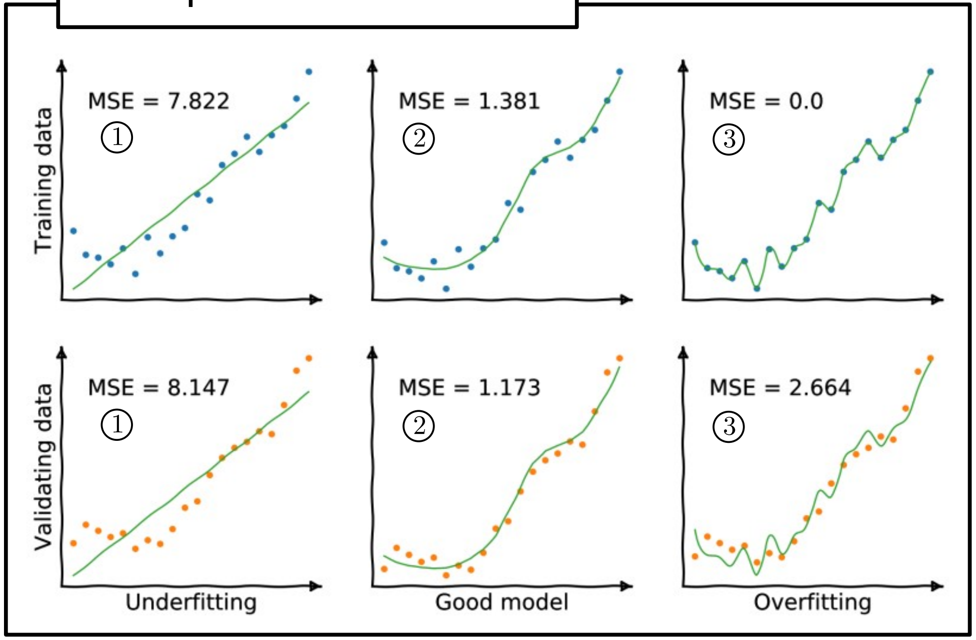
- Typically the risk function drops with increasing number of epochs on the training dataset.
- On the validation dataset the risk function will (mildly) increase again after a certain amount of epochs.

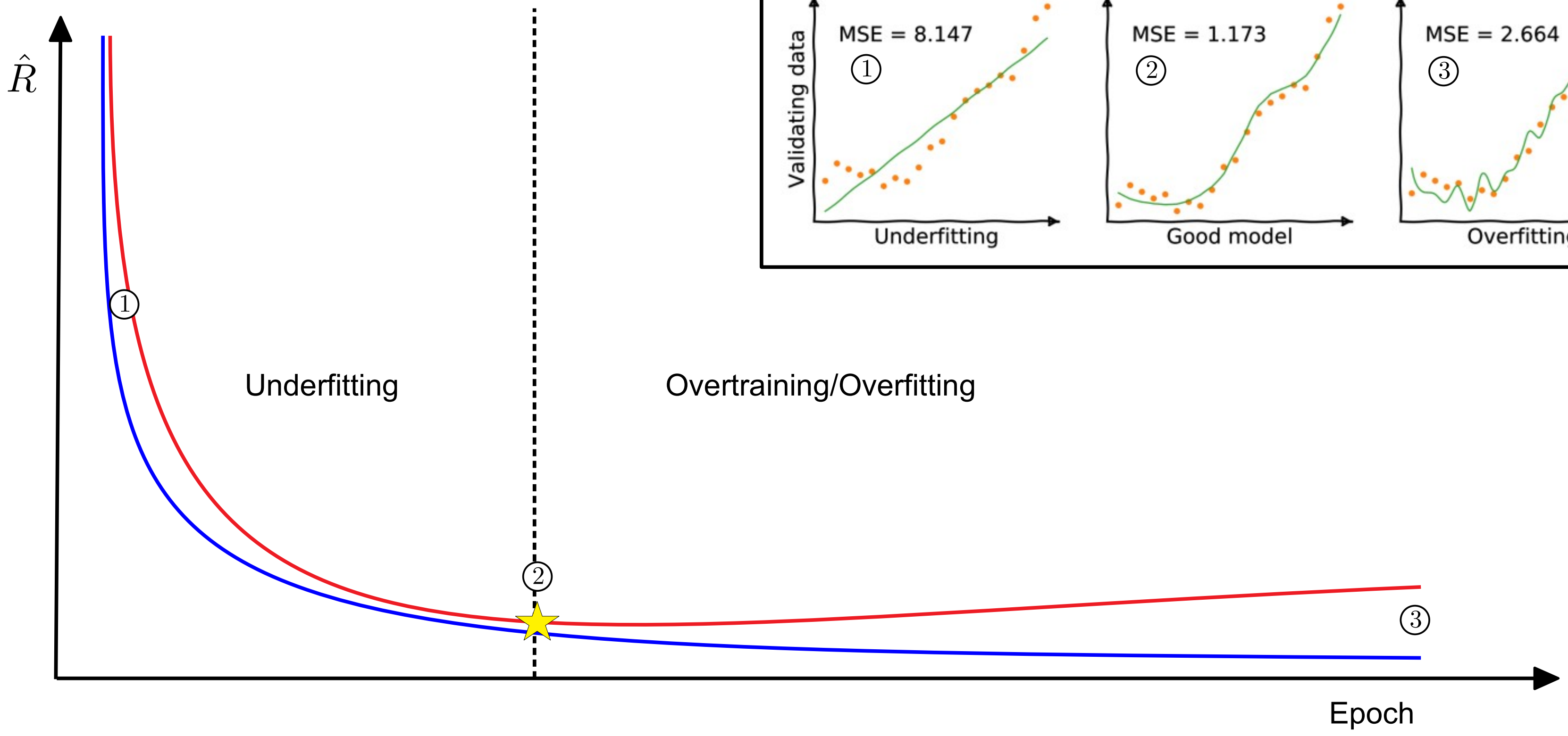


NB: It is not unusual for the loss to take smaller values on the training compared to the validation dataset.

Learning curve

Example of slide 28:





Early stopping

- These thoughts motivate a simple but very effective strategy to guarantee a sufficient level of generalization of the NN model:

Early stopping

- These thoughts motivate a simple but very effective strategy to guarantee a sufficient level of generalization of the NN model:
  - Evaluate $\hat{R}$ after each epoch on $\mathcal{V}$.

Early stopping

- These thoughts motivate a simple but very effective strategy to guarantee a sufficient level of generalization of the NN model:
  - Evaluate $\hat{R}$ after each epoch on $\mathcal{V}$.
  - If $\hat{R}$ evaluated on $\mathcal{V}$ does not decrease any more after a certain *latency*, stop the training.

Early stopping

- These thoughts motivate a simple but very effective strategy to guarantee a sufficient level of generalization of the NN model:
  - Evaluate $\hat{R}$ after each epoch on $\mathcal{V}$.
  - If $\hat{R}$ evaluated on $\mathcal{V}$ does not decrease any more after a certain *latency*, stop the training.
  - Such a procedure is called **early stopping**. Here it is described in it‘s simplest form.

Discussion of generalization

- Since the optimization of the NN model is based on samples, the minimum that is reached on the training dataset can maximally be consistent with the minimum on the validation dataset, i.e. the expectation values of the estimates on both datasets coindice within their variances.

Discussion of generalization

- Since the optimization of the NN model is based on samples, the minimum that is reached on the training dataset can maximally be consistent with the minimum on the validation dataset, i.e. the expectation values of the estimates on both datasets coindice within their variances.

- If a training setup reaches consistency this confirms a good generalization property of the NN model.

Discussion of generalization

- Since the optimization of the NN model is based on samples, the minimum that is reached on the training dataset can maximally be consistent with the minimum on the validation dataset, i.e. the expectation values of the estimates on both datasets coindice within their variances.

- If a training setup reaches consistency this confirms a good generalization property of the NN model.

- If the NN has bad generalization properties it is *in the worst case …?*

Discussion of generalization

- Since the optimization of the NN model is based on samples, the minimum that is reached on the training dataset can maximally be consistent with the minimum on the validation dataset, i.e. the expectation values of the estimates on both datasets coindice within their variances.

- If a training setup reaches consistency this confirms a good generalization property of the NN model.

- If the NN has bad generalization properties it is *in the worst case useless!*

Unfolding vs. NN generalization

- The discussion of generalization in ML has obvious correspondences to the „inverse problem“ of unfolding:

Unfolding:		**Machine Learning**:
Truth to be unfolded		Ground truth to be approximated
Unfolding matrix	⇔	Training dataset
Unfolding		NN model after training

Unfolding vs. NN generalization

- The discussion of generalization in ML has obvious correspondences to the „inverse problem“ of unfolding:

Unfolding:		**Machine Learning**:
Truth to be unfolded	⇔	Ground truth to be approximated
Unfolding matrix	⇔	Training dataset
Unfolding	⇔	NN model after training

NB: A very modern NN architecture, the Normalizing Flow makes this relation explicit.

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Unfolding vs. NN generalization

- The discussion of generalization in ML has obvious correspondences to the „inverse problem“ of unfolding:

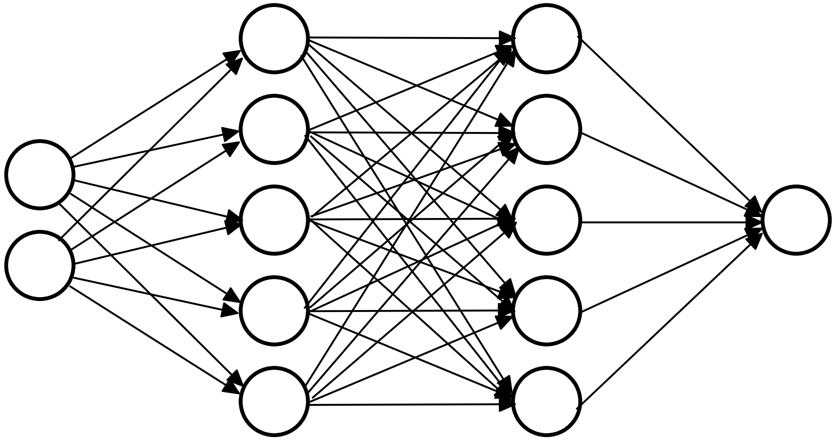
Unfolding:		**Machine Learning**:
Truth to be unfolded		Ground truth to be approximated
Unfolding matrix	⇔	Training dataset
Unfolding		NN model after training

- As in the case of unfolding **regularization** measures help improving the congruence of the model with the ground truth.

Dropout

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

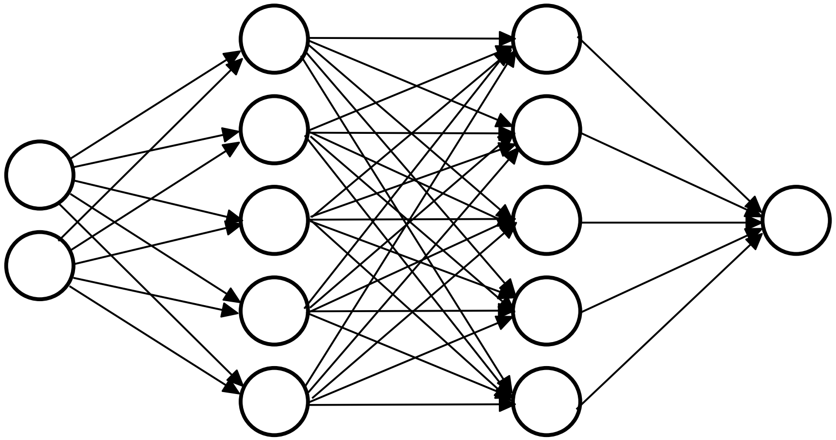
- A simple realization is the so-called (*inverted*) **dropout**:



Dropout

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

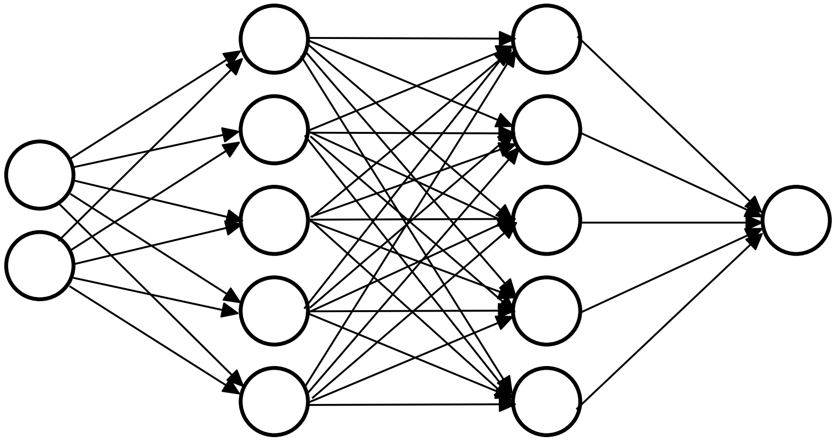
- A simple realization is the so-called (*inverted*) **dropout**:
  - Before each gradient descent, randomly earse nodes and all related connections with a given probability d (including input nodes).



Dropout

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

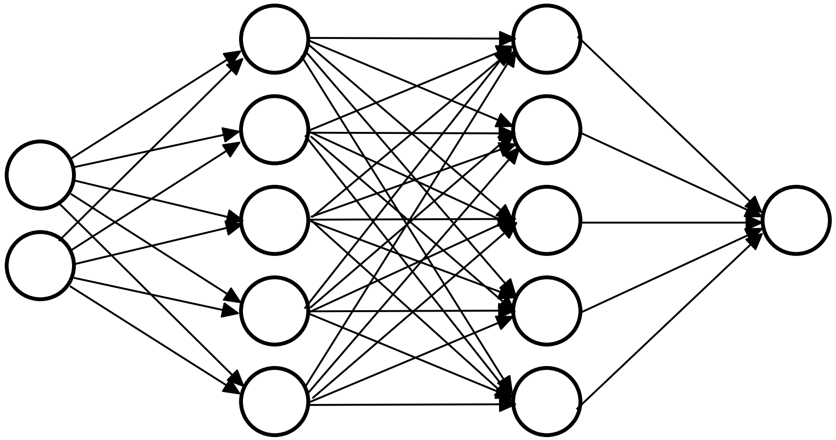
- A simple realization is the so-called (*inverted*) **dropout**:
  - Before each gradient descent, randomly earse nodes and all related connections with a given probability d (including input nodes).
  - This will create a gradient descent step for a slightly varying NN architecture.



Dropout

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

- A simple realization is the so-called (*inverted*) **dropout**:
  - Before each gradient descent, randomly earse nodes and all related connections with a given probability d (including input nodes).
  - This will create a gradient descent step for a slightly varying NN architecture.
  - Rescale all remaining weigths by $1/(1-d)$ Why?



Dropout

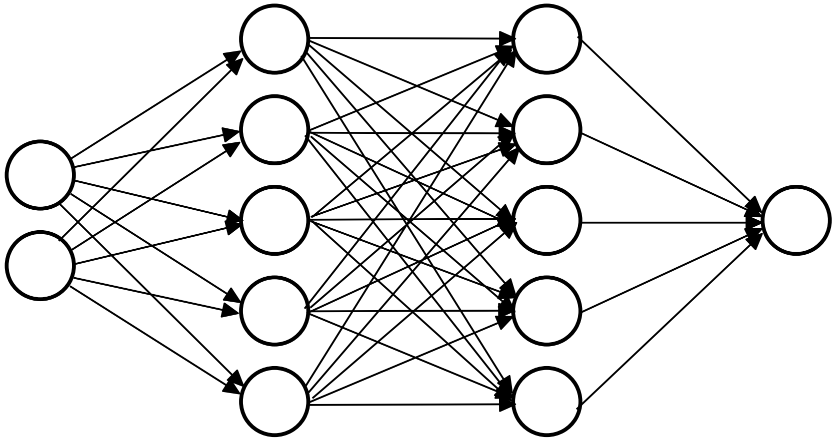
- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

- A simple realization is the so-called (*inverted*) **dropout**:

  - Before each gradient descent, randomly earse nodes and all related connections with a given probability d (including input nodes).

  - This will create a gradient descent step for a slightly varying NN architecture.

  - Rescale all remaining weigths by $1/(1-d)$ Why? – Imagine you erased a fraction d of nodes. The mean inputs to the next layer (k) would than drop to:



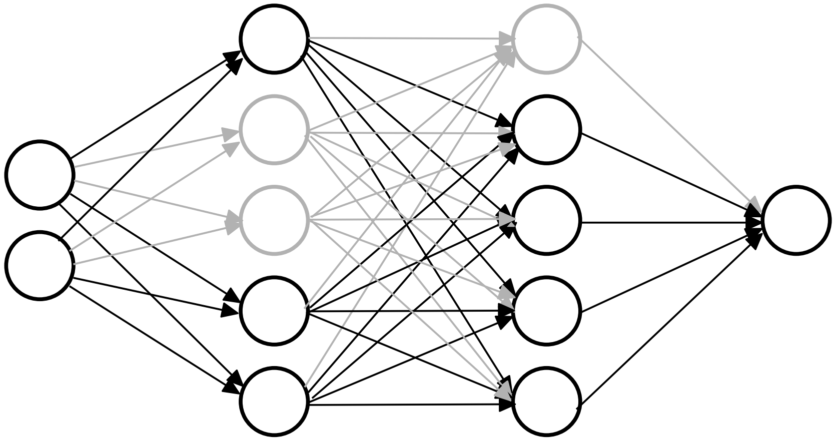
$$z_j^{(k)} = \sum_i y_i^{(k-1)} w_{ij}^{(k)}; \quad \langle z_j^{(k)} \rangle \Big|_d = (1-d)\langle z_j^{(k)} \rangle$$

Dropout – Example

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

- A simple realization is the so-called (*inverted*) **dropout**:

Dropout:



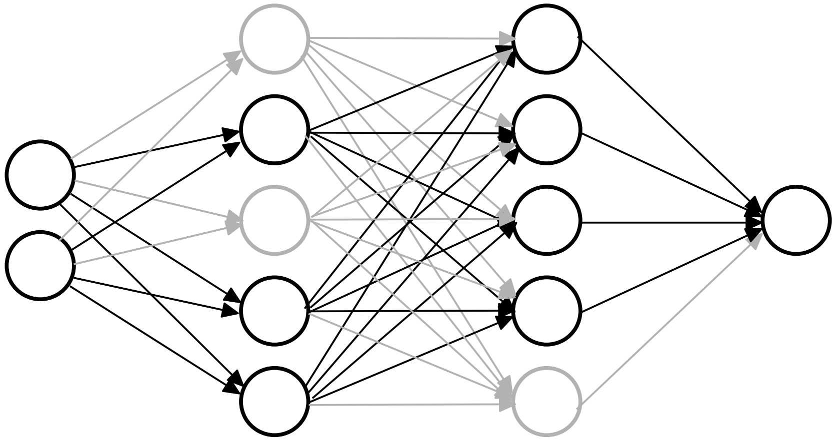
$d = 0.3$

Dropout – Example

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

- A simple realization is the so-called (*inverted*) **dropout**:

Dropout:



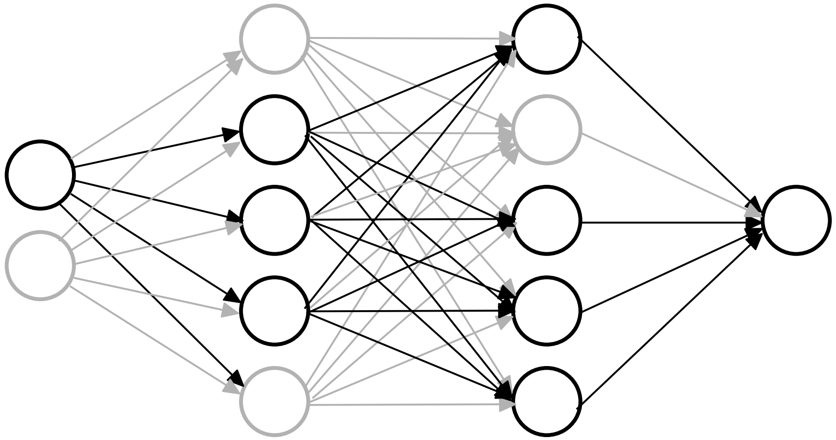
$d = 0.3$

Dropout – Example

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

- A simple realization is the so-called (*inverted*) **dropout**:

Dropout:



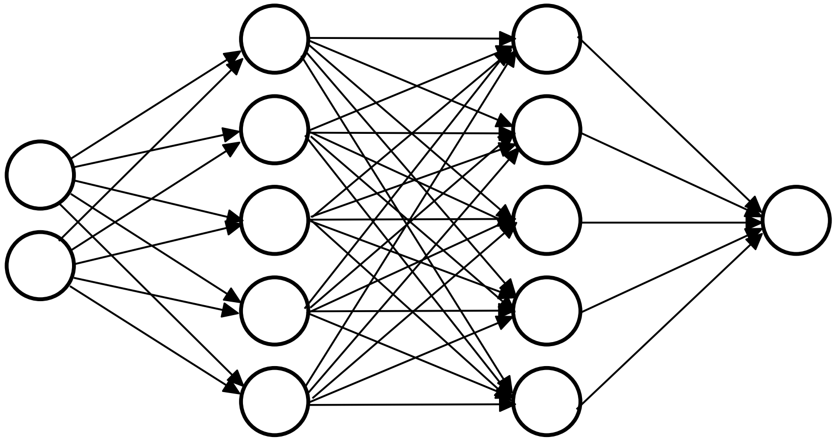
$d = 0.3$

Dropout – Example

- It has been shown to have a regularizing effect to train an ensemble with varying NN architectures and to average over the results within this ensemble.

- A simple realization is the so-called (*inverted*) **dropout**:

Dropout:

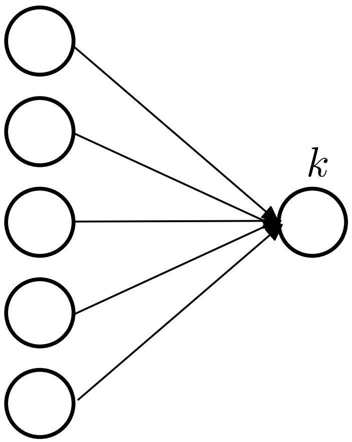


$d = 0.3$

Usual choices of dropout probabilities are $d = 0.3 \ldots 0.5$.

How and why does Dropout work?

- How can dropout regularize the weights of an NN?



- Here node k obtains its information from 5 predecessor nodes.
- Each predecessor node could be erased during the next gradient descent step.
- The decision of node k may not rely on the information of a single predecessor node. The relevant information must be distributed over as many predecessors as possible.

- This leads to a more equalized distribution of weights.
- It can be shown that dropout is equvalent to an L2 regularization, where the parameter λ is determined in each node individually.

L1 and L2 regularization

- The last form of regularization that we will discuss today is L1 and/or L2 regularization.
- This should be known to you from the discussion of optimization tasks with boundary conditions, when implemented in the form of penalty terms.
- Here you simply add the sum of all weights in form of the **L1/L2 norm** to the loss function:

$$L(\{y_j^{(\ell)}\},\{\hat{y}_j^{(\ell)}\}) = -\sum_{j=1}^{n} y_j^{(\ell)} \log\left(\hat{y}_j^{(\ell)}\right) + \lambda\,\|\boldsymbol{\omega}\|_{L1/2}$$

n : No. of categories

$\hat{y}_j^{(\ell)}$: NN prediction for sample (ℓ)

$$\|\boldsymbol{\omega}\|_{L1} = \sum_{\alpha} |\omega_\alpha|$$ (*least absolute shrinkage and selection operator,* ***Lasso***)

$$\|\boldsymbol{\omega}\|_{L2} = \sqrt{\sum_{\alpha} \omega_\alpha^2}$$ (***ridge*** *regularization*)

L1 regularization

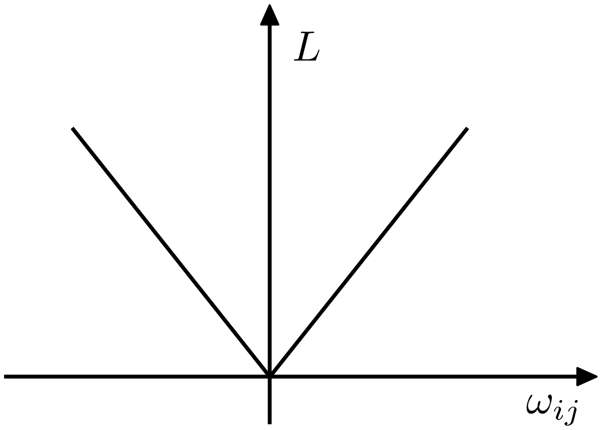
- Derivative of L1-Norm:

$$\frac{\mathrm{d}\|\boldsymbol{\omega}\|_{L1}}{\mathrm{d}\omega_{ij}} = \begin{cases} +1 & \omega_{ij} > 0 \\ -1 & \omega_{ij} < 0 \end{cases}$$

$$\omega_{ij}^{(k+1)} = \omega_{ij}^{(k)} - \eta\,\partial_{\omega_{ij}^{(k)}} L - \mathrm{sgn}(\omega_{ij})\,\lambda, \quad \eta > 0$$

as long as:

$$\left| L(\boldsymbol{\omega}^{(k)}) - L(\boldsymbol{\omega}^{(k-1)}) \right| > \epsilon$$



- **Erase** single weights.

L2 regularization

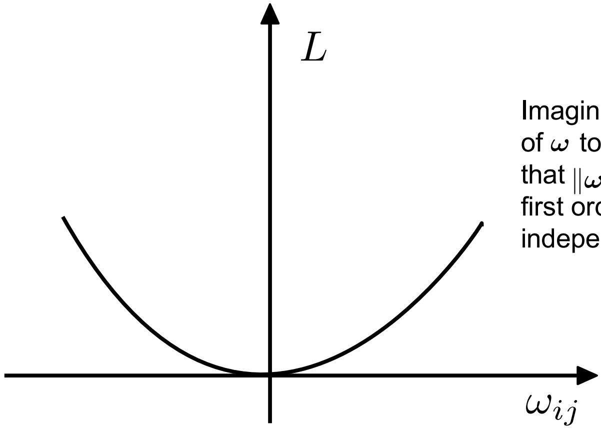
- Derivative L2-Norm:

$$\frac{\mathrm{d}\|\boldsymbol{\omega}\|_{L2}}{\mathrm{d}\omega_{ij}} = \frac{\omega_{ij}}{\|\boldsymbol{\omega}\|_{L2}}$$

$$\omega_{ij}^{(k+1)} = \omega_{ij}^{(k)} - \eta\,\partial_{\omega_{ij}^{(k)}} L - \lambda\,\frac{\omega_{ij}}{\|\boldsymbol{\omega}\|_{L2}}, \quad \eta > 0$$

as long as:

$$\left| L(\boldsymbol{\omega}^{(k)}) - L(\boldsymbol{\omega}^{(k-1)}) \right| > \epsilon$$



Imagine the length of ω to be large, so that $\|\omega\|_{L2}$ will, to first order, be independent of ω_{ij}.

- **Reduce contributions** from individual weights.

L1 regularization

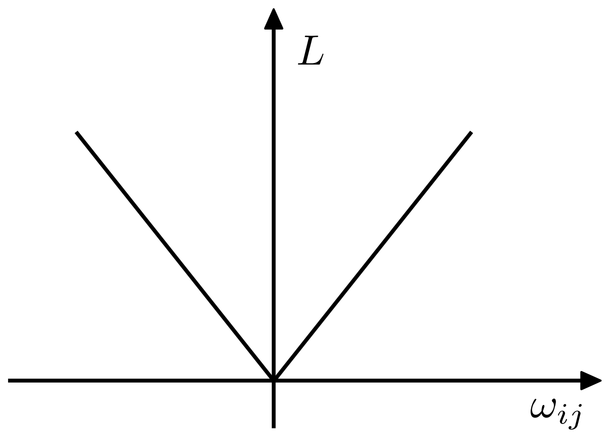
- Derivative of L1-Norm:

$$\frac{\mathrm{d}\|\boldsymbol{\omega}\|_{L1}}{\mathrm{d}\omega_{ij}} = \begin{cases} +1 & \omega_{ij} > 0 \\ -1 & \omega_{ij} < 0 \end{cases}$$

$$\omega_{ij}^{(k+1)} = \omega_{ij}^{(k)} - \eta\,\partial_{\omega_{ij}^{(k)}} L - \mathrm{sgn}(\omega_{ij})\,\lambda, \quad \eta > 0$$

as long as:

$$\left| L(\boldsymbol{\omega}^{(k)}) - L(\boldsymbol{\omega}^{(k-1)}) \right| > \epsilon$$



- **Erase** single weights.

L2 regularization

- Derivative L2-Norm:

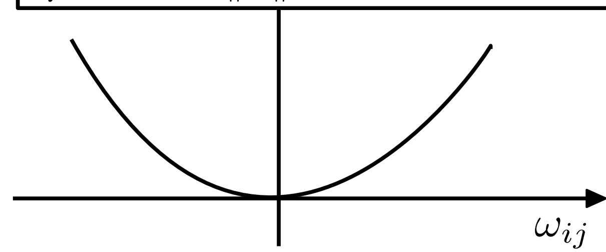
$$\frac{\mathrm{d}\|\boldsymbol{\omega}\|_{L2}}{\mathrm{d}\omega_{ij}} = \frac{\omega_{ij}}{\|\boldsymbol{\omega}\|_{L2}}$$

$$\omega_{ij}^{(k+1)} = \omega_{ij}^{(k)} - \eta\,\partial_{\omega_{ij}^{(k)}} L - \lambda\,\frac{\omega_{ij}}{\|\boldsymbol{\omega}\|_{L2}}, \quad \eta > 0$$

as long as:

$$\left| L(\boldsymbol{\omega}^{(k)}) - L(\boldsymbol{\omega}^{(k-1)}) \right| > \epsilon$$

NB: if the loss function is an NLL function the L2 norm is equivalent to a constraint on the weights based on a multivariate normal distribution with $\boldsymbol{\mu} = 0$ and $\|\boldsymbol{\sigma}\|_{L2} = \lambda$.



- **Reduce contributions** from individual weights.

Discussion of regularization techniques

- In general the following statements hold:
  - The more TPs the higher the risk to overtrain.
  - The larger the training dataset the smaller the risk to overtrain.
  - It is therefore also always possible to reduce the risk of overtraining by increasing the training dataset.
- A procedure that we have not discussed here, since it is irrelevant in particle physics is called *data augmentation*: there one artificially increases the training dataset by turning, stretching, mirroring individual samples of the training dataset.

Success after training

- The success of a training in solving a given task is evaluated comparing the predictions $\hat{y}_j^{(\ell)}$ with the labels $y_j^{(\ell)}$ on $\mathcal{V}$.

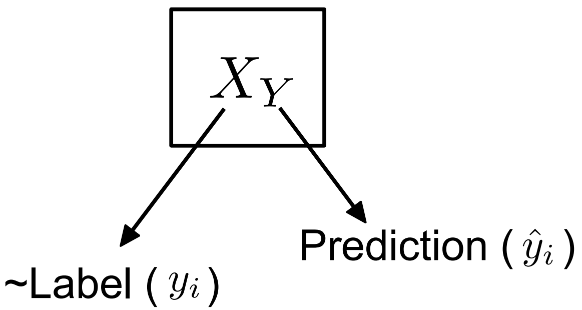
- Does the prediction coincide with the truth label „sufficiently“ often the training was successful in solving the task.

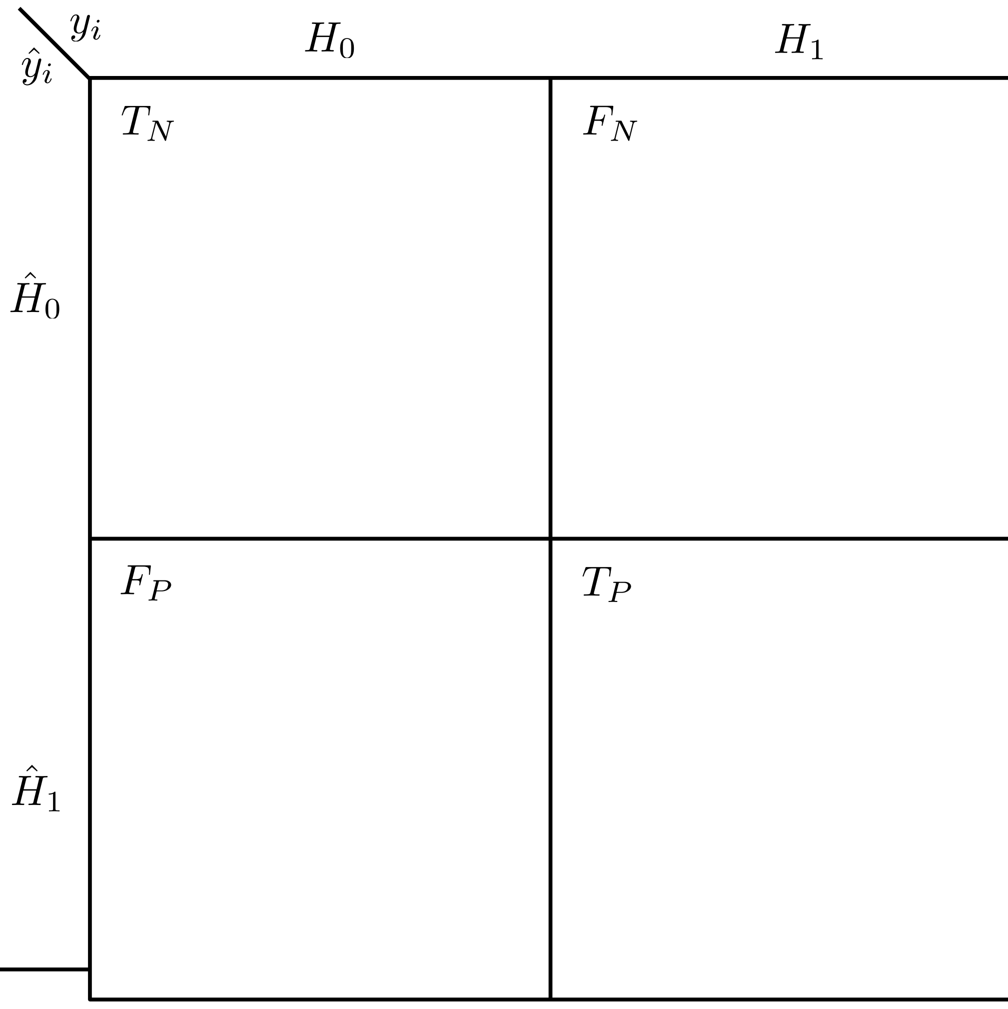
Binary classification

- For the special case of binary classification this assessment can be reduced to the discussion of <u>*binary hypothesis tests*</u>:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund



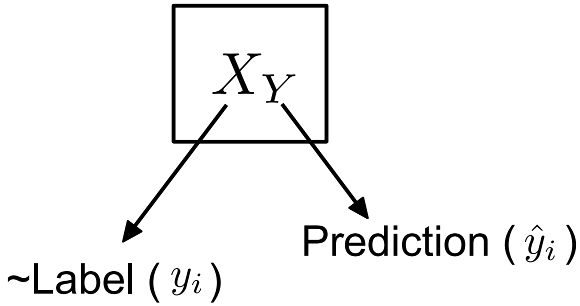


H_0 is true (→ no singal)

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund



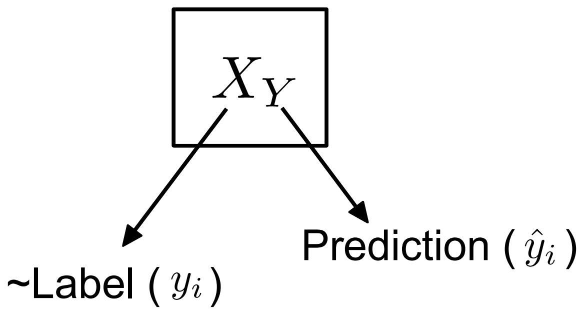
$\hat{y}_i$ \ y_i	H_0	H_1
$\hat{H}_0$	T_N $t_N = \frac{T_N}{F_P + T_N}$ • Specifity • **True negative rate (TNR)**	F_N
$\hat{H}_1$	F_P	T_P

H_0 is true ($\rightarrow$ no singal)

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund

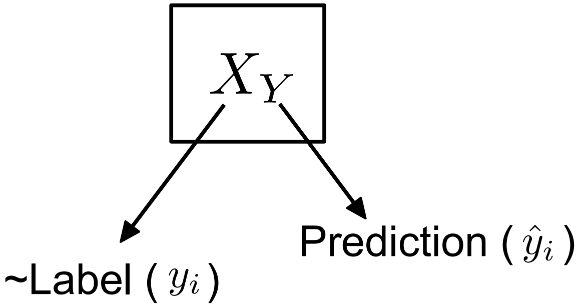


$\hat{y}_i$ \ y_i	H_0	H_1
$\hat{H}_0$	T_N	F_N
$\hat{H}_1$	F_P $f_P = \frac{F_P}{F_P + T_N}$ • Fallout • **False positive rate (FPR)**	T_P

H_1 is true ($\rightarrow$ signal)

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal
H_0 : Alternative hypothesis, Untergrund



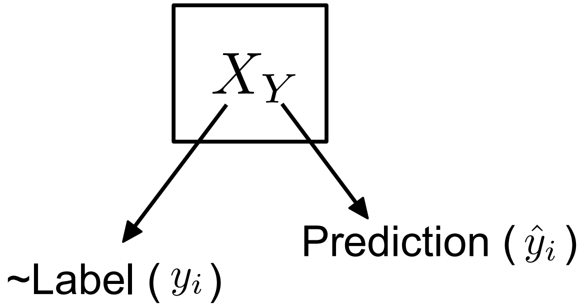
$\hat{y}_i$ \ y_i	H_0	H_1
$\hat{H}_0$	T_N	F_N $f_N = \frac{F_N}{T_P + F_N}$ • Miss rate • **False negative rate (FNR)**
$\hat{H}_1$	F_P	T_P

H_1 is true ($\rightarrow$ signal)

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund

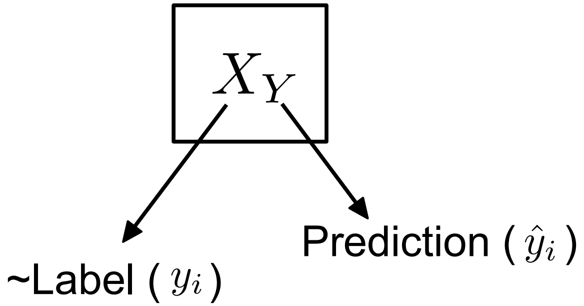


$\hat{y}_i$ \ y_i	H_0	H_1
$\hat{H}_0$	T_N	F_N
$\hat{H}_1$	F_P	T_P $t_P = \frac{T_P}{T_P + F_N}$ • Sensitivity • Recall, hit rate • **True positive rate (TPR)**

$\hat{H}_0$ has been classified

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal
H_0 : Alternative hypothesis, Untergrund



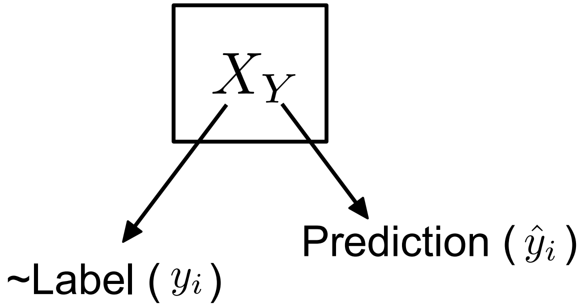
$\hat{y}_i$ \ y_i	H_0	H_1
$\hat{H}_0$	T_N $\tau_N = \frac{T_N}{F_N + T_N}$ • **Negative predictive value (NPV)**	F_N
$\hat{H}_1$	F_P	T_P

$\hat{H}_1$ has been classified

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund



$\hat{y}_i$ \ y_i	H_0	H_1
$\hat{H}_0$	T_N	F_N
$\hat{H}_1$	F_P	T_P $\tau_P = \frac{T_P}{T_P + F_P}$ • Relevance • Precision • **Positive predictive value (PPV)**

Error of 1. und 2. kind

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund

y_i \ p_i	H_0	H_1
$\hat{H}_0$	T_N	F_N
$\hat{H}_1$	F_P	T_P

- To refresh your minds: which of these quantities refers to the error of 1. (α) and 2. (β) kind?

Error of 1. und 2. kind

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal

H_0 : Alternative hypothesis, Untergrund

- To refresh your minds: which of these quantities refers to the error of 1. (α) and 2. (β) kind?

y_i \ p_i	H_0	H_1
$\hat{H}_0$	T_N	F_N β $f_N = \frac{F_N}{T_P + F_N}$ • Miss rate • **False negative rate (FNR)**
$\hat{H}_1$	F_P α $f_P = \frac{F_P}{F_P + T_N}$ • Fallout • **False positive rate (FPR)**	T_P

Reminder separation power

- For the special case of binary classification this assessment can be reduced to the discussion of *binary hypothesis tests*:

H_1 : Hypothesis under test, Signal
H_0 : Alternative hypothesis, Untergrund

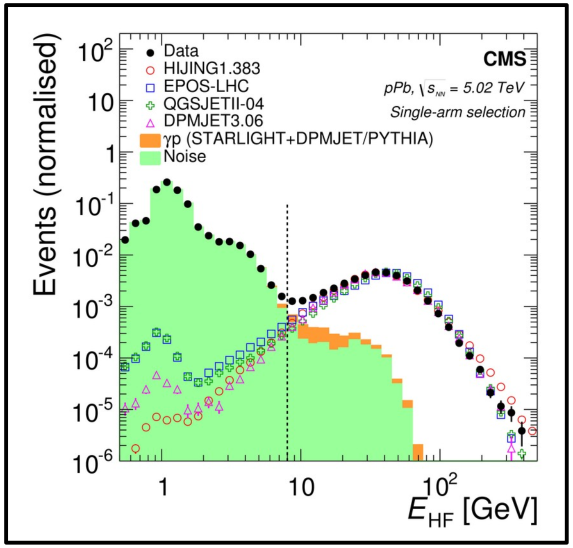
y_i \ p_i	H_0	H_1
$\hat{H}_0$	T_N	F_N β $f_N = \frac{F_N}{T_P + F_N}$ • Miss rate • **False negative rate (FNR)**
$\hat{H}_1$	F_P α $f_P = \frac{F_P}{F_P + T_N}$ • Fallout • **False positive rate (FPR)**	T_P

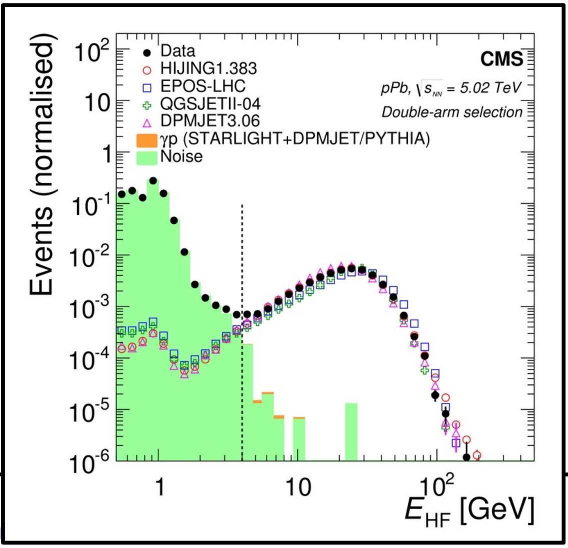
- The function $1 - \beta(\alpha, c, n)$ is called **separation power** of the hypothesis test.

Here c is the critical value of $\hat{y}_i$ on which the acceptance of $\hat{H}_{0/1}$ is based and n is the sample size.

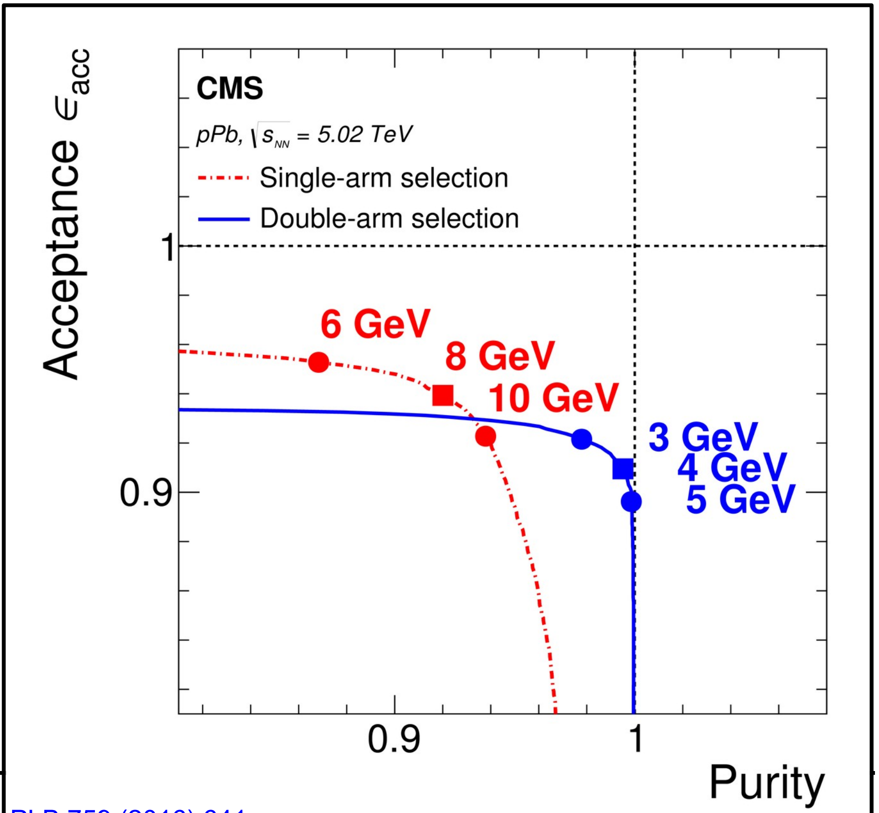
ROC curve

- For binary classification the separation power is often displayed in form of the *receiver operating characteristics* (**ROC**) curve:





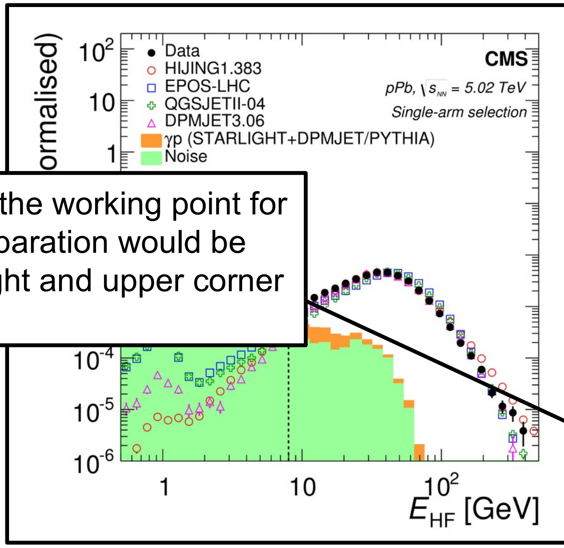
An example from particle physics:



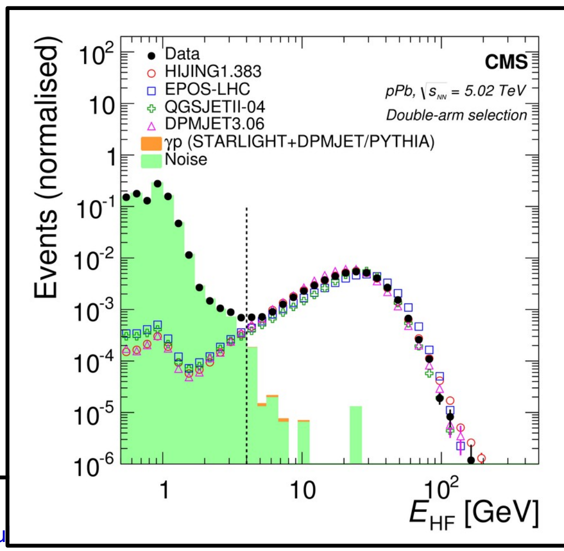
PLB 759 (2016) 641

ROC curve

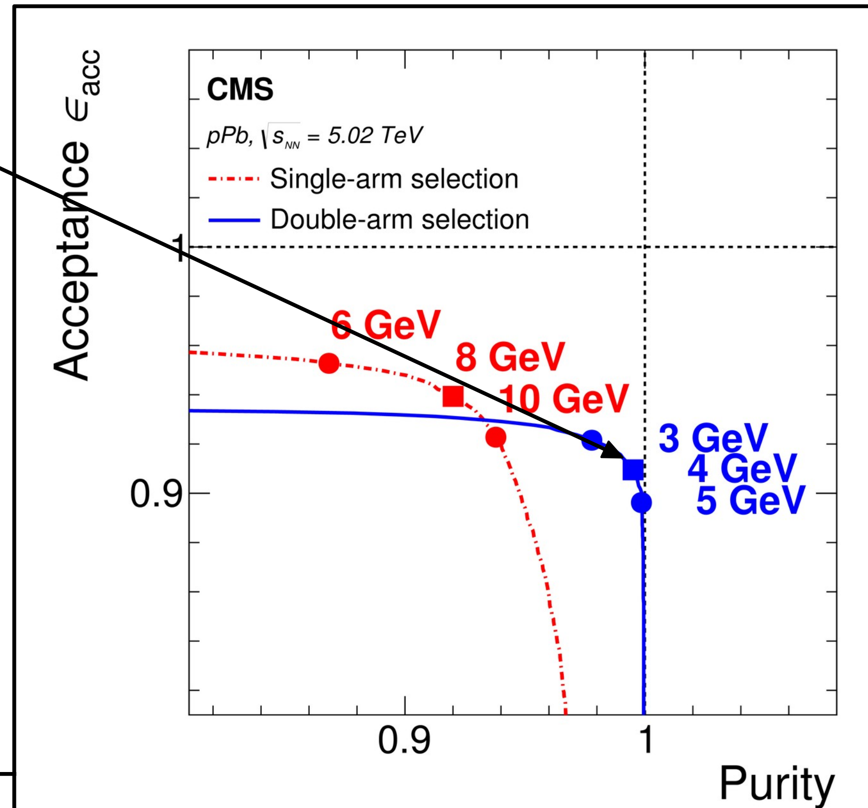
- For binary classification the separation power is often displayed in form of the *reciever operating characteristics* (**ROC**) curve:



In this representation the working point for signal/background separation would be chosen in the most right and upper corner of the ROC curve.



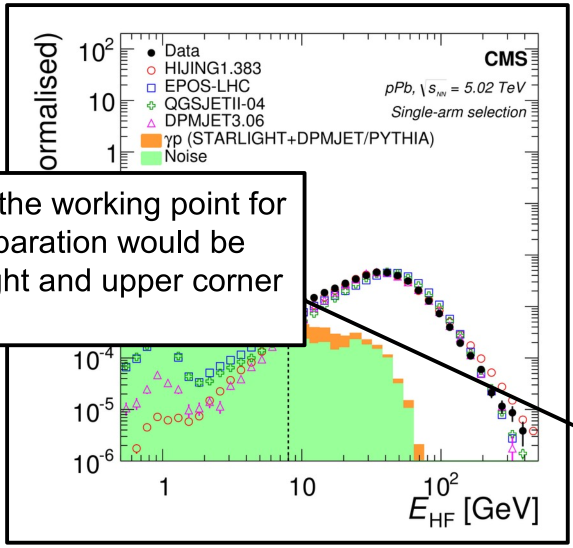
An example from particle physics:



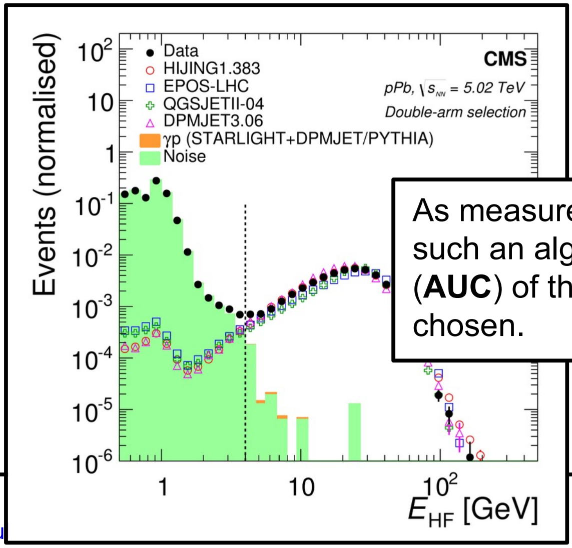
PLB 759 (2016) 641

ROC curve

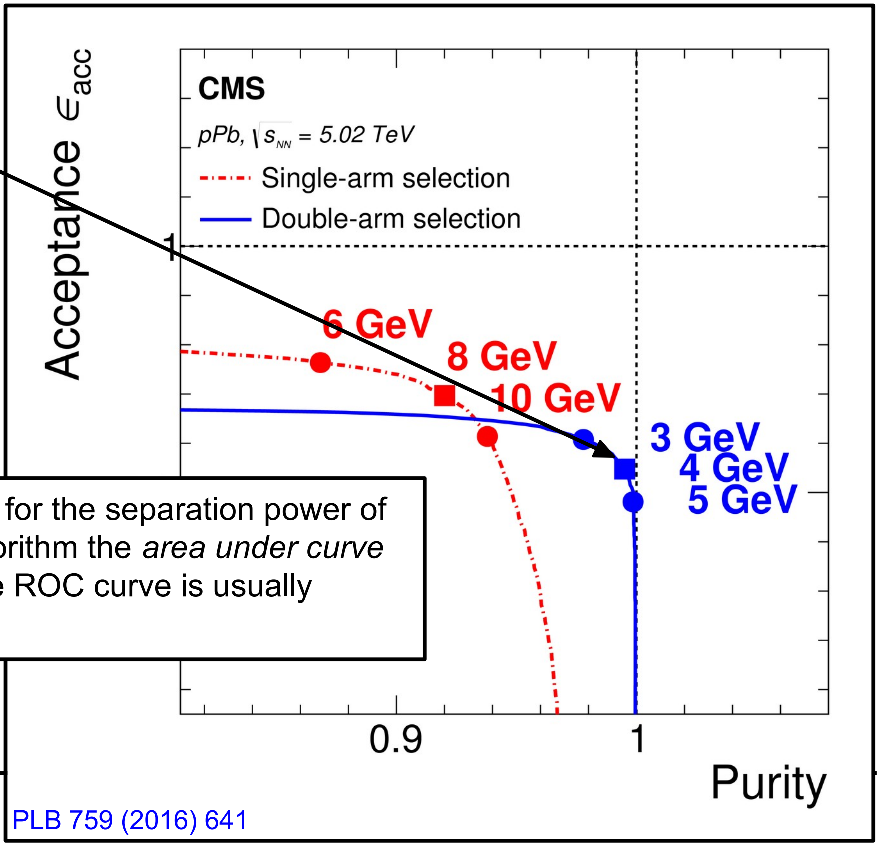
- For binary classification the separation power is often displayed in form of the *reciever operating characteristics* (**ROC**) curve:



In this representation the working point for signal/background separation would be chosen in the most right and upper corner of the ROC curve.



An example from particle physics:

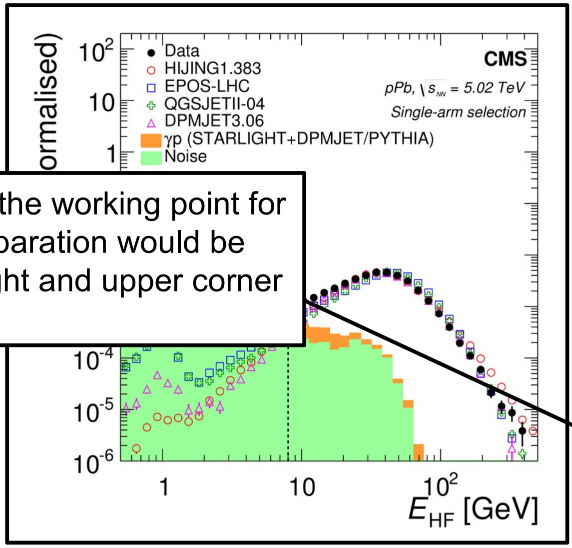


As measure for the separation power of such an algorithm the *area under curve* (**AUC**) of the ROC curve is usually chosen.

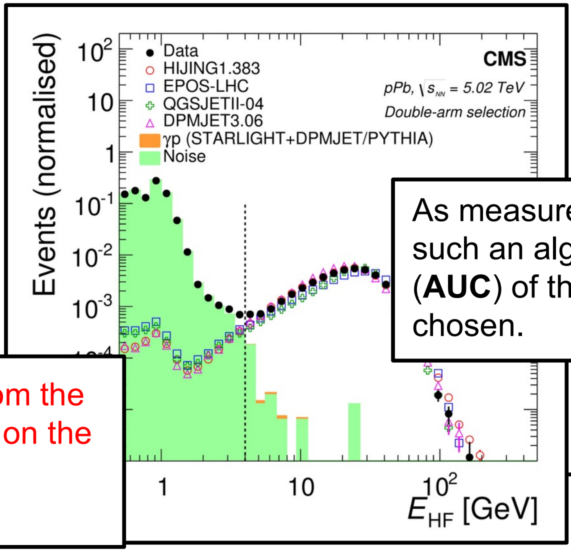
PLB 759 (2016) 641

ROC curve

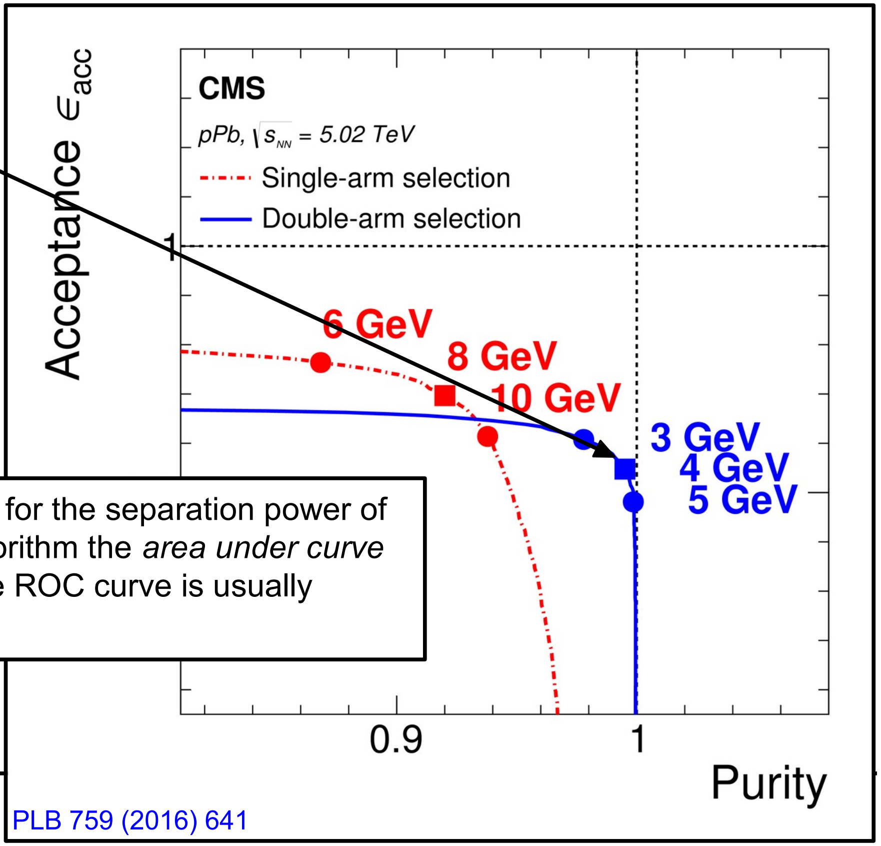
- For binary classification the separation power is often displayed in form of the *reciever operating characteristics* (**ROC**) curve:



In this representation the working point for signal/background separation would be chosen in the most right and upper corner of the ROC curve.



An example from particle physics:

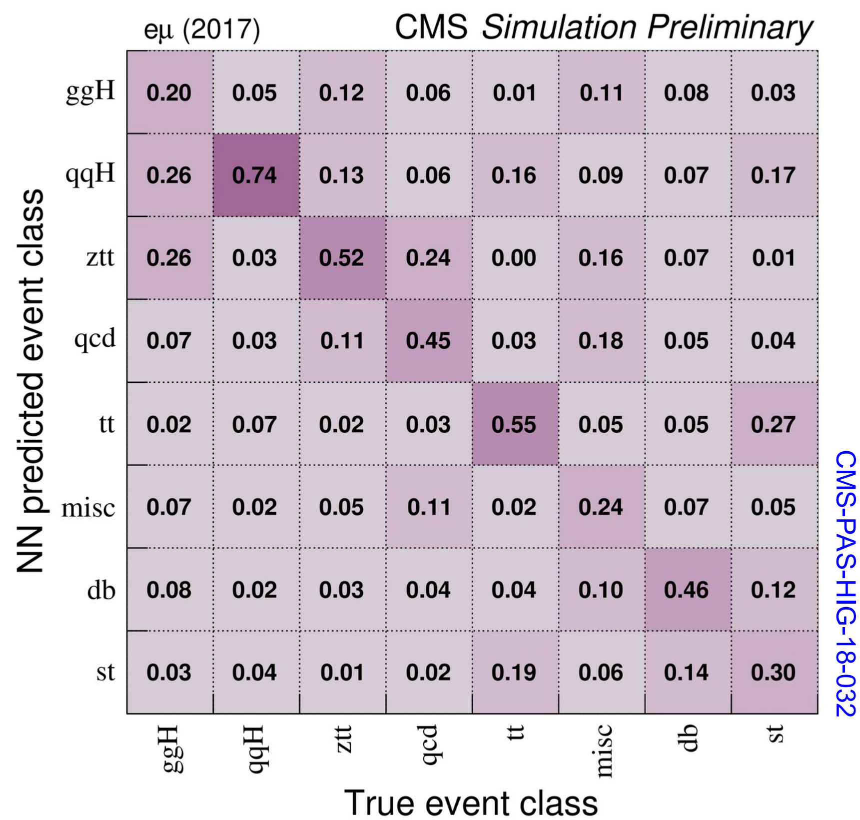


As measure for the separation power of such an algorithm the *area under curve* (**AUC**) of the ROC curve is usually chosen.

Recap: which quantities from the previous slides do you find on the x- and y-axis of this representation?

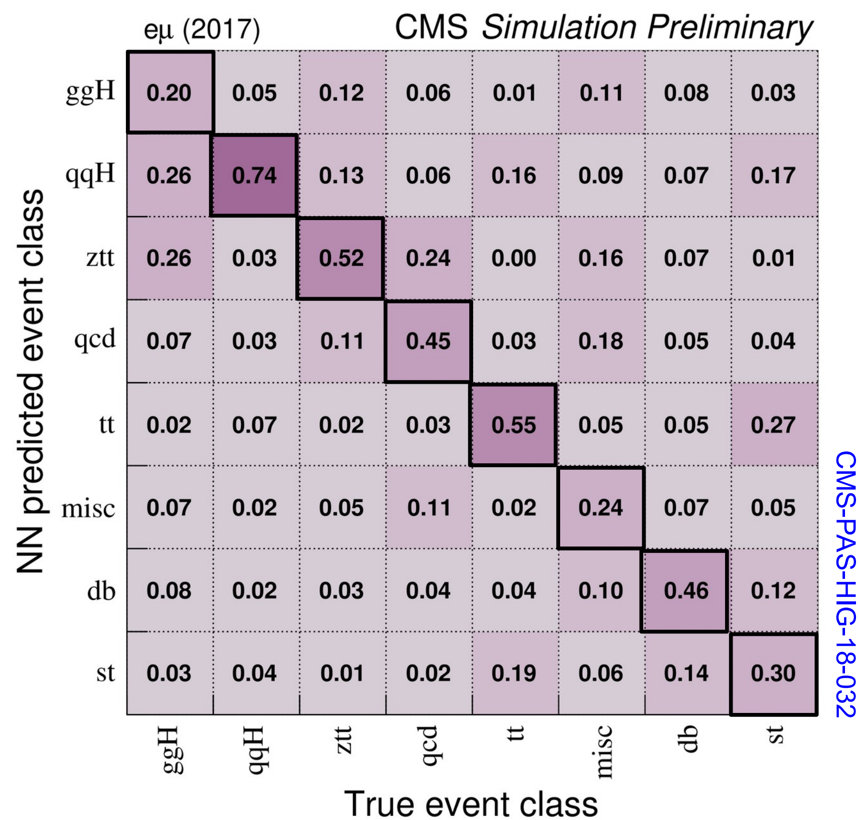
Confusion matrix

- For applying a ROC curve to multi-classification it has to be reduced to pairwise binary classification.
- Alternatively the assessment is based on a form of the confusion matrix:



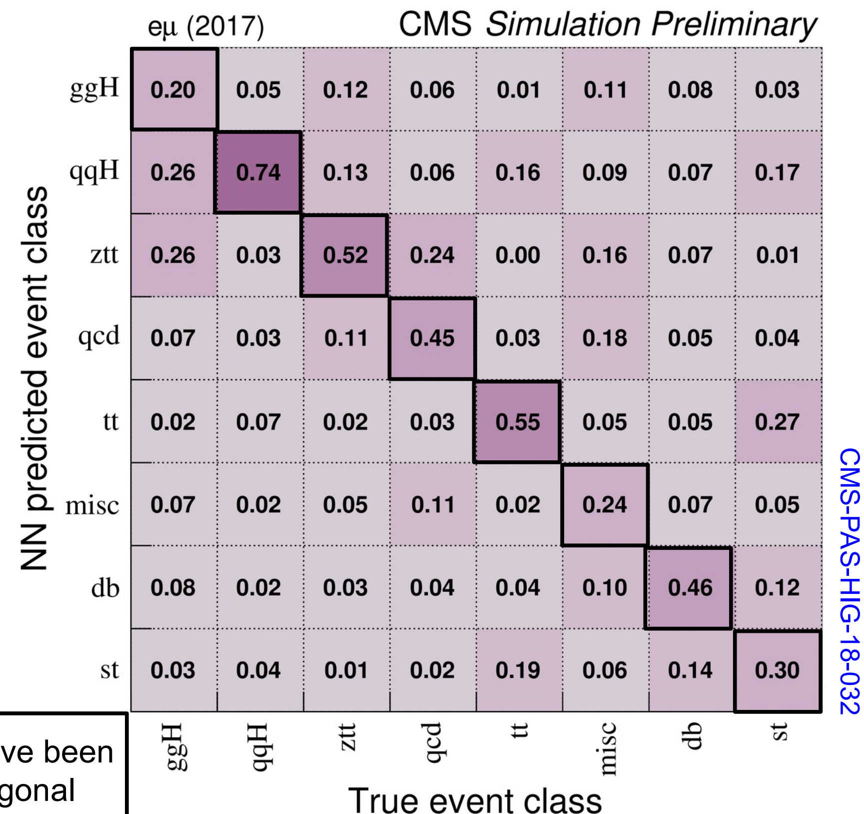
Confusion matrix

- For applying a ROC curve to multi-classification it has to be reduced to pairwise binary classification.
- Alternatively the assessment is based on a form of the confusion matrix:
- Here one prefers large values on the diagonal of the matrix.



Confusion matrix

- For applying a ROC curve to multi-classification it has to be reduced to pairwise binary classification.
- Alternatively the assessment is based on a form of the confusion matrix:
- Here one prefers large values on the diagonal of the matrix.
- There are various flavors of confusion matrices, depending on how its entries have been normalized/scaled (or not).

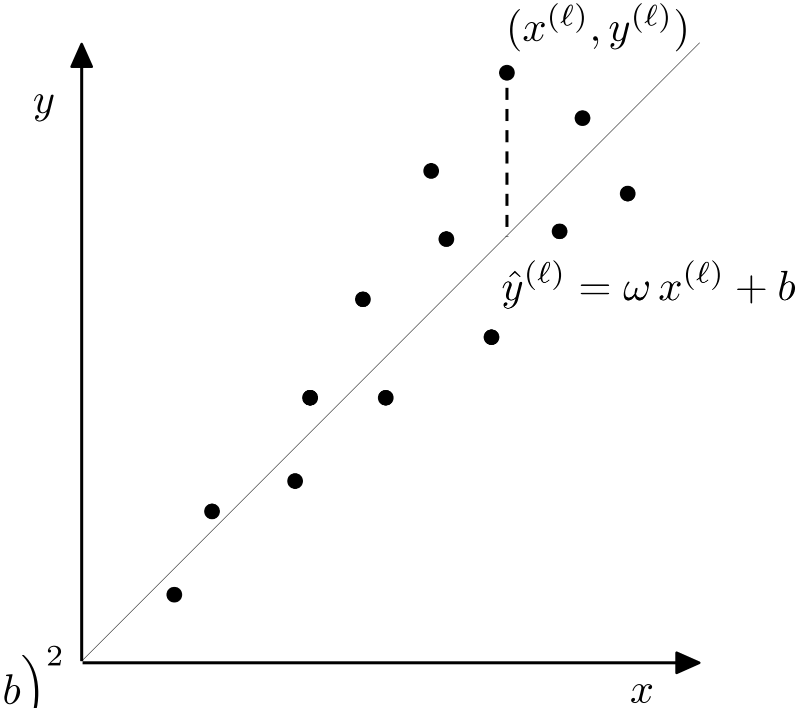


In this case the columns have been normalized to 1, i.e. the diagonal entries correspond to the TPR (also called purity).

Backup

Linear regression – NN model –

- $(x^{(\ell)}, y^{(\ell)})$: value pair of sample ($x^{(\ell)}$) and truth-label ($y^{(\ell)}$);
- Model: $\hat{y}^{(\ell)} = \omega\, x^{(\ell)} + b$
- Activation function: Identity
- Loss function L_2 norm
- Empirical risk functional: MSE
- Minimization algorithm: gradient descent



$$\hat{R}\left(\{y^{(\ell)}\}, \{\hat{y}^{(\ell)}(x^{(\ell)}, \omega, b)\}\right) = \sum_{\ell=1}^{N}\left(y^{(\ell)} - \hat{y}^{(\ell)}\right)^2$$
$$= \sum_{\ell=1}^{N}\left(y^{(\ell)} - \omega\, x^{(\ell)} + b\right)^2$$

NN training (by human)

- Necessary conditions for minimum:

$$\frac{\partial\hat{R}}{\partial b} = -2\sum_{\ell=1}^{N}\left(y^{(\ell)} - \omega\, x^{(\ell)} + b\right) \qquad = 0;$$

$$\frac{\partial\hat{R}}{\partial\omega} = -2\sum_{\ell=1}^{N}\left(y^{(\ell)} - \omega\, x^{(\ell)} + b\right)\, x^{(\ell)} = 0;$$

- Normal equations:

$$\sum y^{(\ell)} \qquad = \omega\sum x^{(\ell)} + N\,b; \qquad (\mathbf{1})$$

$$\sum y^{(\ell)}x^{(\ell)} = \omega\sum x^{(\ell)\,2} + b\sum x^{(\ell)}; (\mathbf{2})$$

$$N\,\overline{x}\,(\mathbf{1}): \qquad N^2 \quad \overline{y}\;\overline{x} \;= N^2\,\omega\,\overline{x}^2 + N^2\,b\,\overline{x}$$

$$N \quad (\mathbf{2}): \qquad N\sum y_i x_i = N^2\,\omega\,\overline{x^2} + N^2\,b\,\overline{x}$$

$$N\,(\mathbf{2}) - N\,\overline{x}\,(\mathbf{1})$$

$$\boxed{\omega = \frac{\sum y_i x_i - N\,\overline{y}\,\overline{x}}{\sum x_i^2 - N\,\overline{x}^2}; \qquad b = \overline{y} - \omega\,\overline{x}}$$