# Neural networks for particle physicists

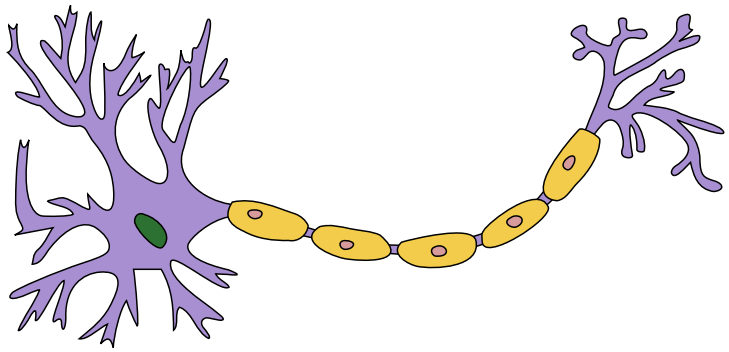
Andrey Popov

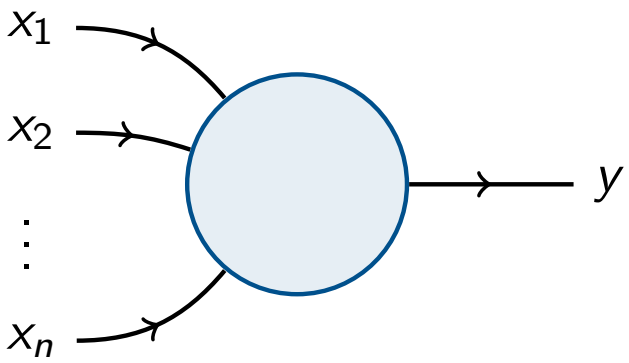
Louvain-la-Neuve
5 Feb 2015

# Part I: Theory

# Artificial neurons

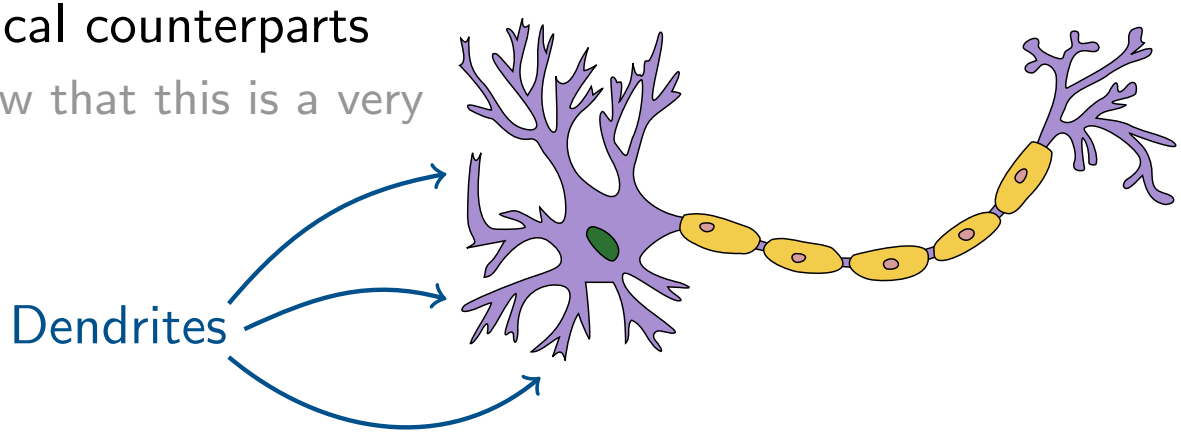
- Artificial neurons were originally created as a model of their biological counterparts
  - Although now we know that this is a very naive description

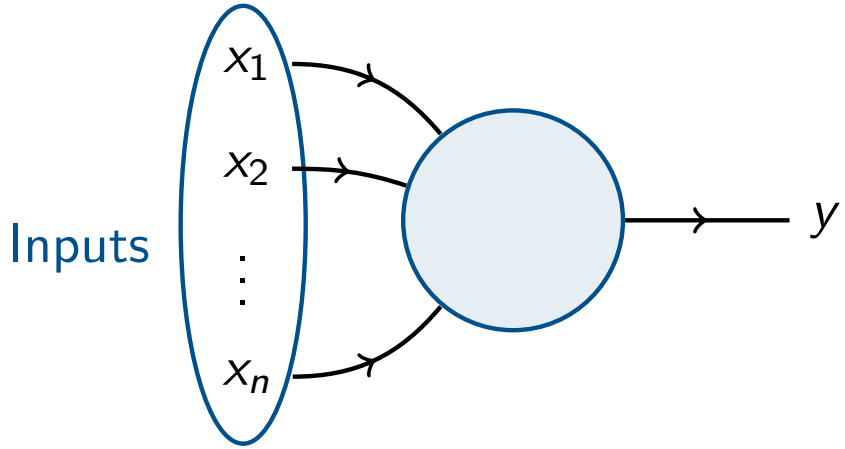




# Artificial neurons

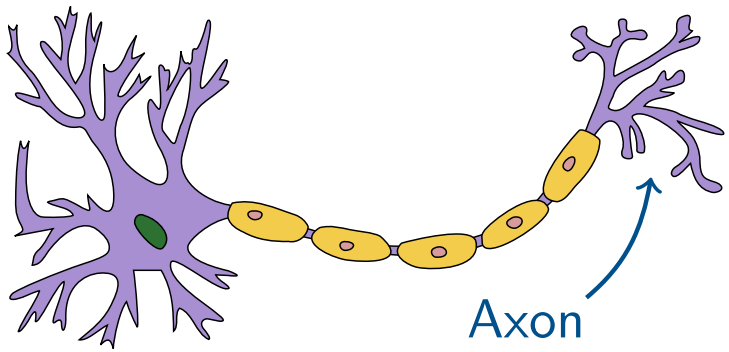
- Artificial neurons were originally created as a model of their biological counterparts
  - Although now we know that this is a very naive description

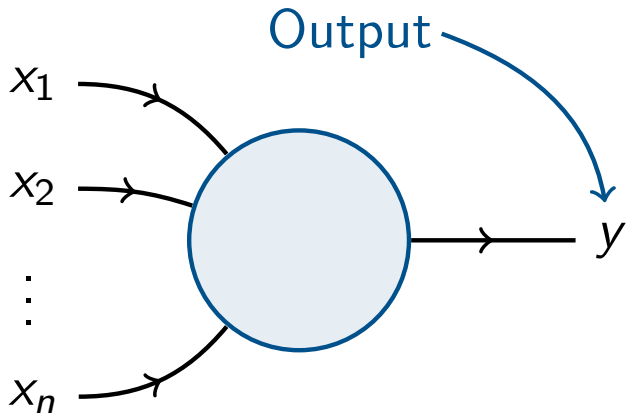


# Artificial neurons

- Artificial neurons were originally created as a model of their biological counterparts
  - Although now we know that this is a very naive description

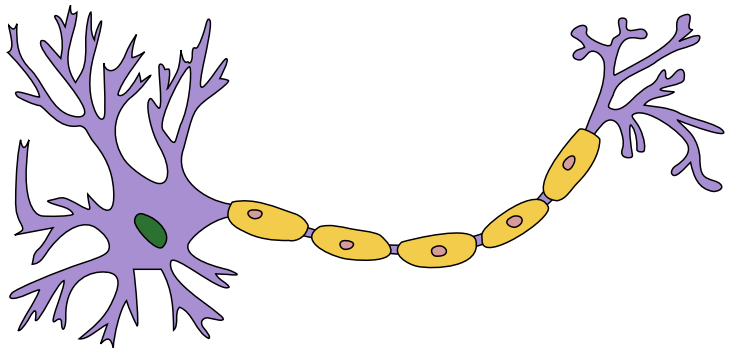


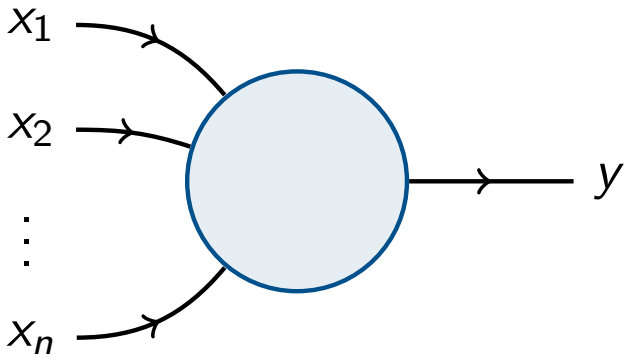



# Artificial neurons

- Artificial neurons were originally created as a model of their biological counterparts
  - Although now we know that this is a very naive description
- Response $y$ to inputs $\mathbf{x}$ is calculated as

$$y = h\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

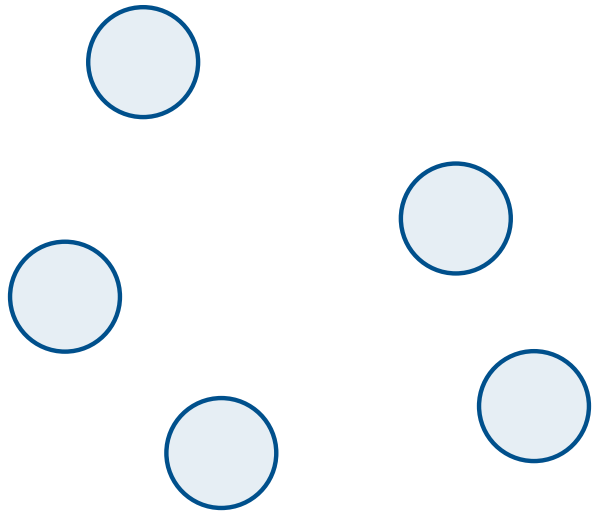





  - Weights $\mathbf{w}$ are parameters of the neuron
  - Different options for activation function $h$
    - Possible choices: step, linear, tanh, sigmoid, radial basis function, ...
    - Best when it is smooth and non-linear

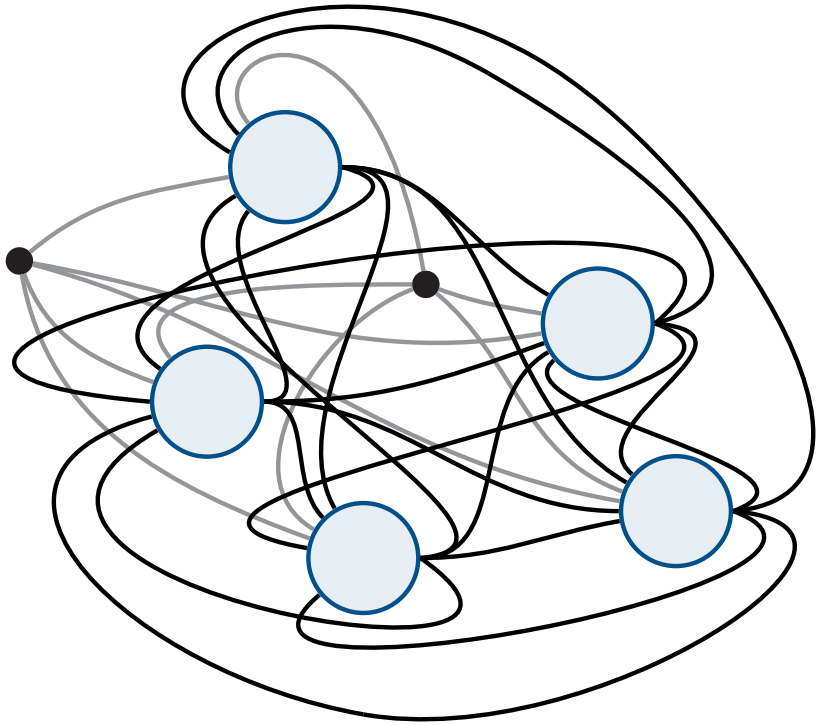
# Neural networks: Putting neurons together

- Several neurons can be combined into a network
  - Outputs of some neurons are connected to inputs of others



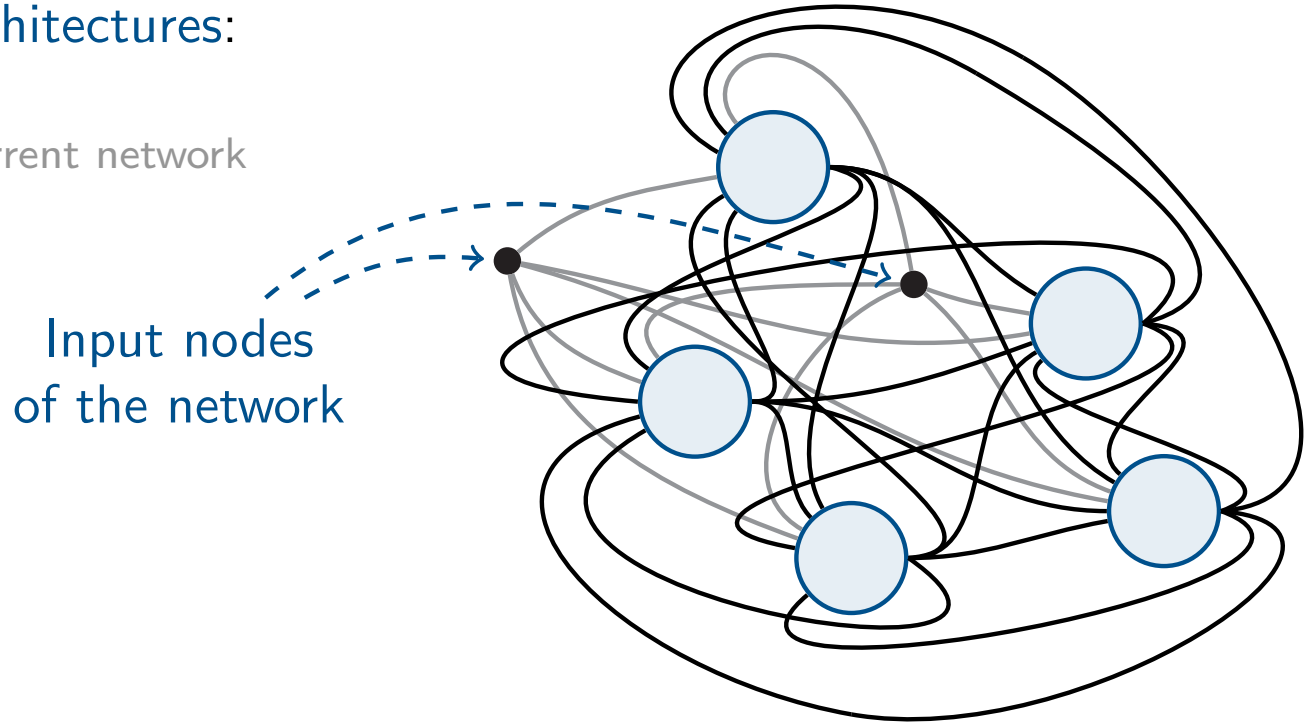
# Neural networks: Putting neurons together

- Several neurons can be combined into a network
  - Outputs of some neurons are connected to inputs of others
- Diversity of architectures:
  - All-to-all
    - Fully recurrent network



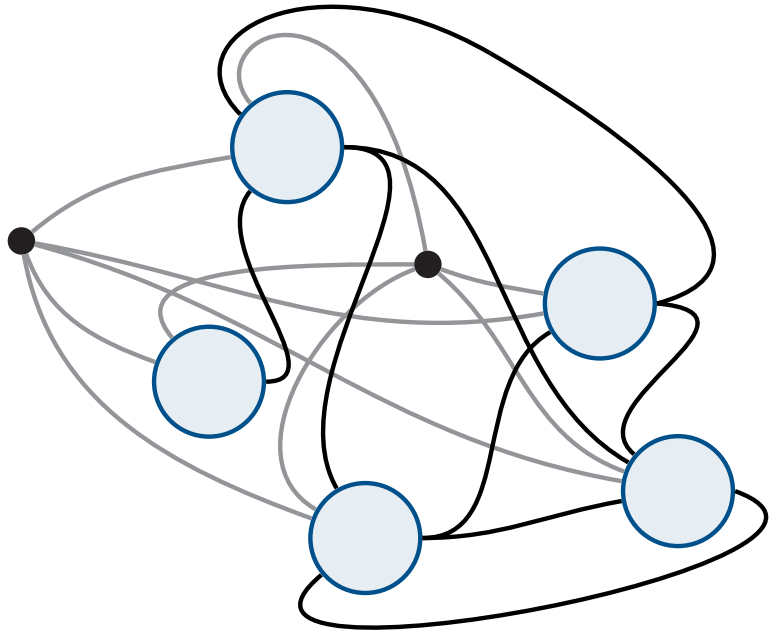
# Neural networks: Putting neurons together

- Several neurons can be combined into a network
  - Outputs of some neurons are connected to inputs of others
- Diversity of architectures:
  - All-to-all
    - Fully recurrent network



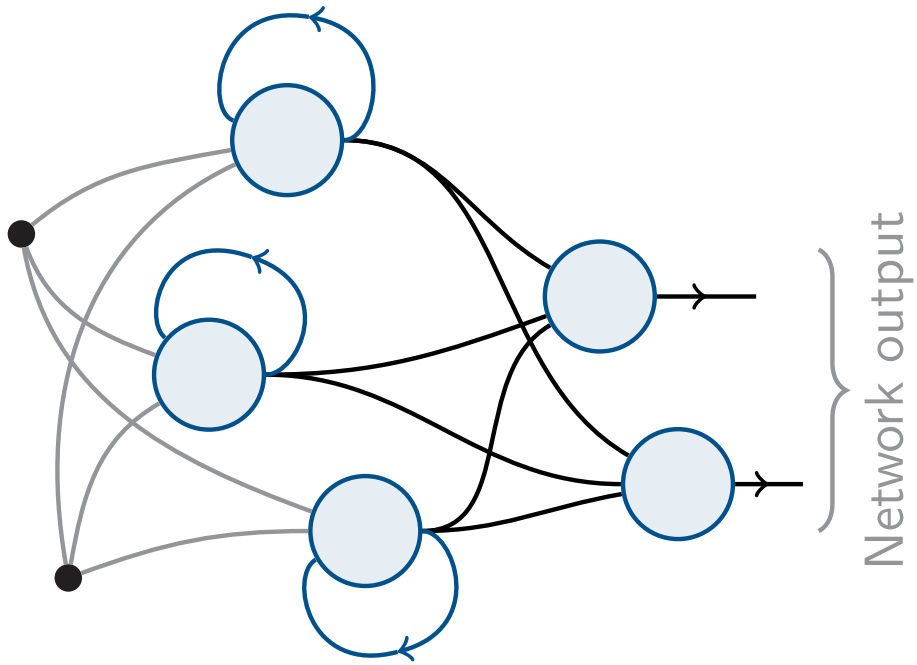
# Neural networks: Putting neurons together

- Several neurons can be combined into a network
  - Outputs of some neurons are connected to inputs of others
- Diversity of architectures:
  - All-to-all
    - Fully recurrent network
  - Random sparse connections
    - Echo state network



# Neural networks: Putting neurons together

- Several neurons can be combined into a network
  - Outputs of some neurons are connected to inputs of others
- Diversity of architectures:
  - All-to-all
    - Fully recurrent network
  - Random sparse connections
    - Echo state network
  - Hierarchical structure with cycles
    - Capable of short-term memory
    - Often used in time series analysis
    - Jordan and Elman networks

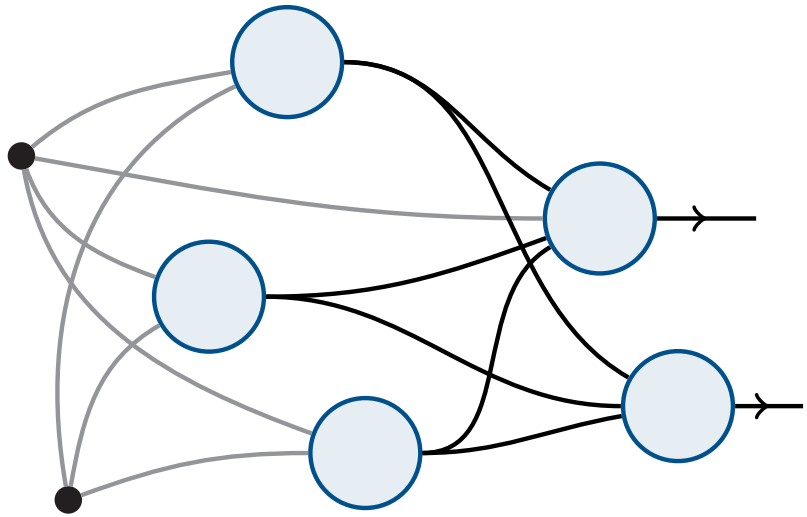


# Neural networks: Putting neurons together

- Several neurons can be combined into a network
  - Outputs of some neurons are connected to inputs of others
- Diversity of architectures:

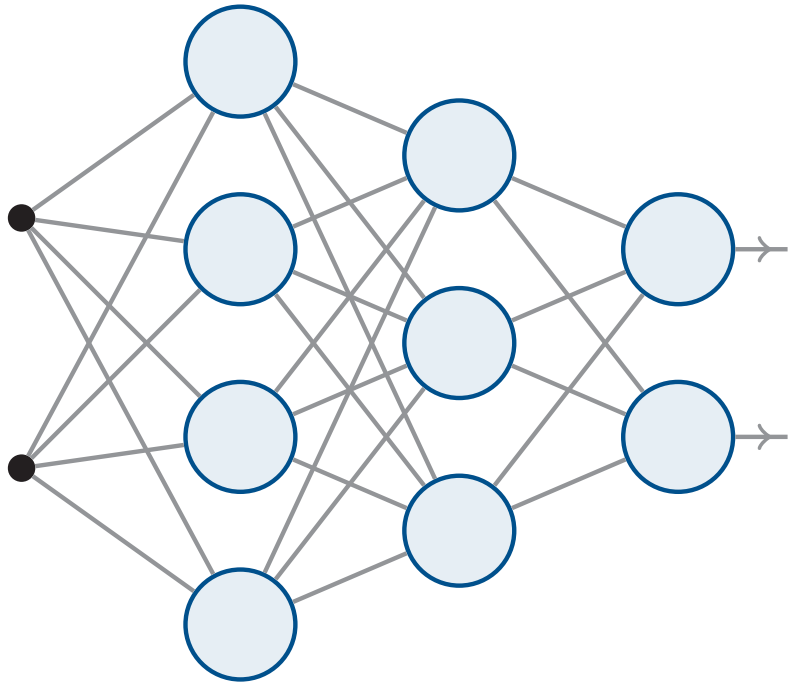
Recurrent NNs

  - All-to-all
    - Fully recurrent network
  - Random sparse connections
    - Echo state network
  - Hierarchical structure with cycles
    - Capable of short-term memory
    - Often used in time series analysis
    - Jordan and Elman networks
  - Feedforward networks
    - Do not contain cycles



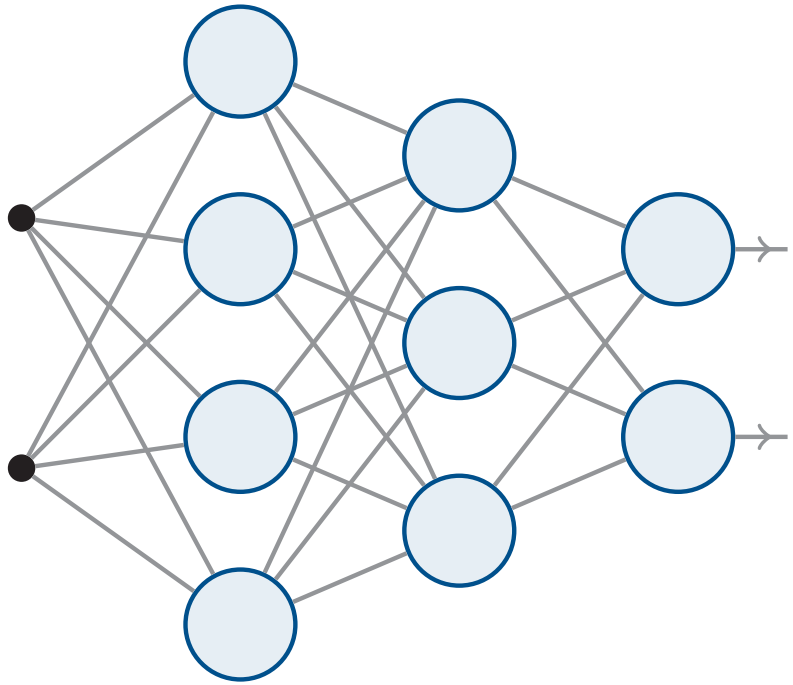
# Multilayer perceptron

- Multilayer perceptron is (arguably) the most popular architecture
  - Simple, yet very powerful



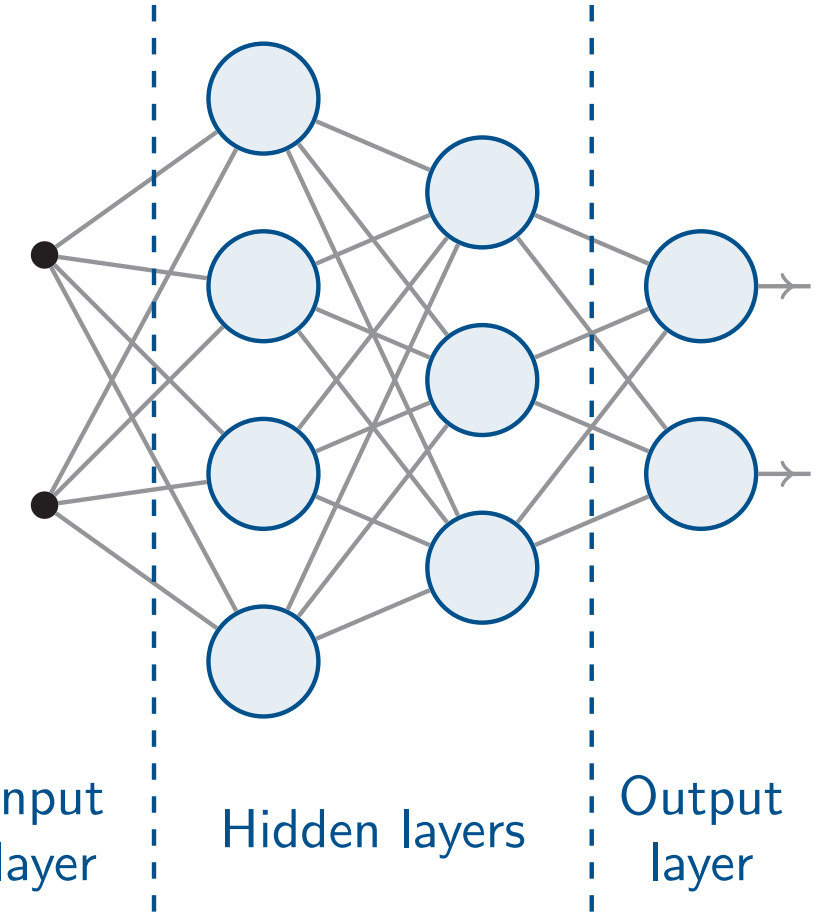
# Multilayer perceptron

- Multilayer perceptron is (arguably) the most popular architecture
  - Simple, yet very powerful
- Feedforward structure



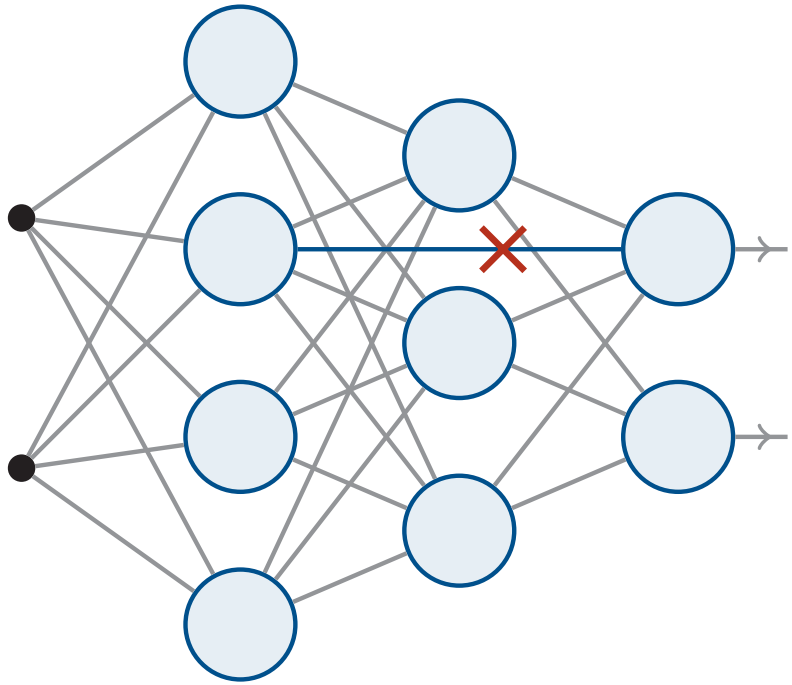
# Multilayer perceptron

- Multilayer perceptron is (arguably) the most popular architecture
  - Simple, yet very powerful
- Feedforward structure
- Organised into layers
  - One or more hidden layers
  - Output layer may contain more than one neuron



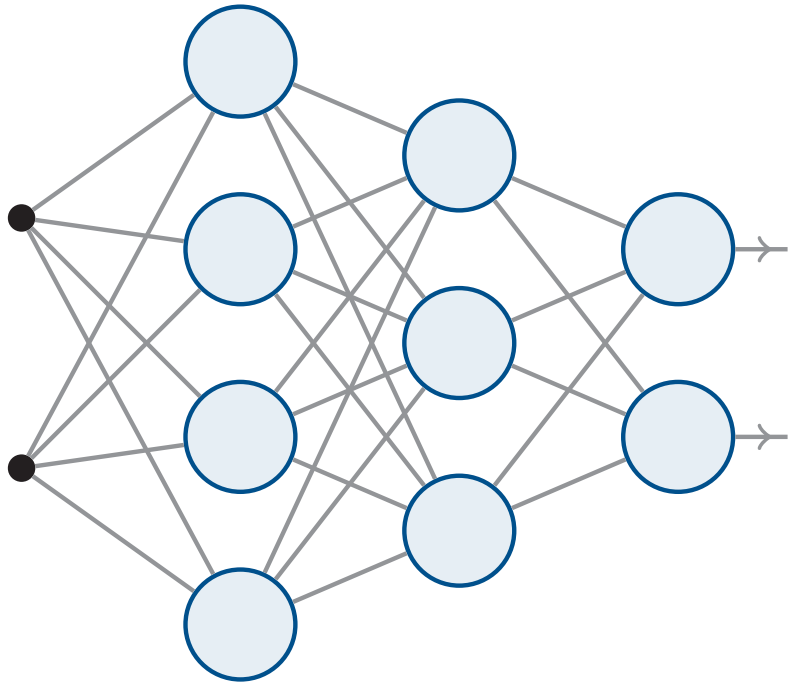
# Multilayer perceptron

- Multilayer perceptron is (arguably) the most popular architecture
  - Simple, yet very powerful
- Feedforward structure
- Organised into layers
  - One or more hidden layers
  - Output layer may contain more than one neuron
- Neurons feed their responses to the next layer only
  - No deep forward connections



# Multilayer perceptron

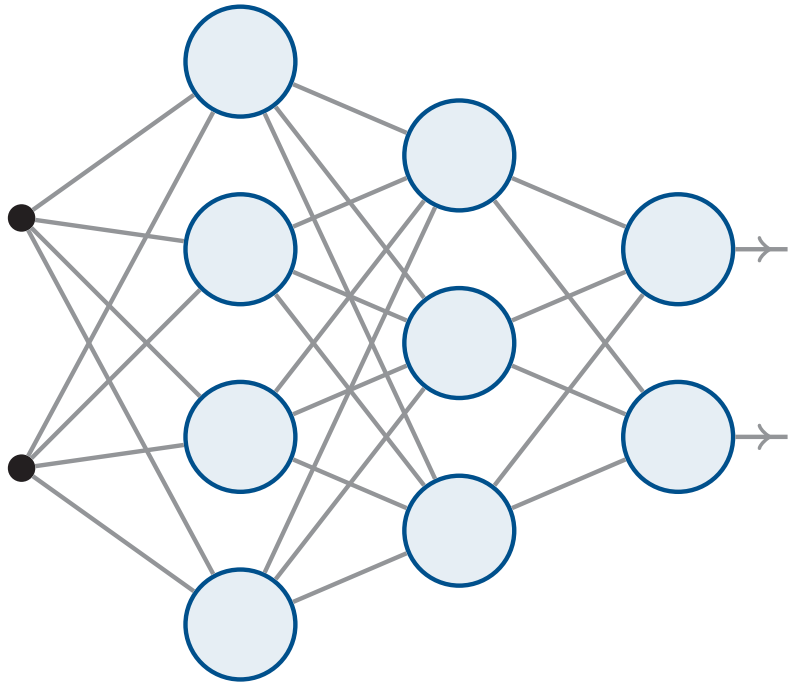
- Multilayer perceptron is (arguably) the most popular architecture
  - Simple, yet very powerful
- Feedforward structure
- Organised into layers
  - One or more hidden layers
  - Output layer may contain more than one neuron
- Neurons feed their responses to the next layer only
  - No deep forward connections



- Can be viewed as a mapping of the space of inputs onto a space of (usually) a smaller dimensionality $\mathbb{R}^n \mapsto \mathbb{R}^m$, $m < n$
  - Weights are parameters of the mapping

# Multilayer perceptron

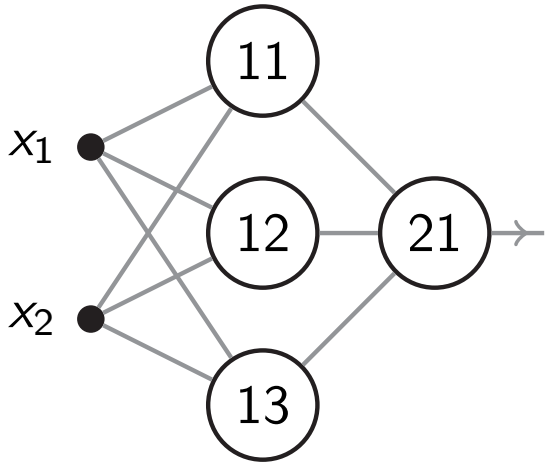
- Multilayer perceptron is (arguably) the most popular architecture
  - Simple, yet very powerful
- Feedforward structure
- Organised into layers
  - One or more hidden layers
  - Output layer may contain more than one neuron
- Neurons feed their responses to the next layer only
  - No deep forward connections



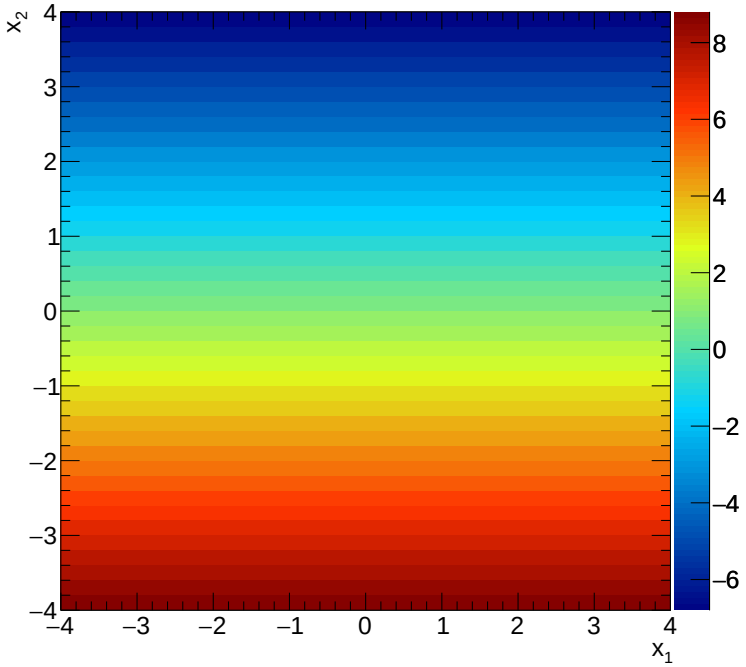
- Can be viewed as a mapping of the space of inputs onto a space of (usually) a smaller dimensionality $\mathbb{R}^n \mapsto \mathbb{R}^m$, $m < n$
  - Weights are parameters of the mapping
- The rest of the talk is focused on multilayer perceptrons

# Visual example

- Let's look at responses of individual neurons in a small multilayer perceptron
  - Single hidden layer only
  - Simple $\mathbb{R}^2 \mapsto \mathbb{R}^1$ problem
- Weights are set manually
- Two different activation functions are tried

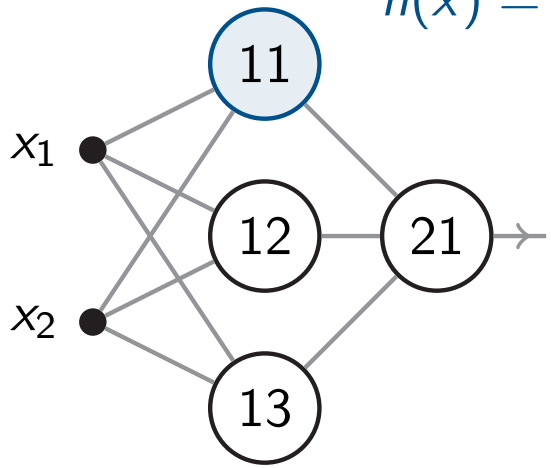


# Visual example



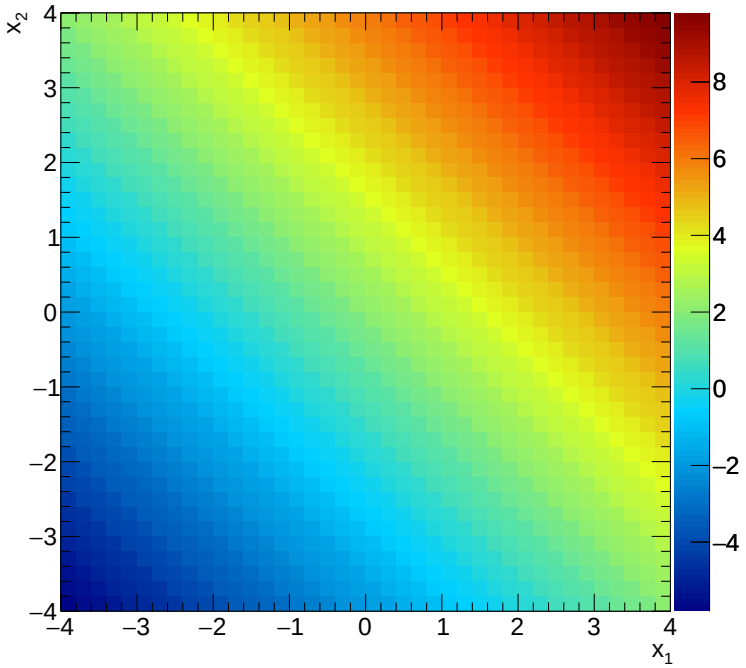

Activation function:

$h(x) = x$



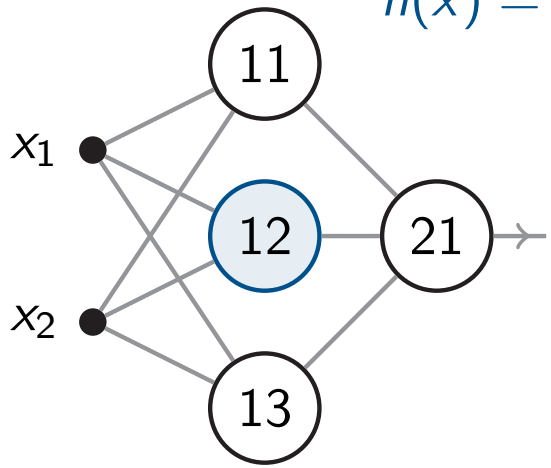

$$y_{11} = -2\,x_2 + 1$$

# Visual example



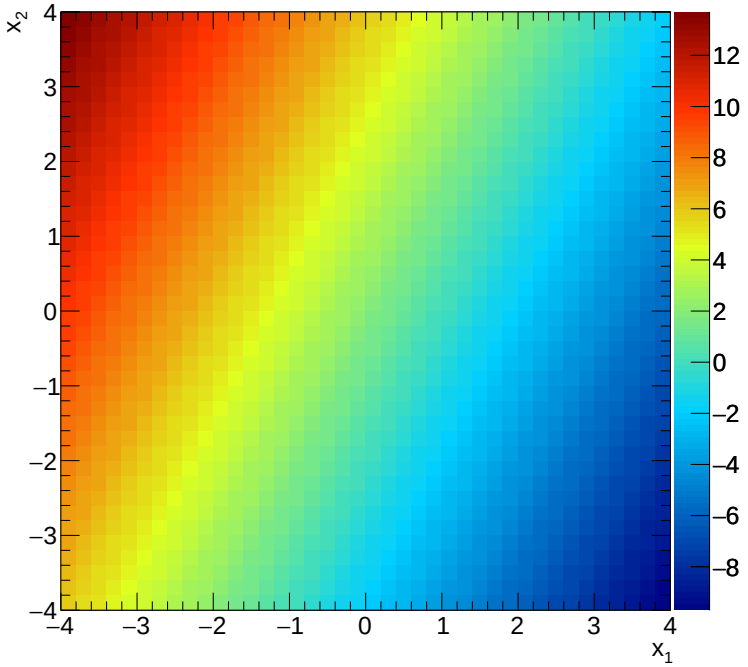

Activation function:

$h(x) = x$



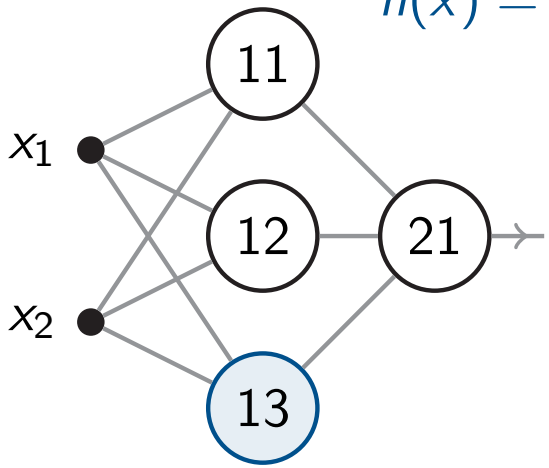

$$y_{12} = x_1 + x_2 + 2$$

# Visual example



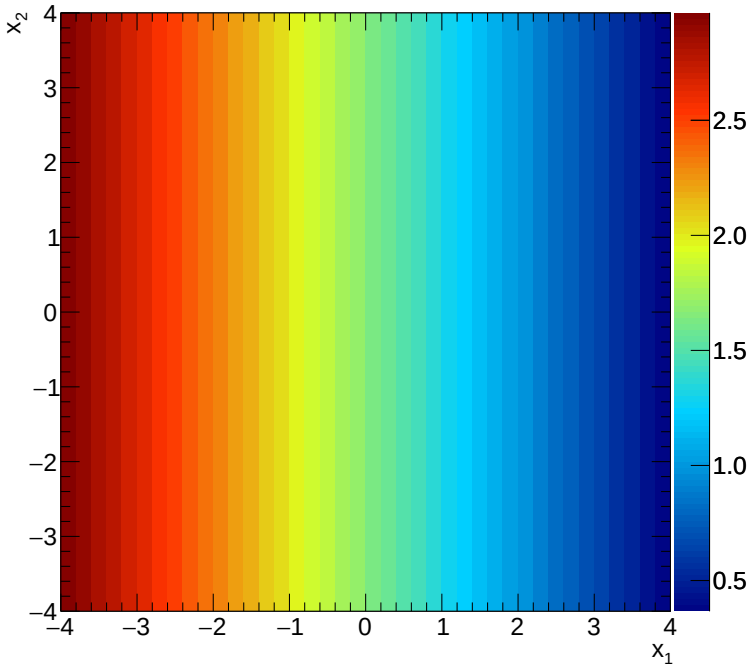

Activation function:

$h(x) = x$

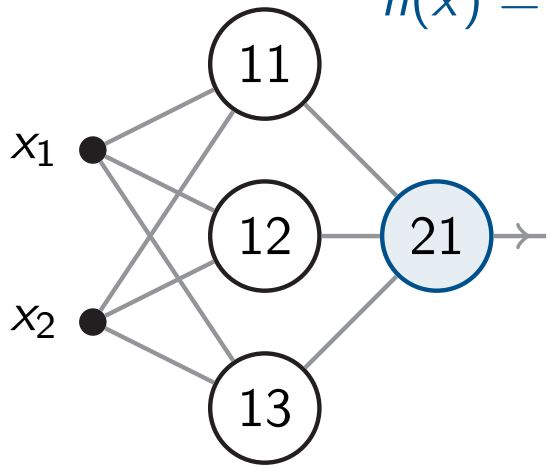



$$y_{13} = 2\,x_1 + x_2 + 2$$

# Visual example



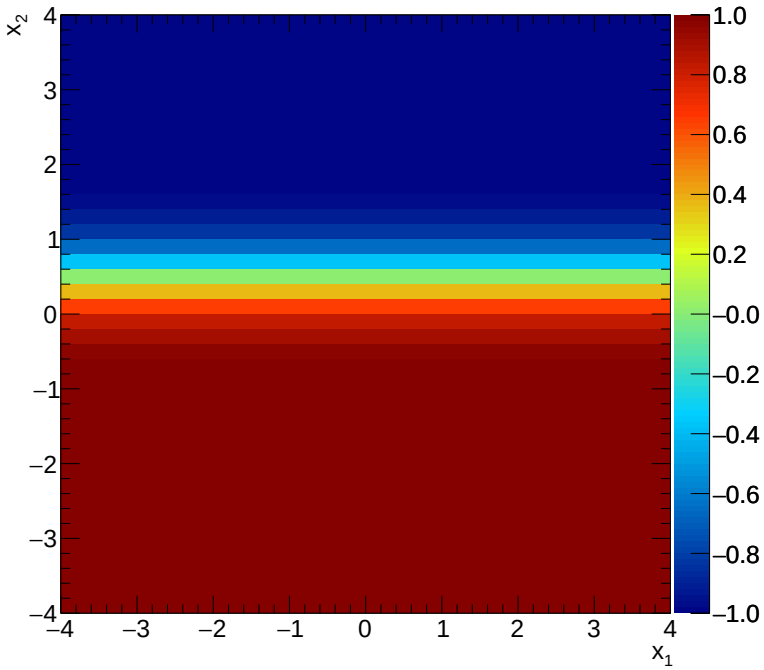

Activation function:
$h(x) = x$



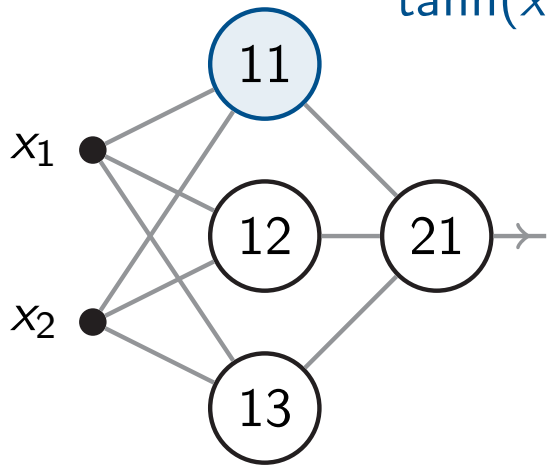

$$y_{21} = \frac{5 - x_1}{3}$$

- Indeed, a linear combination of linear neurons can be described by a single linear neuron

# Visual example

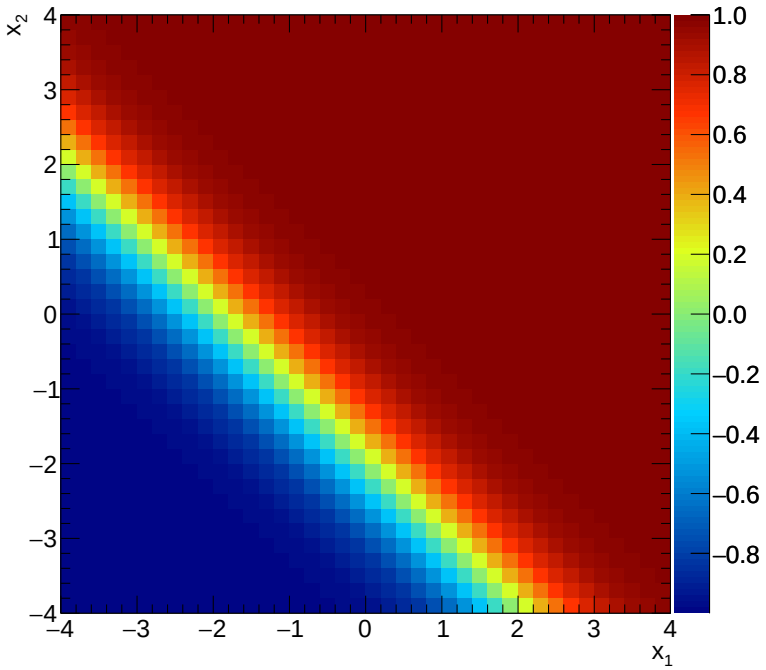



Activation function: $\tanh(x)$

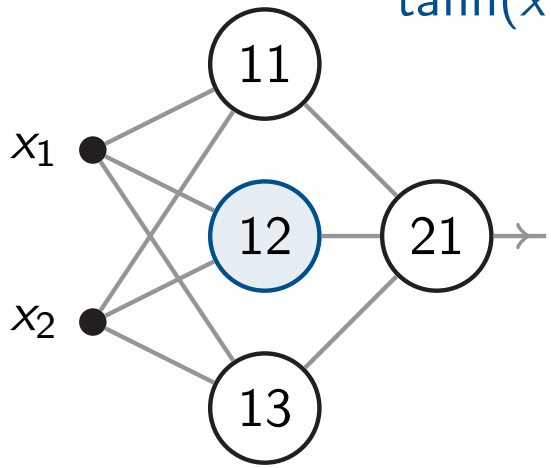



$$y_{11} = \tanh(-2\,x_2 + 1)$$

# Visual example

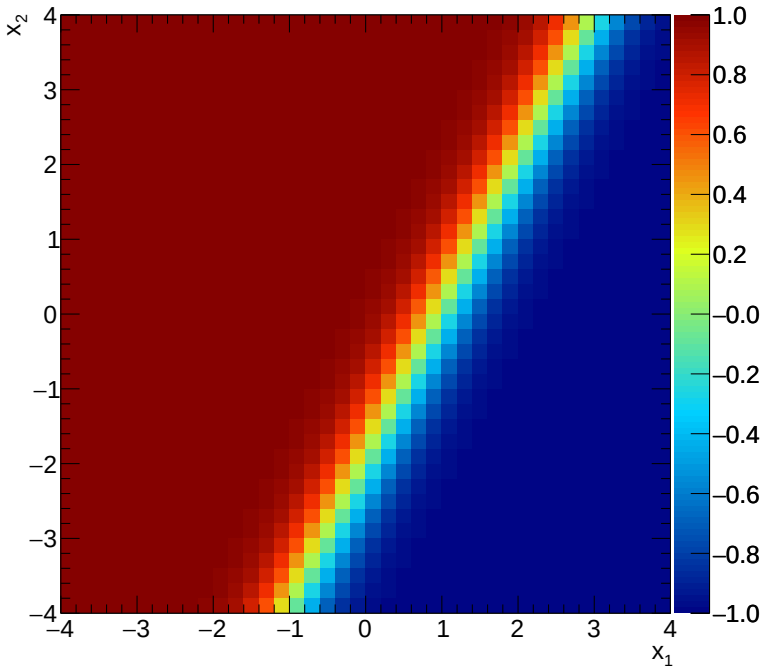



Activation function: $\tanh(x)$

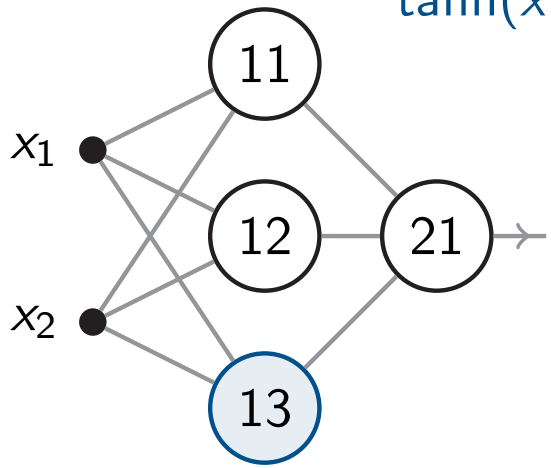



$$y_{12} = \tanh(x_1 + x_2 + 2)$$

# Visual example

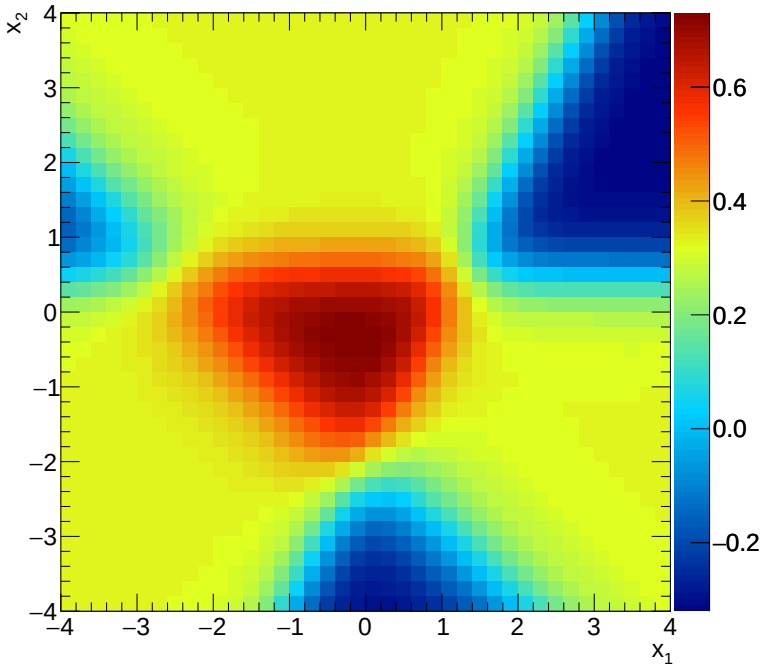



Activation function: $\tanh(x)$

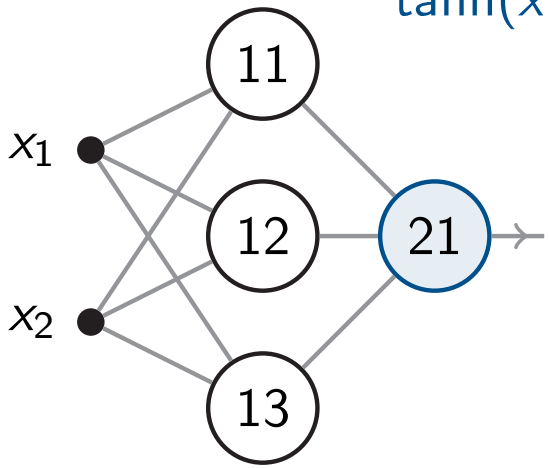



$$y_{13} = \tanh(2\,x_1 + x_2 + 2)$$

# Visual example

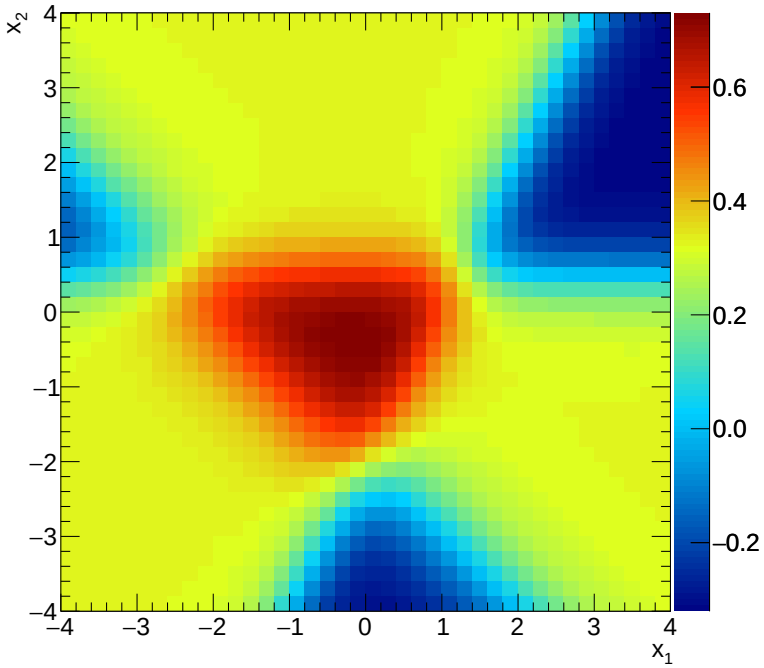



Activation function:
$\tanh(x)$

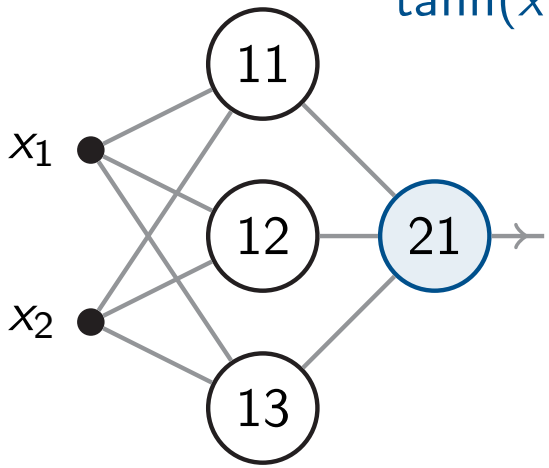



$$y_{21} = \tanh\left(\frac{1}{3}\cdot\tanh(-2\,x_2+1)+\frac{1}{3}\cdot\tanh(x_1+x_2+2)+\frac{1}{3}\cdot\tanh(2\,x_1+x_2+2)\right)$$

# Visual example




Activation function:
$\tanh(x)$




- Non-linear activation function is crucial

# Function approximation

- Universal approximation theorem:
  - A multilayer perceptron with a single hidden layer can approximate any continuous function on a compact subset of $\mathbb{R}^n$

# Function approximation

- Universal approximation theorem:
  - A multilayer perceptron with a single hidden layer can approximate any continuous function on a compact subset of $\mathbb{R}^n$
- The target function is usually known at a finite set of points only
  - NN is given pairs of inputs and desired outputs $\Rightarrow$ supervised learning

# Function approximation

- Universal approximation theorem:
  - A multilayer perceptron with a single hidden layer can approximate any continuous function on a compact subset of $\mathbb{R}^n$
- The target function is usually known at a finite set of points only
  - NN is given pairs of inputs and desired outputs $\Rightarrow$ supervised learning
- A loss function quantifies how much the actual response differs from the desired output over a set of examples

# Function approximation

- Universal approximation theorem:
  - A multilayer perceptron with a single hidden layer can approximate any continuous function on a compact subset of $\mathbb{R}^n$
- The target function is usually known at a finite set of points only
  - NN is given pairs of inputs and desired outputs $\Rightarrow$ supervised learning
- A loss function quantifies how much the actual response differs from the desired output over a set of examples
  - A popular choice is the mean squared error

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (y(\mathbf{x}_i; \mathbf{w}) - \hat{y}_i)^2$$

Here $\{\mathbf{x}_i, \hat{y}_i\}_{i=1}^N$ is the set of examples, and $y$ is the actual NN response, which depends on the weights $\mathbf{w}$

# Function approximation

- Universal approximation theorem:
  - A multilayer perceptron with a single hidden layer can approximate any continuous function on a compact subset of $\mathbb{R}^n$
- The target function is usually known at a finite set of points only
  - NN is given pairs of inputs and desired outputs $\Rightarrow$ supervised learning
- A loss function quantifies how much the actual response differs from the desired output over a set of examples
  - A popular choice is the mean squared error

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (y(\mathbf{x}_i; \mathbf{w}) - \hat{y}_i)^2$$

Here $\{\mathbf{x}_i, \hat{y}_i\}_{i=1}^N$ is the set of examples, and $y$ is the actual NN response, which depends on the weights $\mathbf{w}$

  - Other options are possible, e. g. the likelihood of correct classification, equivalent to the cross-entropy $\frac{1}{N} \sum_i (-\hat{y}_i \ln y_i - (1 - \hat{y}_i) \ln(1 - y_i))$

# Training: Backpropagation

- Training can be viewed as minimisation of the loss function $E(\mathbf{w})$
  - Not an easy problem since the dimensionality is $\mathcal{O}(10^2 - 10^3)$
  - The loss function can have a very complex landscape

# Training: Backpropagation

- Training can be viewed as minimisation of the loss function $E(\mathbf{w})$
  - Not an easy problem since the dimensionality is $\mathcal{O}(10^2 - 10^3)$
  - The loss function can have a very complex landscape
- Backpropagation is a common method
  - Iterative algorithm

# Training: Backpropagation

- Training can be viewed as minimisation of the loss function $E(\mathbf{w})$
  - Not an easy problem since the dimensionality is $\mathcal{O}(10^2 - 10^3)$
  - The loss function can have a very complex landscape
- Backpropagation is a common method
  - Iterative algorithm
  - At each epoch calculates the gradient $\nabla E$ analytically

# Training: Backpropagation

- Training can be viewed as minimisation of the loss function $E(\mathbf{w})$
  - Not an easy problem since the dimensionality is $\mathcal{O}(10^2 - 10^3)$
  - The loss function can have a very complex landscape
- Backpropagation is a common method
  - Iterative algorithm
  - At each epoch calculates the gradient $\nabla E$ analytically
    - A smart choice of the activation function allows to express $\nabla E$ in terms of known responses of individual neurons. For instance,

$$\frac{\partial y}{\partial w_k} = \frac{\partial}{\partial w_k} \tanh\left(\sum_i w_i x_i + w_0\right) = \left(1 - \tanh^2\left(\sum_i w_i x_i + w_0\right)\right) \cdot x_k = (1 - y^2) \cdot x_k$$

# Training: Backpropagation

- Training can be viewed as minimisation of the loss function $E(\mathbf{w})$
  - Not an easy problem since the dimensionality is $\mathcal{O}(10^2 - 10^3)$
  - The loss function can have a very complex landscape
- Backpropagation is a common method
  - Iterative algorithm
  - At each epoch calculates the gradient $\nabla E$ analytically
    - A smart choice of the activation function allows to express $\nabla E$ in terms of known responses of individual neurons
  - Updates weights as

$$\mathbf{w} \to \mathbf{w}' = \mathbf{w} - \eta \nabla E$$

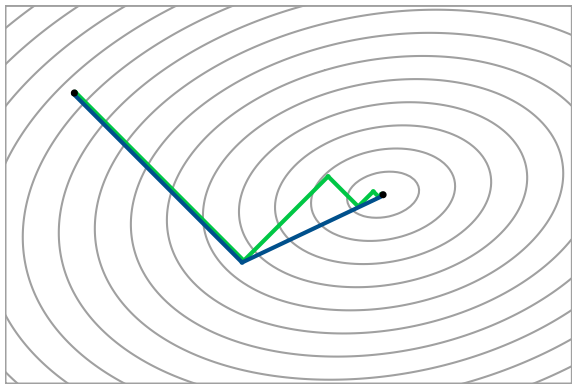
    - Here the learning rate $\eta > 0$ is a parameter of the algorithm
    - It is usually gradually decreased during the training

# Training: Advanced methods

- In its classical form, backpropagation utilises gradient descent

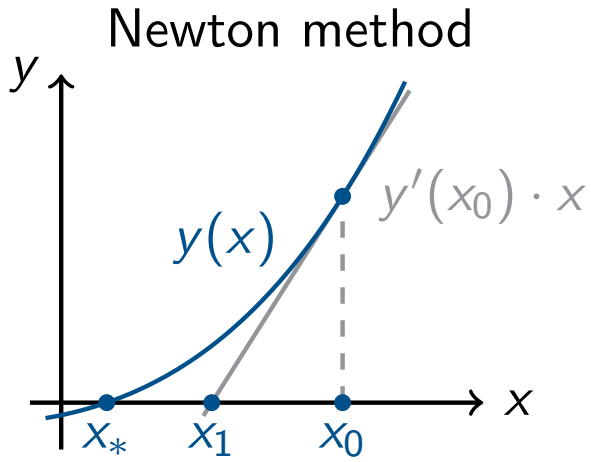
# Training: Advanced methods

- In its classical form, backpropagation utilises gradient descent
- There are better alternatives, for instance:
  - Conjugate gradient method
    - Avoids the zig-zag pattern



# Training: Advanced methods

- In its classical form, backpropagation utilises gradient descent
- There are better alternatives, for instance:
  - Conjugate gradient method
    - Avoids the zig-zag pattern
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
    - Quasi-Newton method (exploits approximated Hessian)



# Training: Advanced methods

- In its classical form, backpropagation utilises gradient descent
- There are better alternatives, for instance:
  - Conjugate gradient method
    - Avoids the zig-zag pattern
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
    - Quasi-Newton method (exploits approximated Hessian)
  - Single step of each algorithm is more expensive, but they take fewer steps
    - Though, sometimes the trade-off can be such that the gradient descent wins

# Training: Advanced methods

Local minimum

- In its classical form, backpropagation utilises gradient descent
- There are better alternatives, for instance:
  - Conjugate gradient method
    - Avoids the zig-zag pattern
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
    - Quasi-Newton method (exploits approximated Hessian)
  - Single step of each algorithm is more expensive, but they take fewer steps
    - Though, sometimes the trade-off can be such that the gradient descent wins
- There are algorithms that ignore the gradient completely and search for the global minimum
  - Simulated annealing, genetic algorithms, and others

# Training: Advanced methods

- In its classical form, backpropagation utilises gradient descent
- There are better alternatives, for instance:
  - Conjugate gradient method
    - Avoids the zig-zag pattern
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
    - Quasi-Newton method (exploits approximated Hessian)
  - Single step of each algorithm is more expensive, but they take fewer steps
    - Though, sometimes the trade-off can be such that the gradient descent wins
- There are algorithms that ignore the gradient completely and search for the global minimum
  - Simulated annealing, genetic algorithms, and others
- Other approaches consider many (suboptimal) points in the **w** space
  - Bayesian NNs build a posterior distribution in **w** and sample from it

# Training: Sampling of examples

- If weights are adjusted in small steps (like in classical backpropagation), the loss function $E(\mathbf{w})$ can be calculated also on a subset $T$ of examples available for training

$$\mathbf{w} \to \mathbf{w}' = \mathbf{w} - \eta \nabla E, \quad E(\mathbf{w}) = \frac{1}{N_T} \sum_{i \in T} E_1(i, \mathbf{w})$$

Single-example loss function

# Training: Sampling of examples

- If weights are adjusted in small steps (like in classical backpropagation), the loss function $E(\mathbf{w})$ can be calculated also on a subset $T$ of examples available for training

$$\mathbf{w} \to \mathbf{w}' = \mathbf{w} - \eta \nabla E, \quad E(\mathbf{w}) = \frac{1}{N_T} \sum_{i \in T} E_1(i, \mathbf{w})$$

  - Sequential training: $E(\mathbf{w})$ is calculated from a single example ($N_T = 1$)
    - At each epoch the example is chosen randomly
    - Allows online training, when examples arrive at the time of training
    - If the examples evolve over time, the NN will be adapting to them

# Training: Sampling of examples

- If weights are adjusted in small steps (like in classical backpropagation), the loss function $E(\mathbf{w})$ can be calculated also on a subset $T$ of examples available for training

$$\mathbf{w} \to \mathbf{w}' = \mathbf{w} - \eta \nabla E, \quad E(\mathbf{w}) = \frac{1}{N_T} \sum_{i \in T} E_1(i, \mathbf{w})$$

  - Sequential training: $E(\mathbf{w})$ is calculated from a single example $(N_T = 1)$
    - At each epoch the example is chosen randomly
    - Allows online training, when examples arrive at the time of training
    - If the examples evolve over time, the NN will be adapting to them
  - Training with sampling: a fraction of all examples is used $(1 < N_T < N)$
    - The subset $T$ can be chosen randomly at each epoch
    - Alternatively, examples in which NN is mistaken most can be prioritised
    - The method is useful for very large training sets

# Training: Sampling of examples

- If weights are adjusted in small steps (like in classical backpropagation), the loss function $E(\mathbf{w})$ can be calculated also on a subset $T$ of examples available for training

$$\mathbf{w} \to \mathbf{w}' = \mathbf{w} - \eta \nabla E, \quad E(\mathbf{w}) = \frac{1}{N_T} \sum_{i \in T} E_1(i, \mathbf{w})$$

  - Sequential training: $E(\mathbf{w})$ is calculated from a single example $(N_T = 1)$
    - At each epoch the example is chosen randomly
    - Allows online training, when examples arrive at the time of training
    - If the examples evolve over time, the NN will be adapting to them
  - Training with sampling: a fraction of all examples is used $(1 < N_T < N)$
    - The subset $T$ can be chosen randomly at each epoch
    - Alternatively, examples in which NN is mistaken most can be prioritised
    - The method is useful for very large training sets
  - Batch training: all training examples are utilised $(N_T = N)$
    - Most robust method

# Training: Sampling of examples

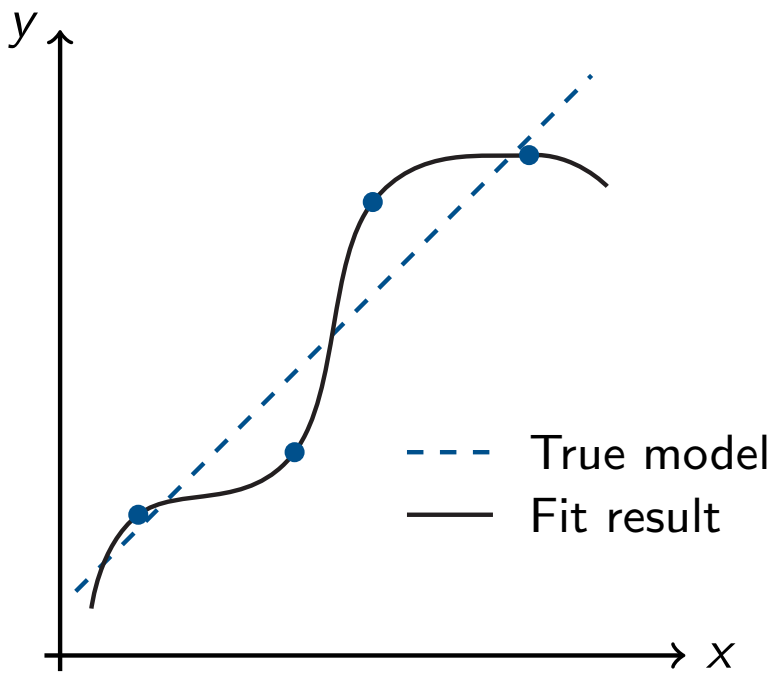
- If weights are adjusted in small steps (like in classical backpropagation), the loss function $E(\mathbf{w})$ can be calculated also on a subset $T$ of examples available for training

$$\mathbf{w} \to \mathbf{w}' = \mathbf{w} - \eta \nabla E, \quad E(\mathbf{w}) = \frac{1}{N_T} \sum_{i \in T} E_1(i, \mathbf{w})$$

  - Sequential training: $E(\mathbf{w})$ is calculated from a single example $(N_T = 1)$
    - At each epoch the example is chosen randomly
    - Allows online training, when examples arrive at the time of training
    - If the examples evolve over time, the NN will be adapting to them
  - Training with sampling: a fraction of all examples is used $(1 < N_T < N)$
    - The subset $T$ can be chosen randomly at each epoch
    - Alternatively, examples in which NN is mistaken most can be prioritised
    - The method is useful for very large training sets
  - Batch training: all training examples are utilised $(N_T = N)$
    - Most robust method
- For all other methods the batch training is strongly preferred

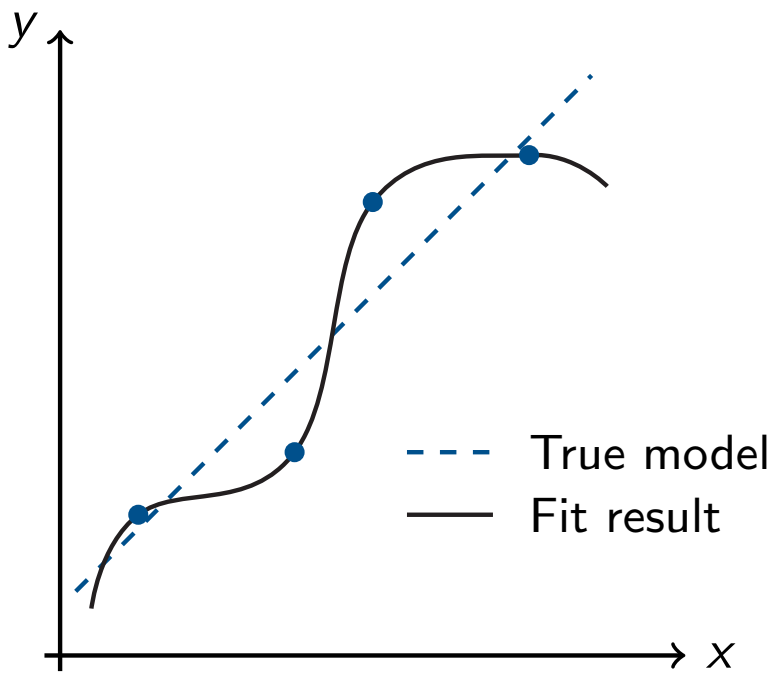
# Overfitting

- Overfitting occurs when a model (e. g. an NN) tries to describe noise in data rather than the desired underlying relationship
  - Results in unrealistically small error on examples used for training
  - Causes poor generalisation



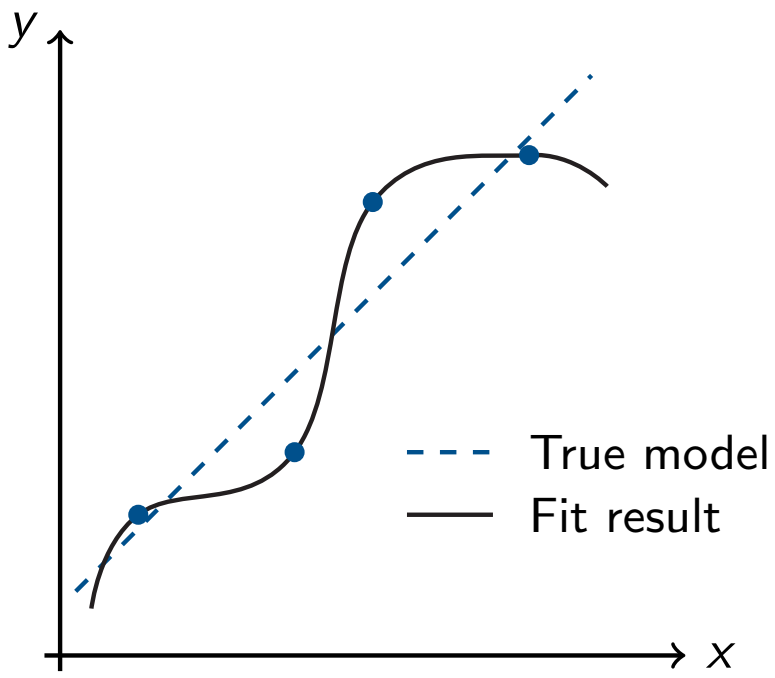
# Overfitting

- Overfitting occurs when a model (e. g. an NN) tries to describe noise in data rather than the desired underlying relationship
  - Results in unrealistically small error on examples used for training
  - Causes poor generalisation
- Usually indicates that the model is too complex for the problem
  - The large number of free parameters allows to "memorise" the training data



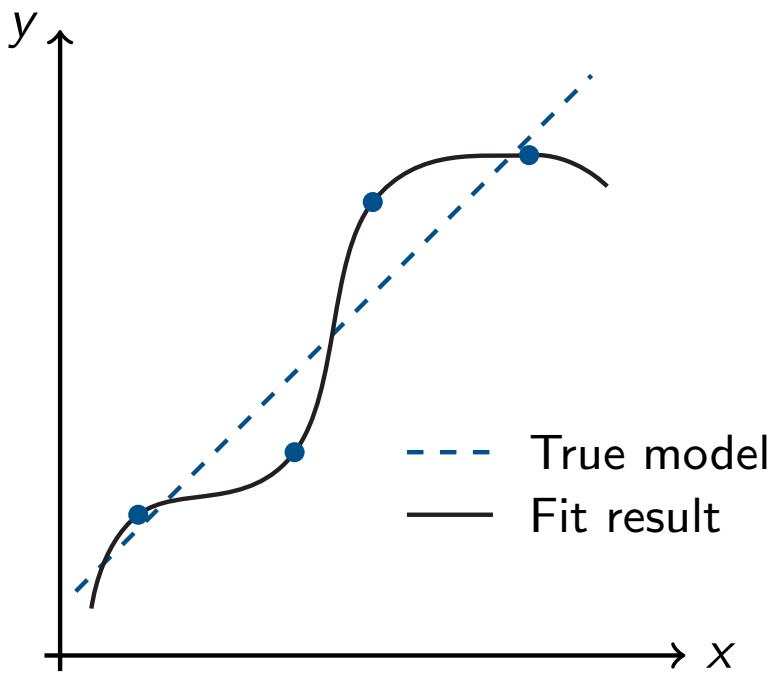
# Overfitting

- Overfitting occurs when a model (e. g. an NN) tries to describe noise in data rather than the desired underlying relationship
  - Results in unrealistically small error on examples used for training
  - Causes poor generalisation
- Usually indicates that the model is too complex for the problem
  - The large number of free parameters allows to "memorise" the training data
- A robust way to address the problem is to split available examples in two sets:
  - Training set is used to train the NN and optimise its architecture
  - Testing set consists of examples never seen by the NN before and is used to evaluate its performance



# Overfitting

- Overfitting occurs when a model (e. g. an NN) tries to describe noise in data rather than the desired underlying relationship
  - Results in unrealistically small error on examples used for training
  - Causes poor generalisation
- Usually indicates that the model is too complex for the problem
  - The large number of free parameters allows to "memorise" the training data
- A robust way to address the problem is to split available examples in two sets:
  - Training set is used to train the NN and optimise its architecture
  - Testing set consists of examples never seen by the NN before and is used to evaluate its performance



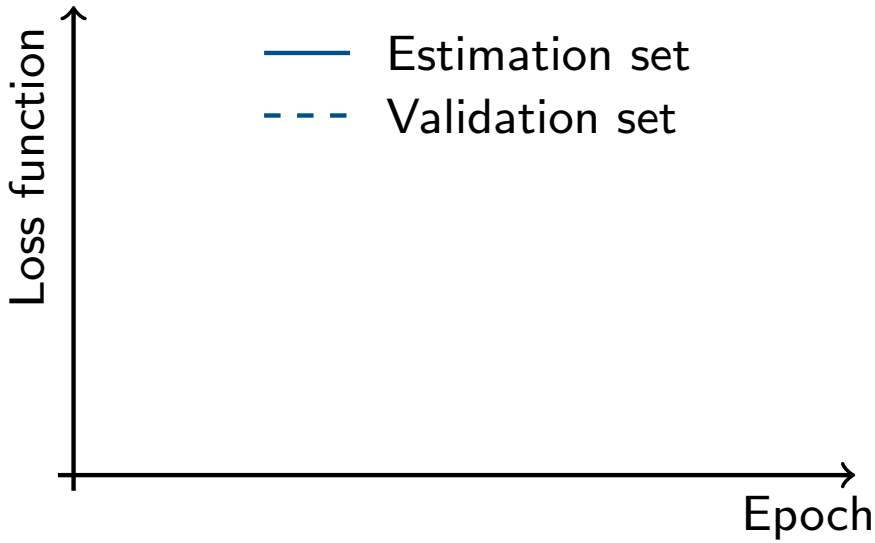

- One must never measure the performance with events utilised in training

# Overfitting: Early stopping

- Method of early stopping is a popular way to control overfitting
  - Applicable to iterative training algorithms only

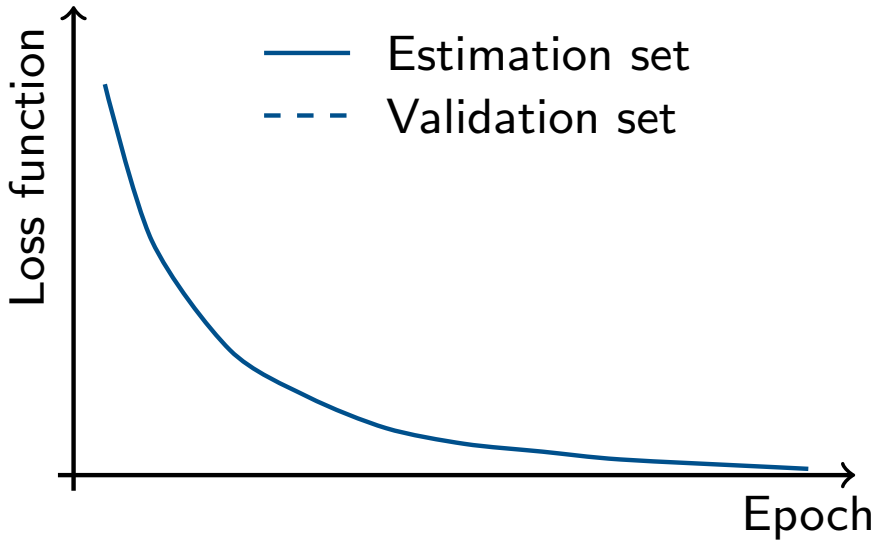
# Overfitting: Early stopping

- Method of early stopping is a popular way to control overfitting
  - Applicable to iterative training algorithms only
- Training set is split in two:
  - Estimation set is used to perform the actual training
  - Every few epochs the loss function is evaluated on validation set



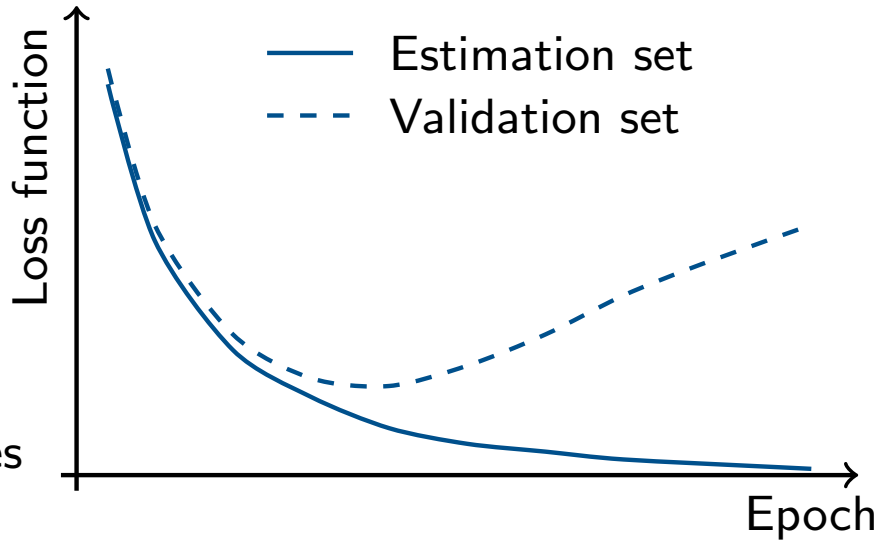
# Overfitting: Early stopping

- Method of early stopping is a popular way to control overfitting
  - Applicable to iterative training algorithms only
- Training set is split in two:
  - Estimation set is used to perform the actual training
  - Every few epochs the loss function is evaluated on validation set
- Behaviour of the loss function:
  - Descreases on the estimation set



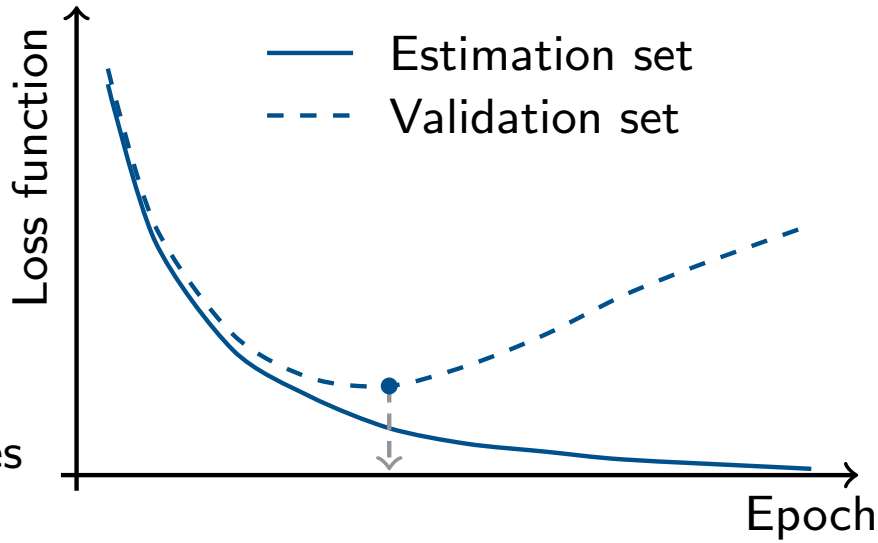
# Overfitting: Early stopping

- Method of early stopping is a popular way to control overfitting
  - Applicable to iterative training algorithms only
- Training set is split in two:
  - Estimation set is used to perform the actual training
  - Every few epochs the loss function is evaluated on validation set
- Behaviour of the loss function:
  - Descreases on the estimation set
  - On the validation set it first decreases but at some point starts to grow



# Overfitting: Early stopping

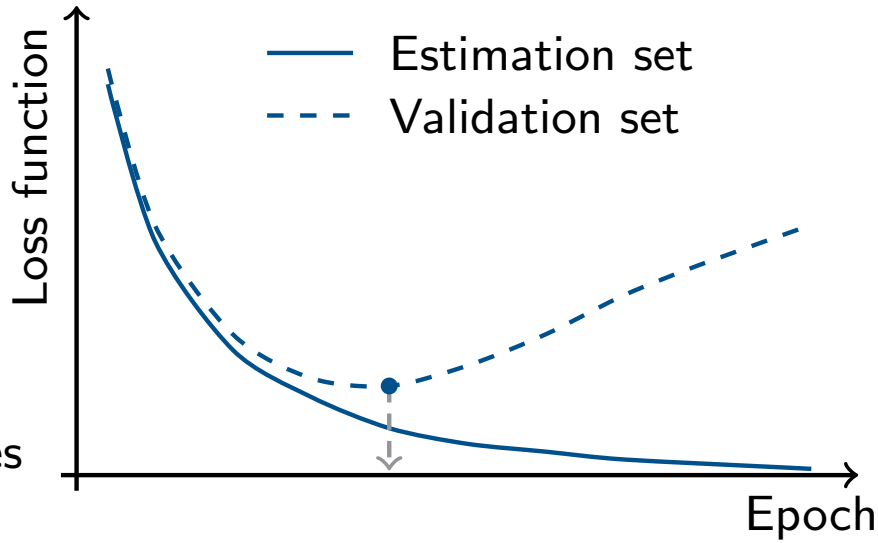
- Method of early stopping is a popular way to control overfitting
  - Applicable to iterative training algorithms only
- Training set is split in two:
  - Estimation set is used to perform the actual training
  - Every few epochs the loss function is evaluated on validation set
- Behaviour of the loss function:
  - Descreases on the estimation set
  - On the validation set it first decreases but at some point starts to grow




- The training is stopped when the loss on the validation set starts to grow
  - In real life the behaviour might be more complex, with several local minima

# Overfitting: Early stopping

- Method of early stopping is a popular way to control overfitting
  - Applicable to iterative training algorithms only
- Training set is split in two:
  - Estimation set is used to perform the actual training
  - Every few epochs the loss function is evaluated on validation set
- Behaviour of the loss function:
  - Descreases on the estimation set
  - On the validation set it first decreases but at some point starts to grow




- The training is stopped when the loss on the validation set starts to grow
  - In real life the behaviour might be more complex, with several local minima
- The final performance is measured in the testing set

# Overfitting: Weight decay

- Overfitting can be mitigated by penalising complex models
  - A naive approach is to limit the number of neurons in hidden layer(s), but it limits the class of functions that can be approximated

# Overfitting: Weight decay

- Overfitting can be mitigated by penalising complex models
  - A naive approach is to limit the number of neurons in hidden layer(s), but it limits the class of functions that can be approximated
- In the weight decay method the loss function is changed to

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \alpha\,\mathbf{w}^2$$

  suppressing NNs with large values of weights
  - Technically, it is $L_2$ regularisation
  - Can be interpreted also as assigning Gaussian priors for weights

# Overfitting: Weight decay

- Overfitting can be mitigated by penalising complex models
  - A naive approach is to limit the number of neurons in hidden layer(s), but it limits the class of functions that can be approximated
- In the weight decay method the loss function is changed to

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \alpha\,\mathbf{w}^2$$

  suppressing NNs with large values of weights
  - Technically, it is $L_2$ regularisation
  - Can be interpreted also as assigning Gaussian priors for weights
- Exhaustive training is performed (until the end, no early stopping)

# Overfitting: Weight decay

- Overfitting can be mitigated by penalising complex models
  - A naive approach is to limit the number of neurons in hidden layer(s), but it limits the class of functions that can be approximated
- In the weight decay method the loss function is changed to

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \alpha \, \mathbf{w}^2$$

  suppressing NNs with large values of weights
  - Technically, it is $L_2$ regularisation
  - Can be interpreted also as assigning Gaussian priors for weights
- Exhaustive training is performed (until the end, no early stopping)
- As always, the performance is measured in the testing set
  - The regularisation mitigates overfitting but does not eliminate it completely

# Classification problem

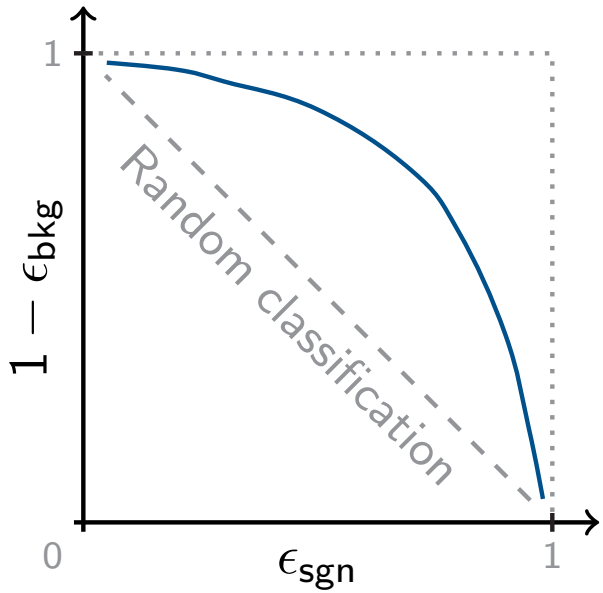
- NNs naturally perform regression

# Classification problem

- NNs naturally perform regression
- Binary classification can be viewed as a special case of regression
  - With desired response 1 for signal and 0 for background
  - In this case the NN will usually approximate purity $n_s(\mathbf{x})/(n_s(\mathbf{x}) + n_b(\mathbf{x}))$, where $n(\mathbf{x})$ is concentration in point $\mathbf{x}$
  - Cross-entropy is usually the preferred version of the loss function

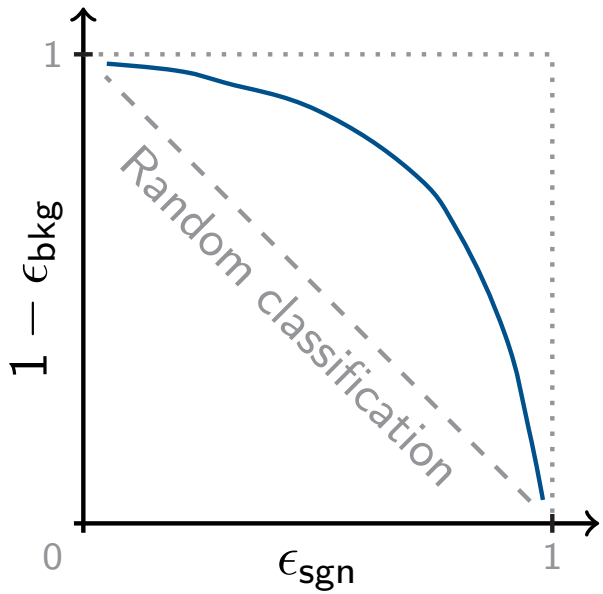
# Classification problem

- NNs naturally perform regression
- Binary classification can be viewed as a special case of regression
  - With desired response 1 for signal and 0 for background
  - In this case the NN will usually approximate purity $n_s(\mathbf{x})/(n_s(\mathbf{x}) + n_b(\mathbf{x}))$, where $n(\mathbf{x})$ is concentration in point $\mathbf{x}$
  - Cross-entropy is usually the preferred version of the loss function
- Performance of a binary discriminator is fully described by receiver operating characteristic (ROC curve)



# Classification problem

- NNs naturally perform regression
- Binary classification can be viewed as a special case of regression
  - With desired response 1 for signal and 0 for background
  - In this case the NN will usually approximate purity $n_s(\mathbf{x})/(n_s(\mathbf{x}) + n_b(\mathbf{x}))$, where $n(\mathbf{x})$ is concentration in point $\mathbf{x}$
  - Cross-entropy is usually the preferred version of the loss function
- Performance of a binary discriminator is fully described by receiver operating characteristic (ROC curve)
- For multiclass classification NNs with several outputs (one for each class) are utilised
  - But a combination of several binary discriminators can also be used



# Part II: Practical matters

# Inputs

- A neural network can approximate any function. . . in theory
  - Worst thing one could do is to feed an NN with components of 4-momenta

# Inputs

- A neural network can approximate any function. . . in theory
  - Worst thing one could do is to feed an NN with components of 4-momenta
- Input variables should reflect characteristic properties of your processes
  - E. g. invariant masses ($p_T$, $\eta$) for $s$-channel ($t$-channel) diagrams
  - Angular observables can be useful
  - Can profit from detector-level information, e. g. $b$-tagging
  - Characteristic observables for the background are also important

# Inputs

- A neural network can approximate any function... in theory
  - Worst thing one could do is to feed an NN with components of 4-momenta
- Input variables should reflect characteristic properties of your processes
  - E. g. invariant masses ($p_T$, $\eta$) for $s$-channel ($t$-channel) diagrams
  - Angular observables can be useful
  - Can profit from detector-level information, e. g. $b$-tagging
  - Characteristic observables for the background are also important
- Preprocess the inputs
  - Keep only necessary information (does the sign of $\eta$ matter?)
  - NNs do not like variables with long tails, transform them
    - Usually it is better to consider $\log(p_T)$ instead of $p_T$
  - (Linearly) normalise the inputs to a range $[-1, 1]$

# Inputs: Interdependencies

- Usage of neural networks is motivated if only there are significant dependencies among the inputs
  - Otherwise a product of 1D likelihoods will be a better option

# Inputs: Interdependencies

- Usage of neural networks is motivated if only there are significant dependencies among the inputs
  - Otherwise a product of 1D likelihoods will be a better option
- Try to reduce the interdependencies if possible
  - Linear correlations can be eliminated by diagonalisation of the covariance matrix (decorrelation) or with the principal component analysis

# Inputs: Interdependencies

- Usage of neural networks is motivated if only there are significant dependencies among the inputs
  - Otherwise a product of 1D likelihoods will be a better option
- Try to reduce the interdependencies if possible
  - Linear correlations can be eliminated by diagonalisation of the covariance matrix (decorrelation) or with the principal component analysis
- Some of inputs that strongly depend on each other could be dropped to reduce dimensionality of the problem
  - The dependence is usually non-linear, thus mutual information might be a better choice than the linear correlation factor

# Inputs: Interdependencies

- Usage of neural networks is motivated if only there are significant dependencies among the inputs
  - Otherwise a product of 1D likelihoods will be a better option
- Try to reduce the interdependencies if possible
  - Linear correlations can be eliminated by diagonalisation of the covariance matrix (decorrelation) or with the principal component analysis
- Some of inputs that strongly depend on each other could be dropped to reduce dimensionality of the problem
  - The dependence is usually non-linear, thus mutual information might be a better choice than the linear correlation factor
- If the final goal is to compare against experimental data, the interdependencies should be modelled correctly in MC
  - Although it is not as crucial as to check that NN response is modelled well

# Ranking of inputs

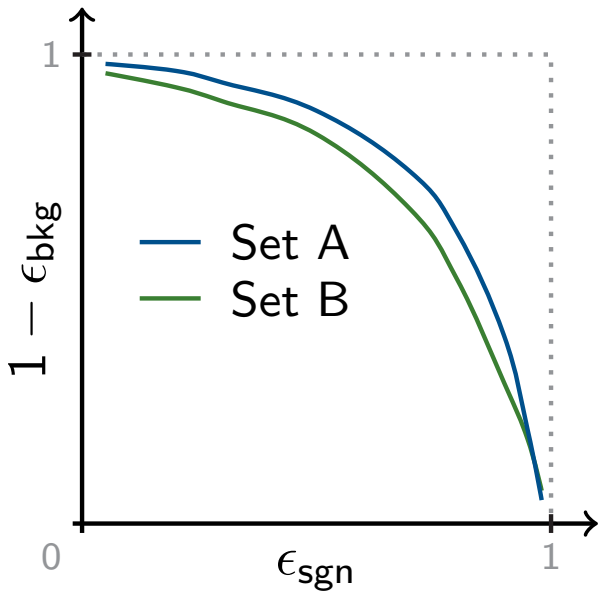
- You will be asked what are the most significant input variables
  - A number of heuristic approaches attempt to answer this question

# Ranking of inputs

- You will be asked what are the most significant input variables
  - A number of heuristic approaches attempt to answer this question
- But the question is ill-posed: in a multivariate problem not only individual inputs are important but also dependencies between them
  - It might happen that variables $x_1$ and $x_2$ are poor discriminators individually but their combination is very powerful

# Ranking of inputs

- You will be asked what are the most significant input variables
  - A number of heuristic approaches attempt to answer this question
- But the question is ill-posed: in a multivariate problem not only individual inputs are important but also dependencies between them
  - It might happen that variables $x_1$ and $x_2$ are poor discriminators individually but their combination is very powerful
- The robust approach is to study degradation in performance when a single input is removed from the reference set
  - A comparison of two sets of input variables is well-defined
  - Note that the significance of an input depends on the choice of reference set



# Training with multiple backgrounds

- Sometimes multiple backgrounds of similar importance might present

# Training with multiple backgrounds

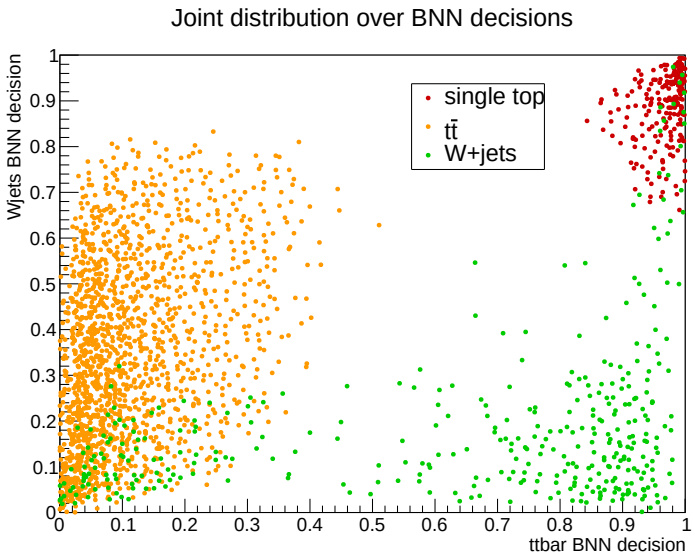
- Sometimes multiple backgrounds of similar importance might present
- Possible approaches:
  - Combine all backgrounds into one
    - The combined background will be more complex than each
    - Makes the classification problem harder

# Training with multiple backgrounds

- Sometimes multiple backgrounds of similar importance might present
- Possible approaches:
  - Combine all backgrounds into one
    - The combined background will be more complex than each
    - Makes the classification problem harder
  - Adopt multiclass classification
    - If the backgrounds share common features,
      the NN could exploit it

# Training with multiple backgrounds

- Sometimes multiple backgrounds of similar importance might present
- Possible approaches:
  - Combine all backgrounds into one
    - The combined background will be more complex than each
    - Makes the classification problem harder
  - Adopt multiclass classification
    - If the backgrounds share common features, the NN could exploit it
  - Train a set of one-vs-one discriminators
    - Dedicated NN to discriminate signal against each background
    - Base final decision responses of the NNs — a problem of a much smaller dimensionality



# Final words

- Neural networks are a powerful tool to solve multivariate classification and regression problems

# Final words

- Neural networks are a powerful tool to solve multivariate classification and regression problems
- As with every other advanced tool, there is a large room to tune it... but also to break it
  - Architecture, type of training, parameters of the optimisation algorithm

# Final words

- Neural networks are a powerful tool to solve multivariate classification and regression problems
- As with every other advanced tool, there is a large room to tune it... but also to break it
  - Architecture, type of training, parameters of the optimisation algorithm
- Neural networks do not do magic: you should try to simplify the problem as much as possible
  - Construct smart discriminative observables, avoid difficult shapes of inputs (e. g. tails), try to eliminate dependencies between them, reduce the dimensionality of the problem

# Final words

- Neural networks are a powerful tool to solve multivariate classification and regression problems
- As with every other advanced tool, there is a large room to tune it... but also to break it
  - Architecture, type of training, parameters of the optimisation algorithm
- Neural networks do not do magic: you should try to simplify the problem as much as possible
  - Construct smart discriminative observables, avoid difficult shapes of inputs (e. g. tails), try to eliminate dependencies between them, reduce the dimensionality of the problem
- After all, if you can perform your analysis without neural networks, do so
  - Or switch to a more interesting measurement