Ensemble learning

Ensembles

Voting classifiers

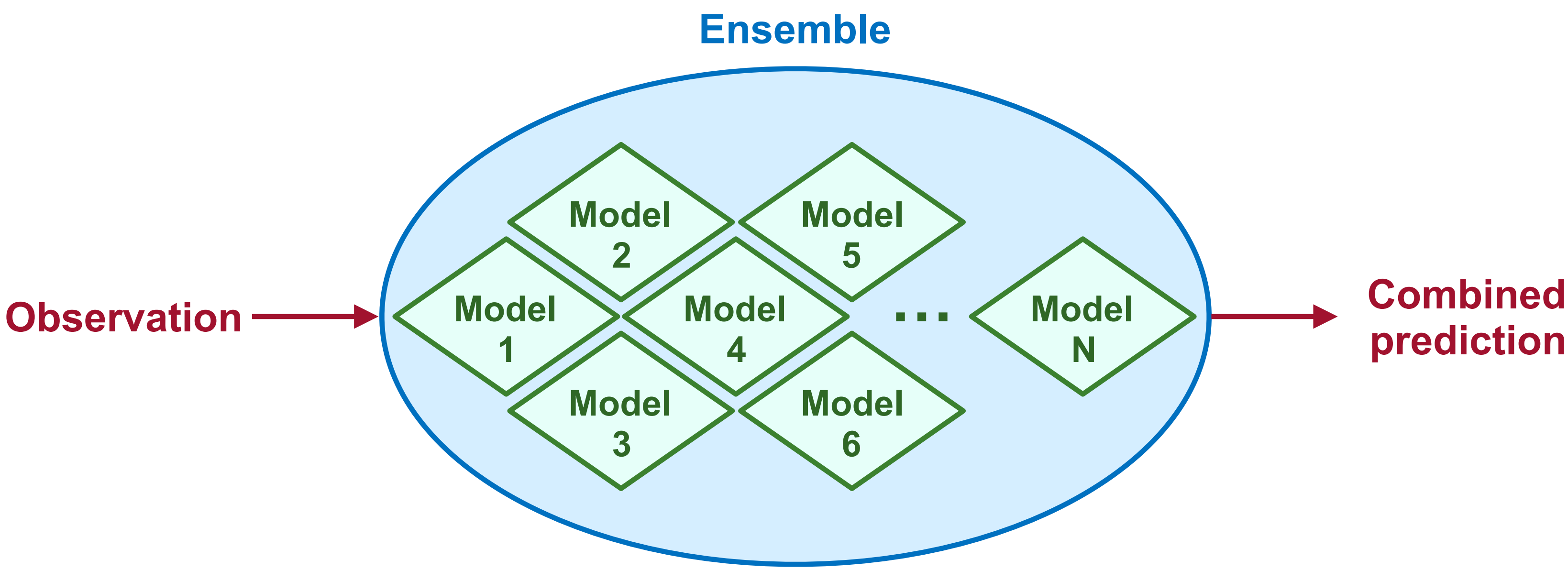
When do ensembles work ?

Ensemble learning (1/2)

- **No free launch theorem**
  - *There is no machine learning algorithm that is **always** superior to others*
- Which algorithm should we pick, then?
  - *Each technique learns a different model*
  - *Each model makes its “unique” mistakes*
- **Idea** ➜ Combine several models into a better one
  - *Single models make different mistakes and have a partial knowledge*
  - *The majority of them is less likely to make the same mistake*
  - *Thus, their combination can yield better decisions*

Ensemble learning (2/2)

- **Ensemble ➜** *A set of models whose individual predictions are combined in some way to make a better prediction*



Voting classifiers (1/2)

- **In theory…**
  - *Suppose you have **n** binary classifiers*
  - *Assume they are individually correct only **p = 51%** of the time*
  - *What is the probability that **the majority of them** is right ?*

$$\mathrm{Pr}(\text{voting}) = \sum_{k=\lfloor n/2 \rfloor + 1}^{n} \binom{n}{k} p^k (1-p)^{n-k}$$

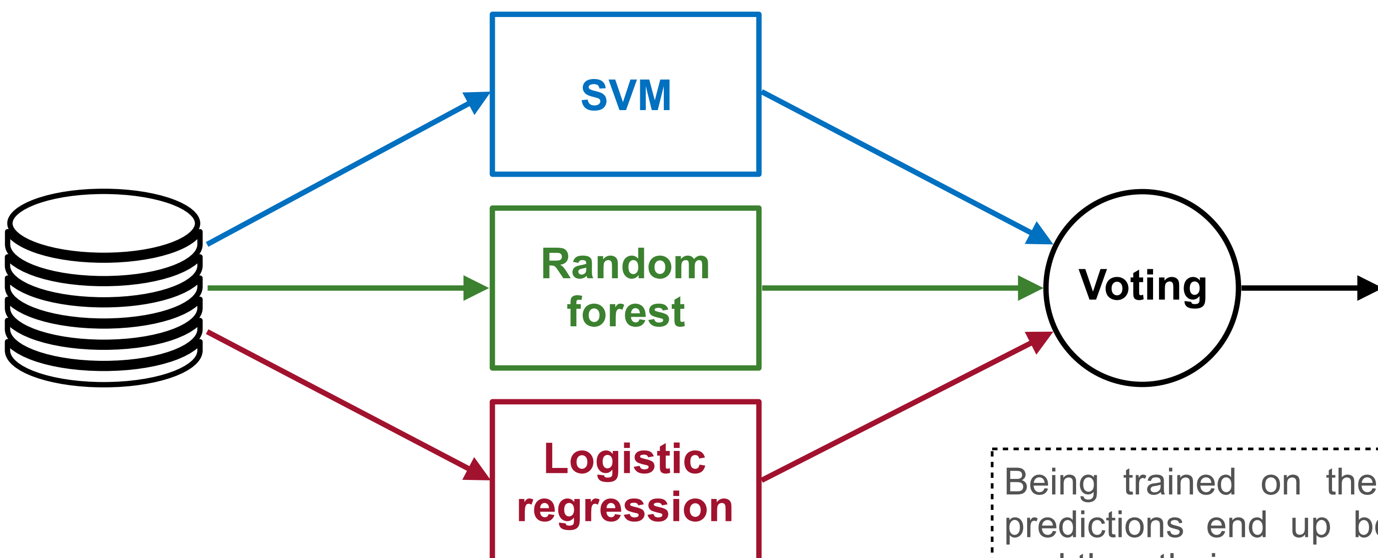
$$= 1 - \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{k} p^k (1-p)^{n-k}$$

n	1	1.000	10.000
Pr(voting)	51%	75%	97%

- ***Reality check*** ➜ *This is only true if the classifiers are **independent !!!***

Voting classifiers (2/2)

- **A first attempt…**
  - *A way to get diverse classifiers is to train them using different algorithms*
  - *This increases the chance they will make different types of errors*
  - *Their predictions are then aggregated by selecting the "most voted" class*



Being trained on the same data, the predictions end up being very similar, and thus their aggregation isn't effective.

When do ensembles work ?

- An ensemble is more accurate than its members **only if**
  - ***Accuracy*** ➜ *The members are better than guessing*
  - ***Diversity*** ➜ *The members make different types of errors on new data*

- How to make an ensemble work ?
  - *Use "simple models" for the individual members*
  - *Train the members in "different ways"*
  - *The goal is to make them as independent from one another as possible*

- **Problem** ➜ We only have **one** training set
  - *Ensembles methods provide strategies to circumvent this obstacle !!!*

Ensemble methods

Training the ensemble
Structuring the ensemble
Combining the ensemble

Ensemble methods

- Ensemble methods address the following questions

1) ***How to train the ensemble ?***
    - Bootstrapping
    - Feature sampling
    - Diversified learning

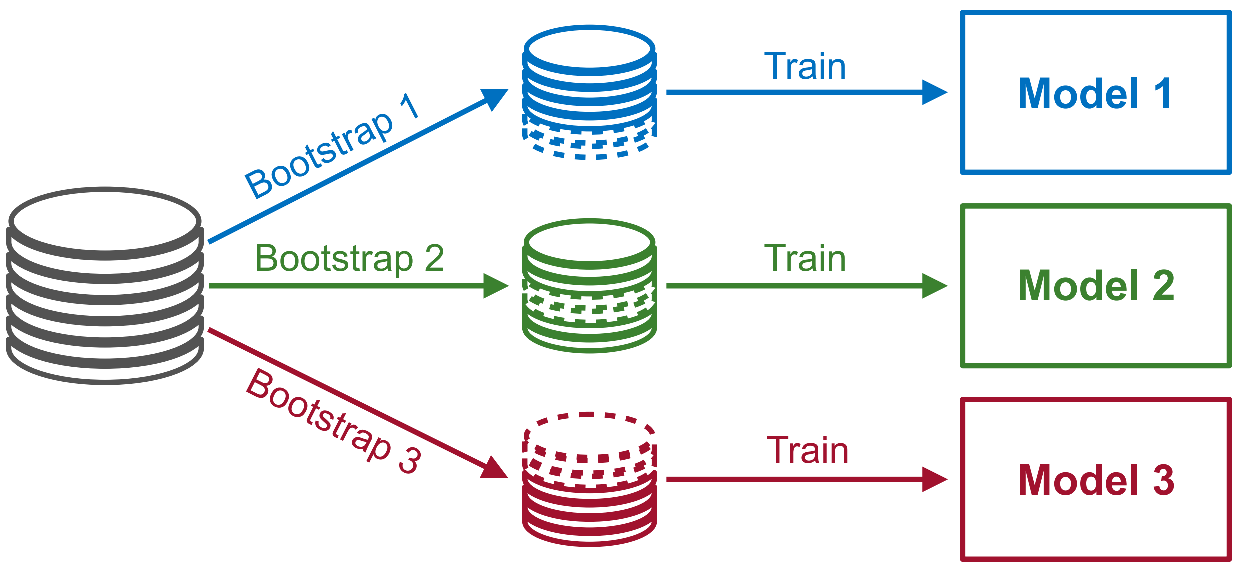
2) ***How to structure the ensemble ?***
    - Parallel
    - Cascading
    - Hierarchical

3) ***How to combine the ensemble ?***
    - Static rule
    - Stacking
    - End-to-end training

Training the ensemble (1/3)

1) Subsampling the training set

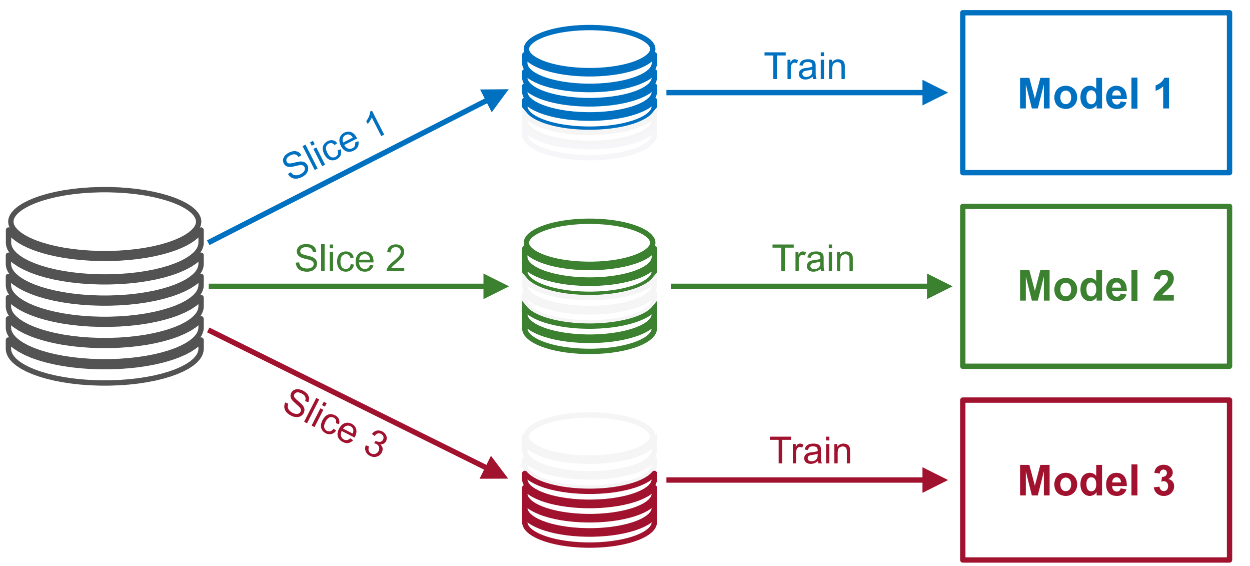
- *Multiple hypotheses are generated by training individual models on different datasets obtained by resampling a common training set*
- ***Used in*** ➜ *bagging, random forests*



Training the ensemble (2/3)

2) Subsampling the input features

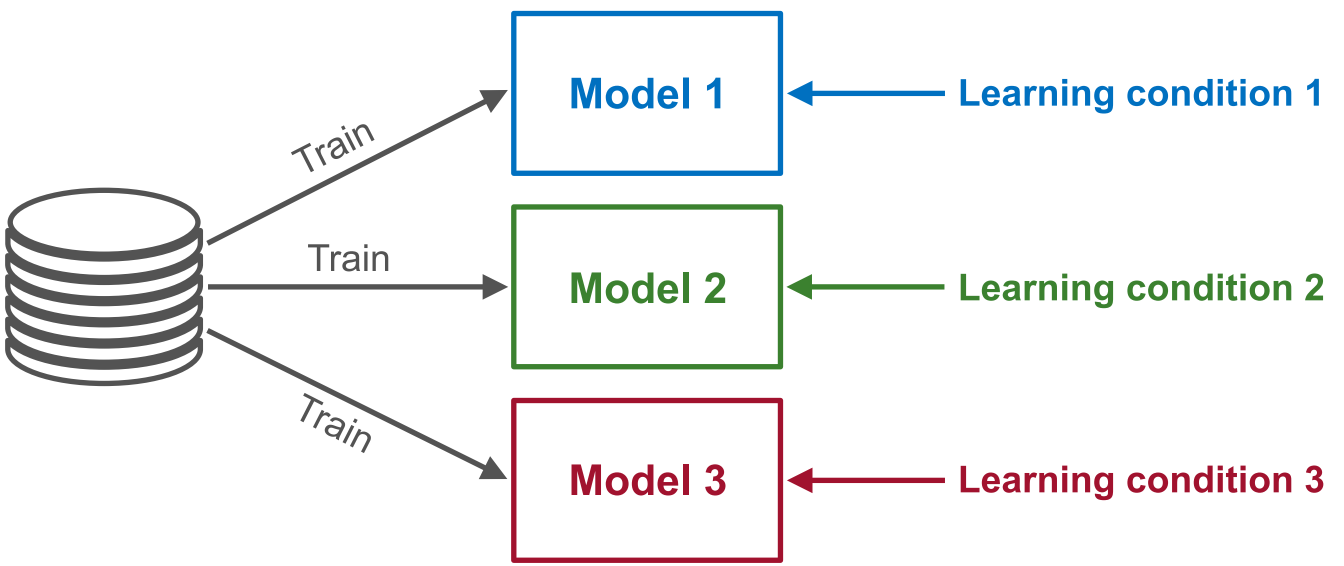
- *Multiple hypotheses are generated by training individual models on different (e.g., randomly generated) subsets of the input features*
- ***Used in* ➜ *random forests***



Training the ensemble (3/3)

3) Modifying the learning conditions

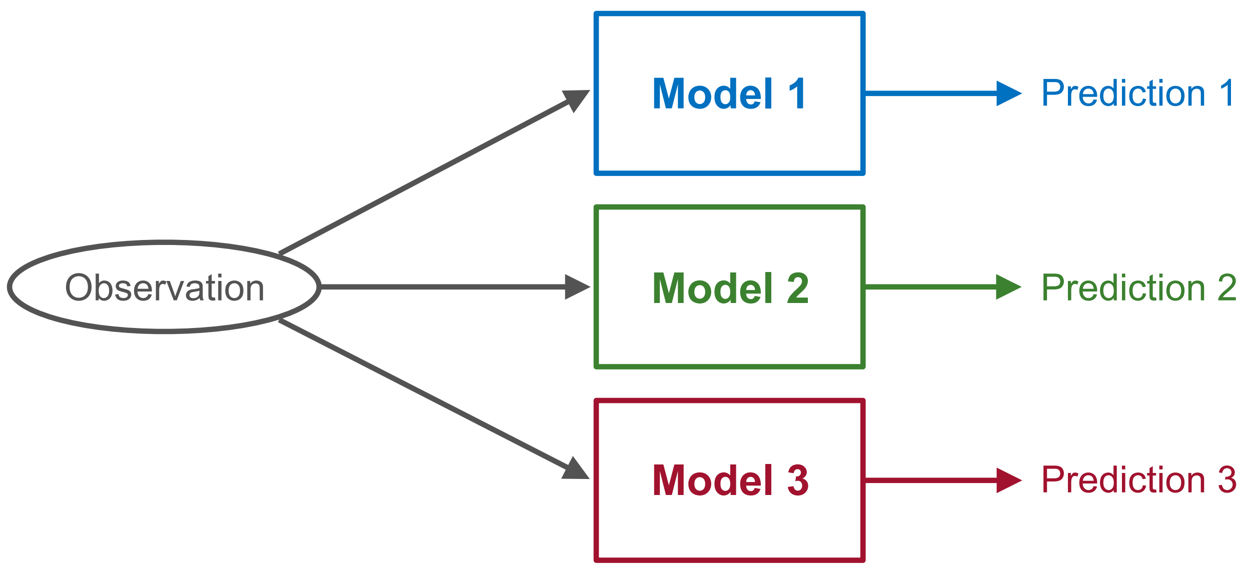
- *Multiple hypotheses are generated by training individual models in different learning conditions (e.g., random initializations, modified data, …)*
- ***Used in* ➜ *voting classifiers, boosting, neural networks***



Structuring the ensemble (1/2)

1) Parallel structure

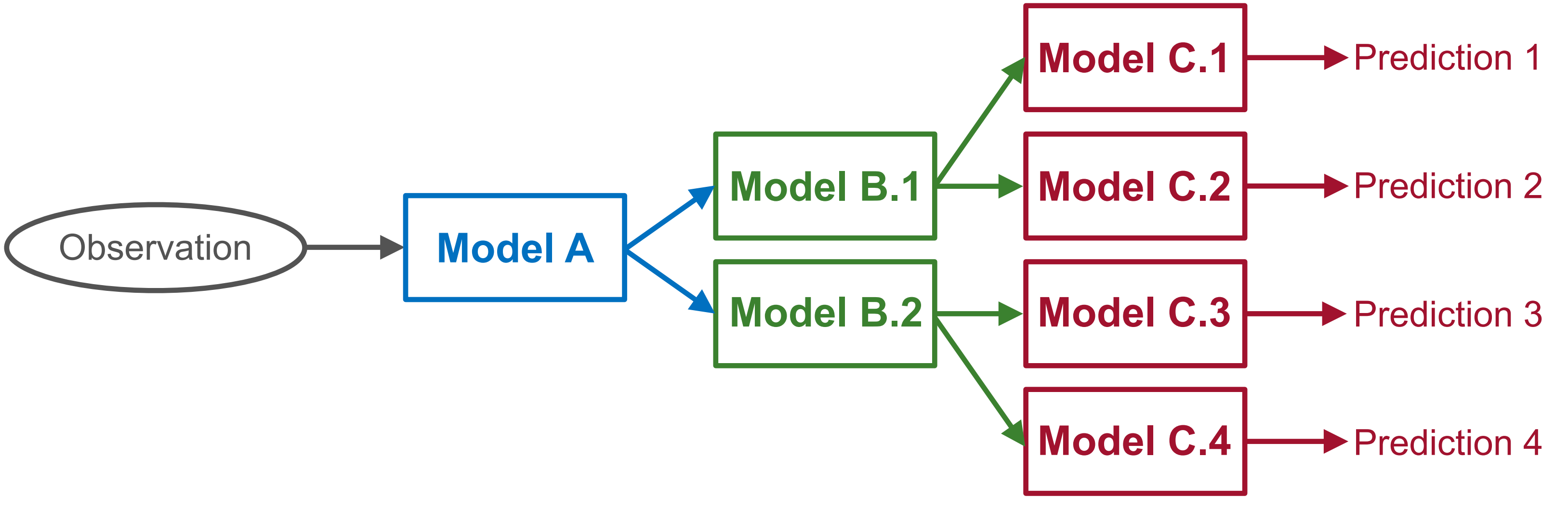
- *The individual models are invoked independently*
- ***Used in*** **➜** *voting classifiers, bagging, random forest, boosting*



Structuring the ensemble (2/2)

2) Cascading or hierarchical structure

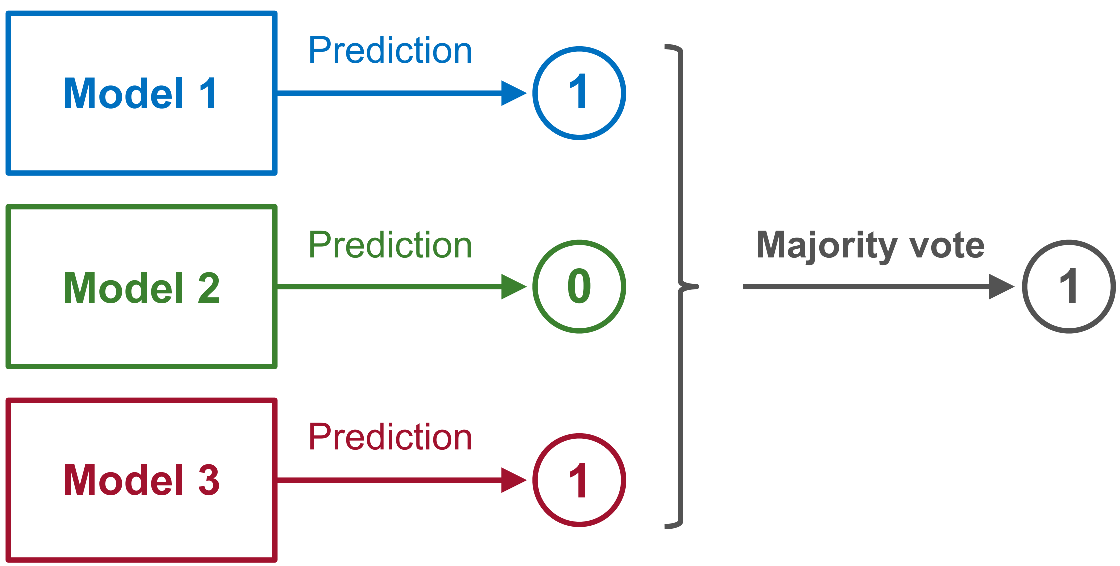
- *The individual models are invoked in a sequential or tree-based fashion*
- ***Used in*** **➜** *stacking, neural networks*



Combining the ensemble (1/3)

1) Static combination rule

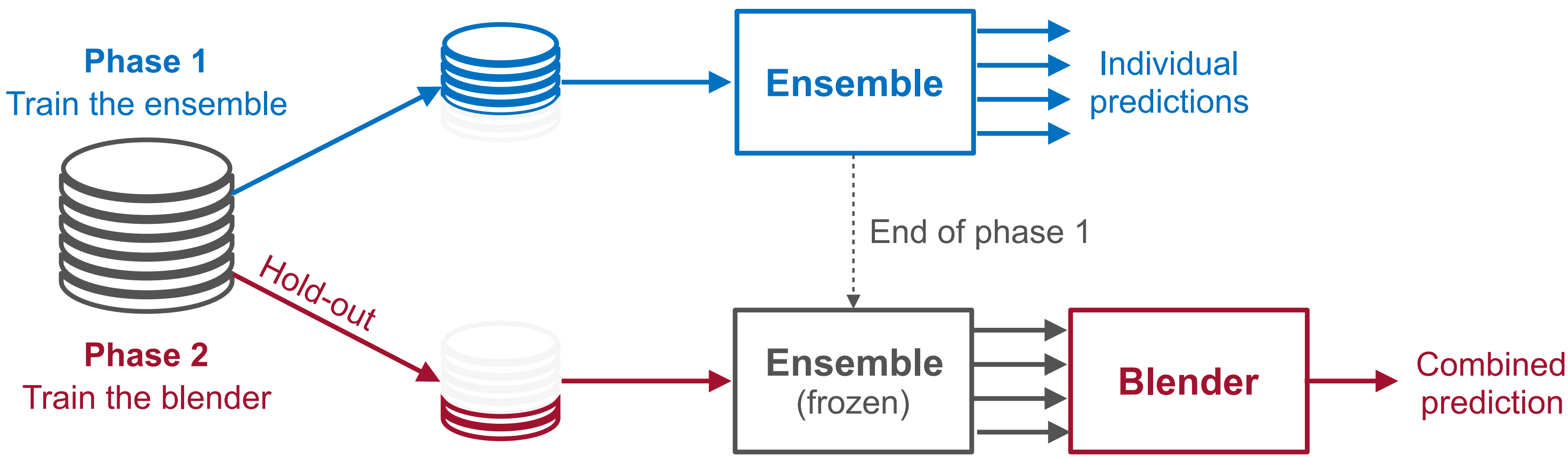
- *The rule only depends on the type of output produced by the members: voting (for labels), averaging (for estimates), or board count (for ranks)*
- ***Used in*** ➜ *voting classifiers, bagging, random forests*



Combining the ensemble (2/3)

2) Stacked generalization

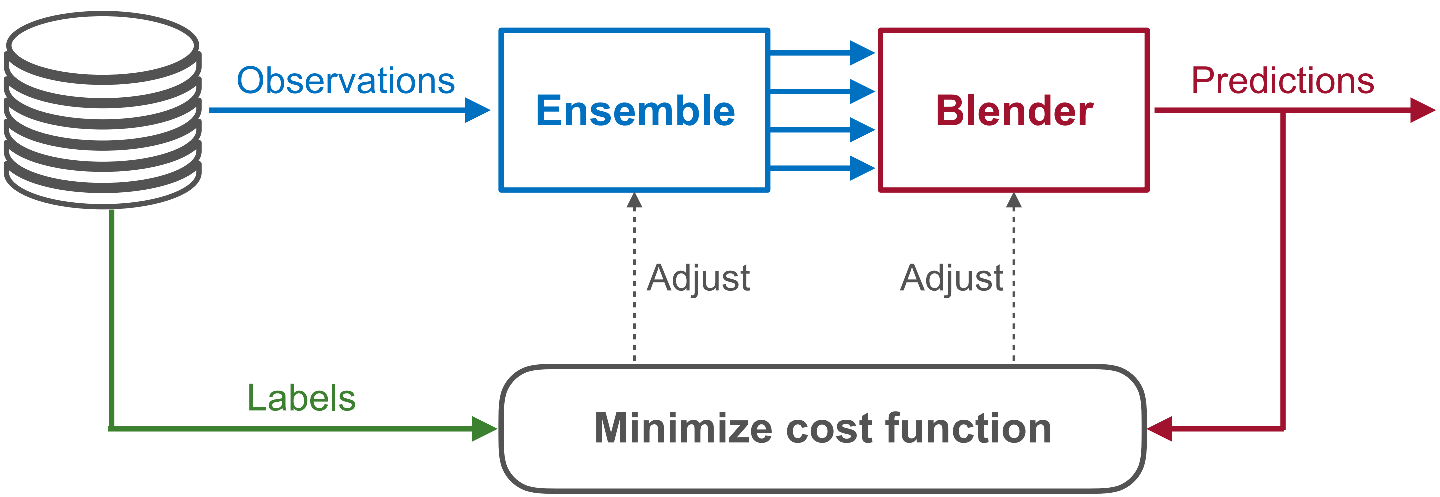
- *The member predictions serve as input features for the blender, which undergoes a separate training to learn how to combine its inputs*
- ***Used in ➜** boosting, stacking*



Combining the ensemble (3/3)

3) End-to-end training

- *The ensemble and the blender are trained jointly by minimizing a cost function through an iterative method (e.g., gradient descent)*
- ***Used in*** **➜** *neural networks*



Tree-based ensembles

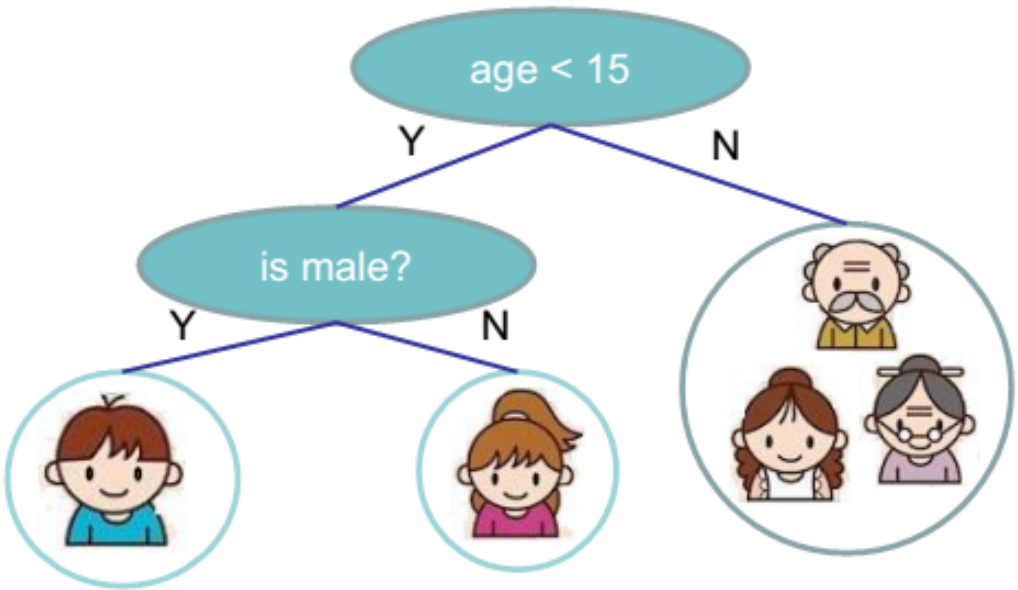
Random forest

Boosting

One-hot stacking

Decision trees (1/3)

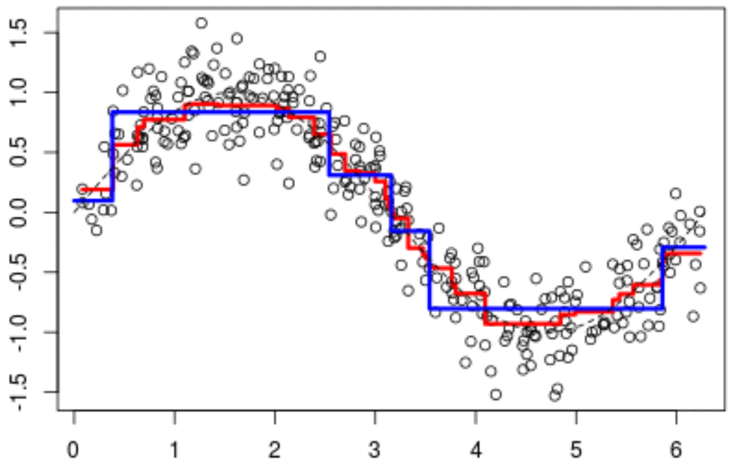
- A **decision tree** is a prediction model defined by
  - *… a hierarchy of simple binary rules*
  - *… an outcome for each sequence of binary decisions*



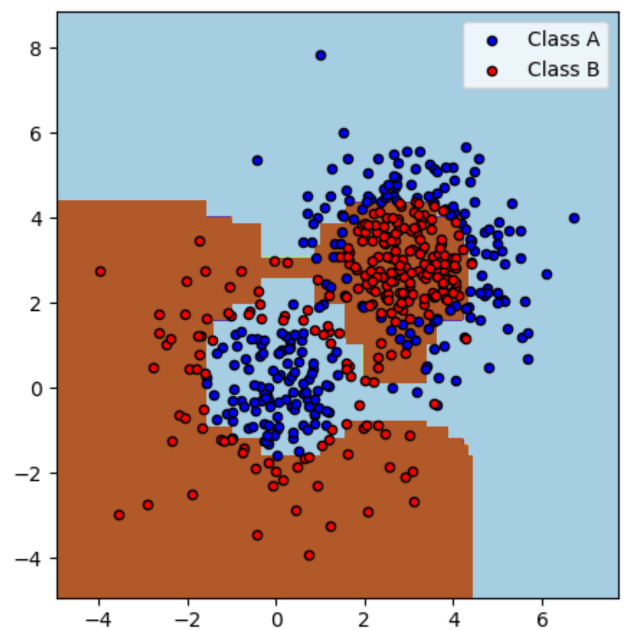
Decision trees (2/3)

- Decision trees are **nonlinear models**
  - *They yield a constant piecewise function*

Prediction of a ***regression tree***



Decision boundary of a ***classification tree***

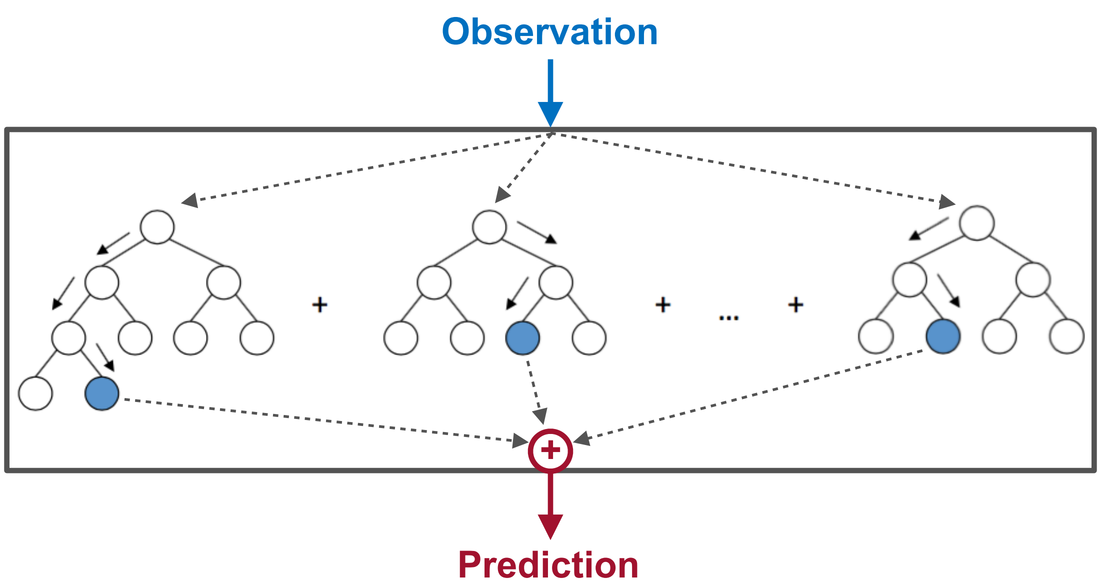


Decision trees (3/3)

- **Important limitations** (can be fixed via ensemble learning)
  - *Trees do not tend to be as accurate as other approaches*
  - *A small change in the training set can result in a completely different tree*
  - *Trees tend to quickly over-fit to the training data*
  - *Some patterns are hard to learn with trees*

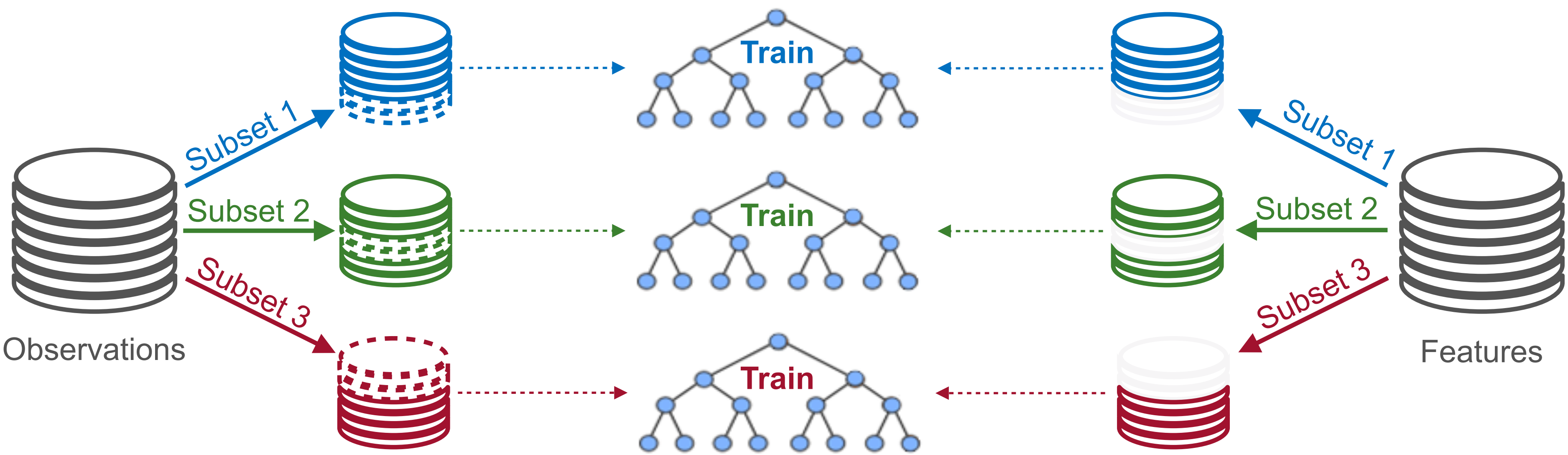
Random forest (1/3)

- A **random forest** is an ensemble of trees
  - ***Training*** ➜ *Bootstrapping + Feature sampling*
  - ***Structure*** ➜ *Parallel trees*
  - ***Combination*** ➜ *Static rule*



Random forest (2/3)

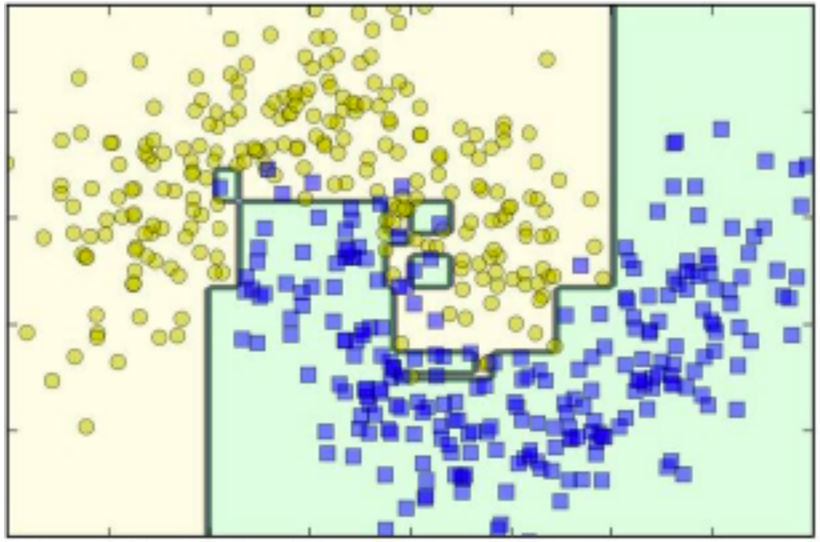
- The individual trees are **grown big**
  - *Each tree has **high variance** and **low bias***
  - *Their combination yields a **variance reduction***



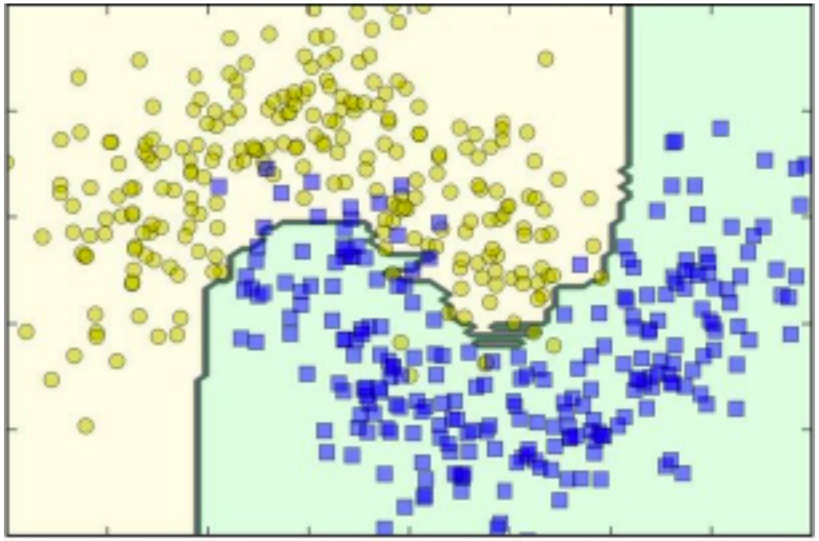
Random forest (3/3)

- Random forest generalizes better than a single tree
  - *Decision trees are highly sensible to over-fitting*
  - *Random forests tend to have lower variance*

Decision tree

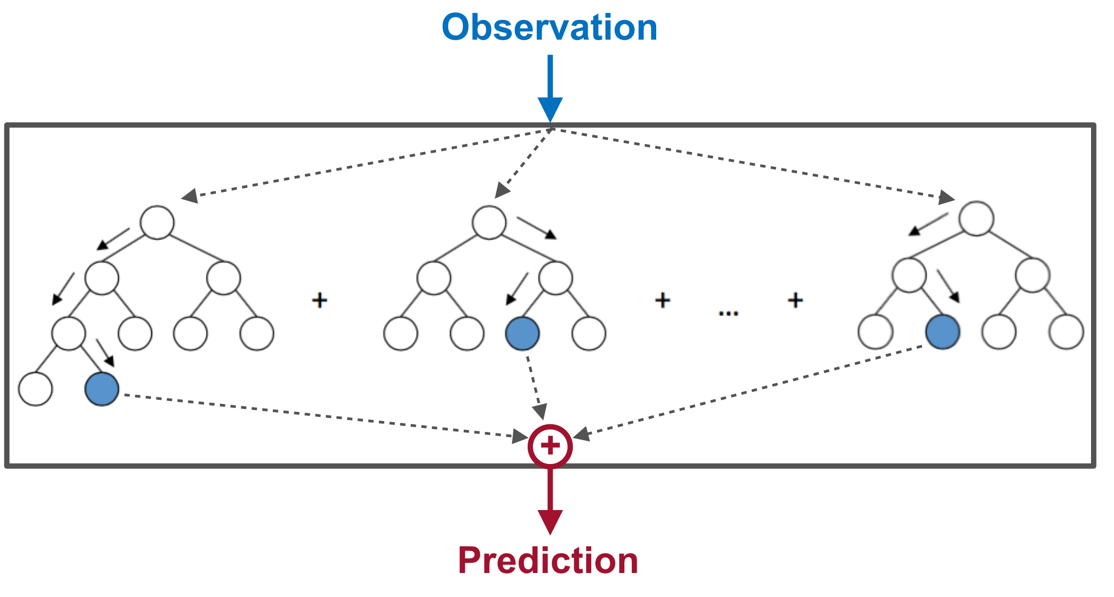


Random forest



Boosting (1/3)

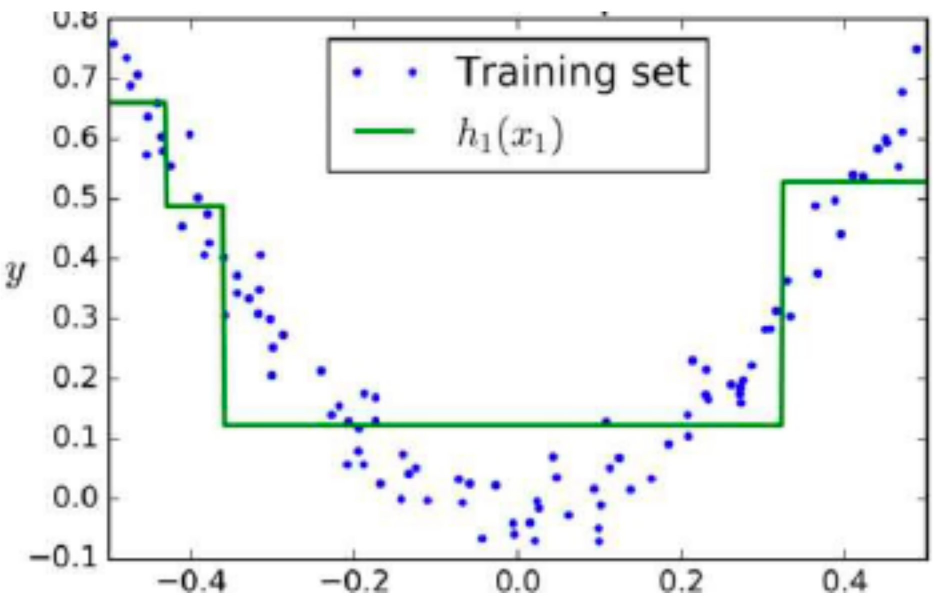
- **Boosting** is a way to build an ensemble of trees
  - ***Training* ➜ *Residual fitting***
  - ***Structure* ➜ *Parallel trees***
  - ***Combination* ➜ *Weighted average***



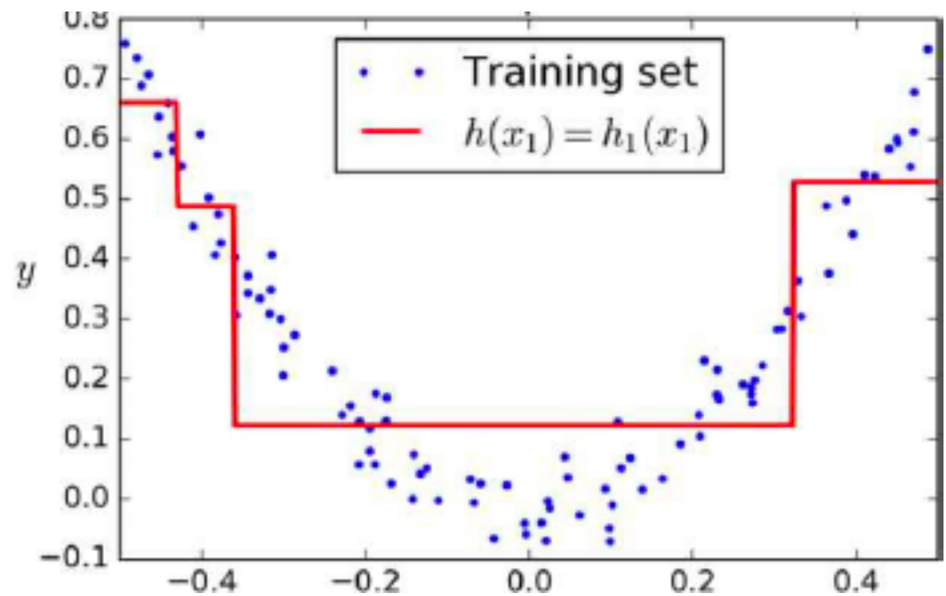
Boosting (2/3)

- **Residual fitting** ➜ Iteration 1
  - *The tree* $\boldsymbol{h_1(x^{(n)})}$ *is fitted to the original labels* $\boldsymbol{y^{(n)}}$

Residuals and tree predictions



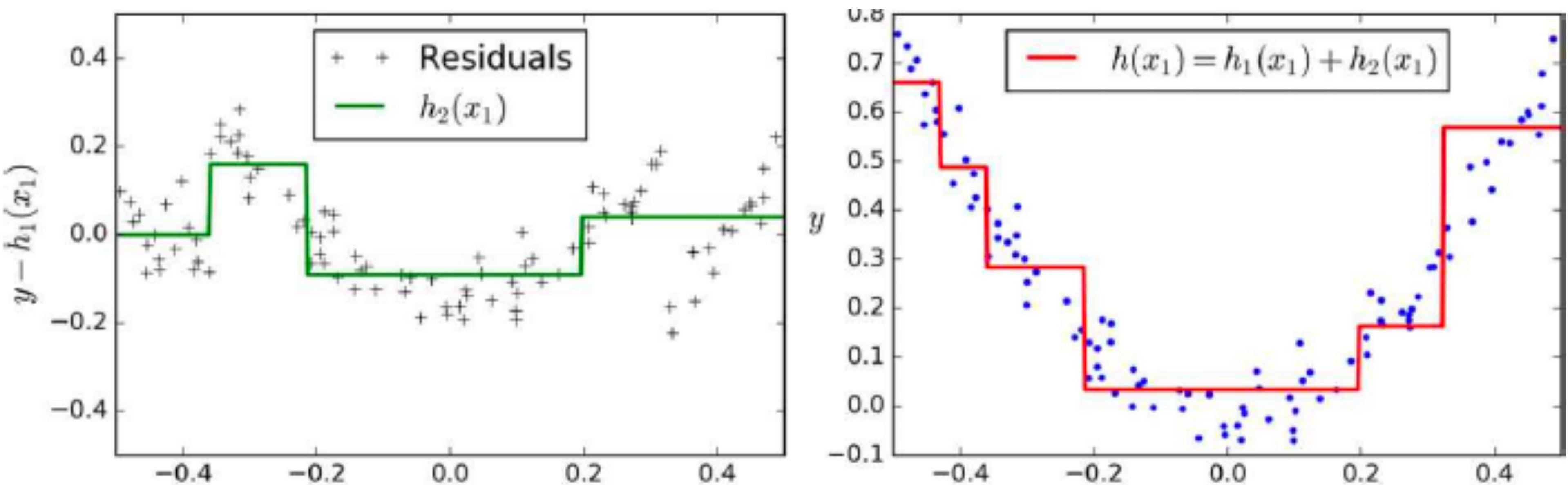
Ensemble predictions



Boosting (2/3)

- **Residual fitting** ➜ Iteration 2
  - *The tree* $\boldsymbol{h_2(x^{(n)})}$ *is fitted to the residuals* $\boldsymbol{y^{(n)} - h_1(x^{(n)})}$

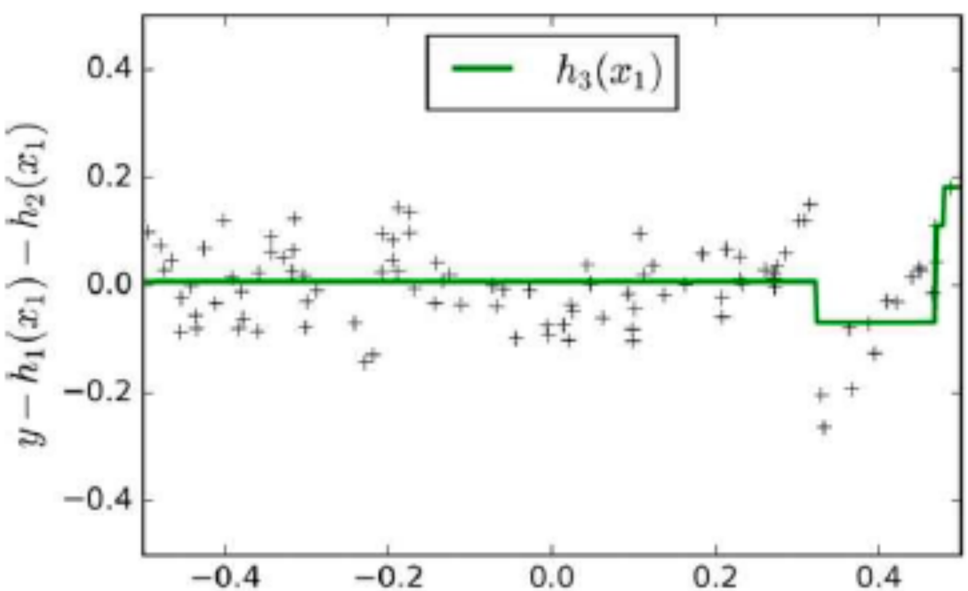
Residuals and tree predictions *Ensemble predictions*



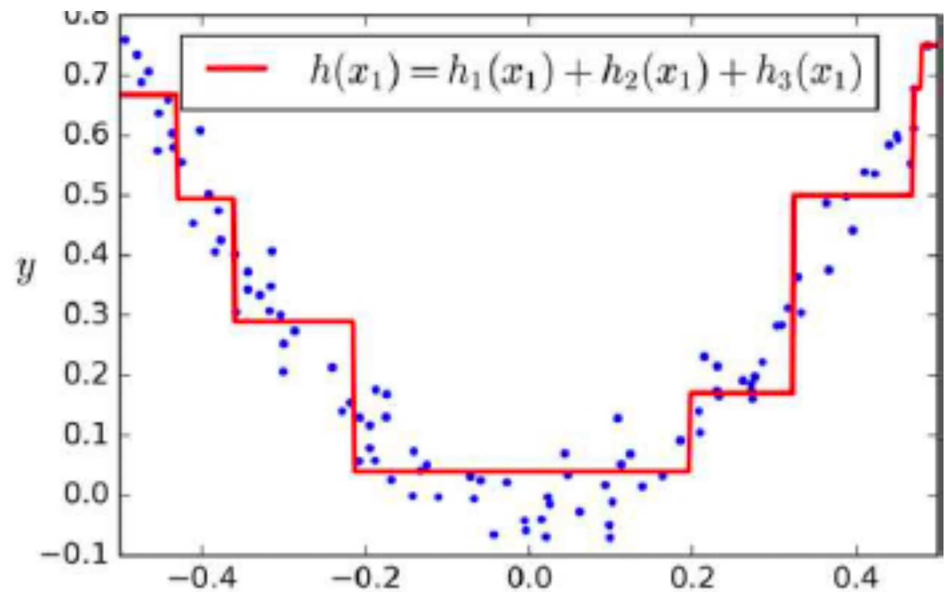
Boosting (2/3)

- **Residual fitting** ➜ Iteration 3
  - *The tree* $\boldsymbol{h_3(x^{(n)})}$ *is fitted to the residuals* $\boldsymbol{y^{(n)} - h_1(x^{(n)}) - h_2(x^{(n)})}$

Residuals and tree predictions

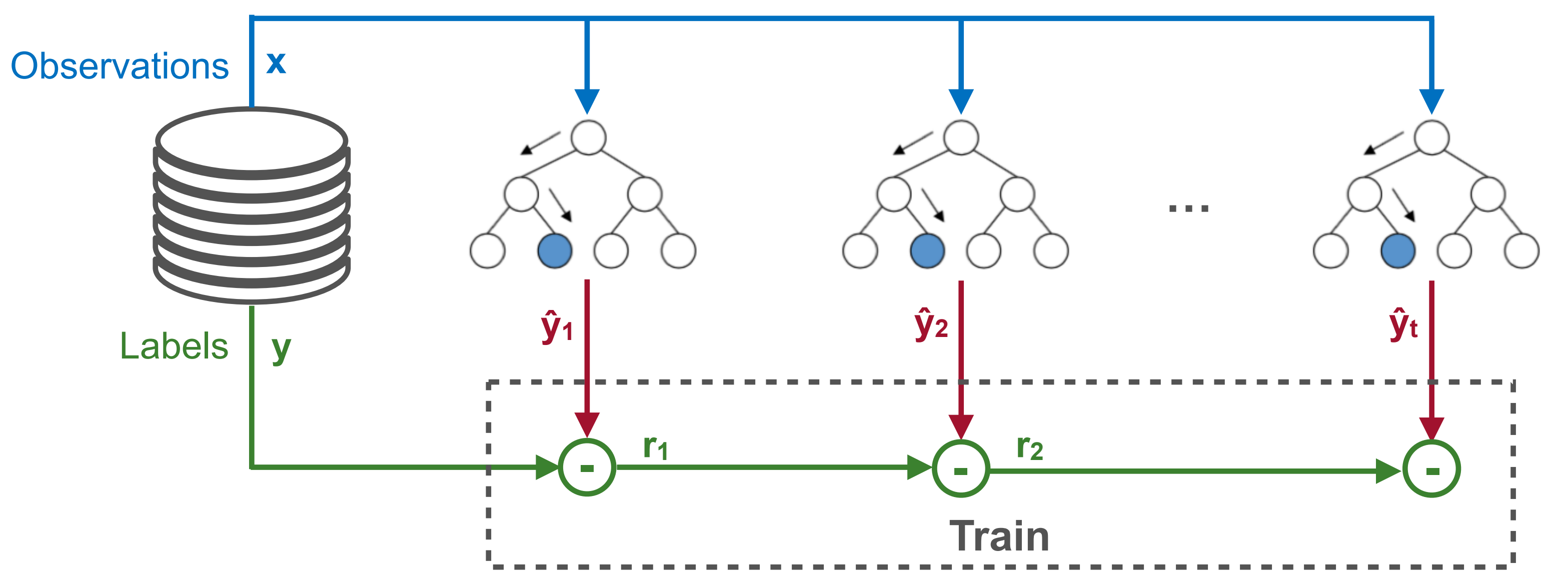


Ensemble predictions



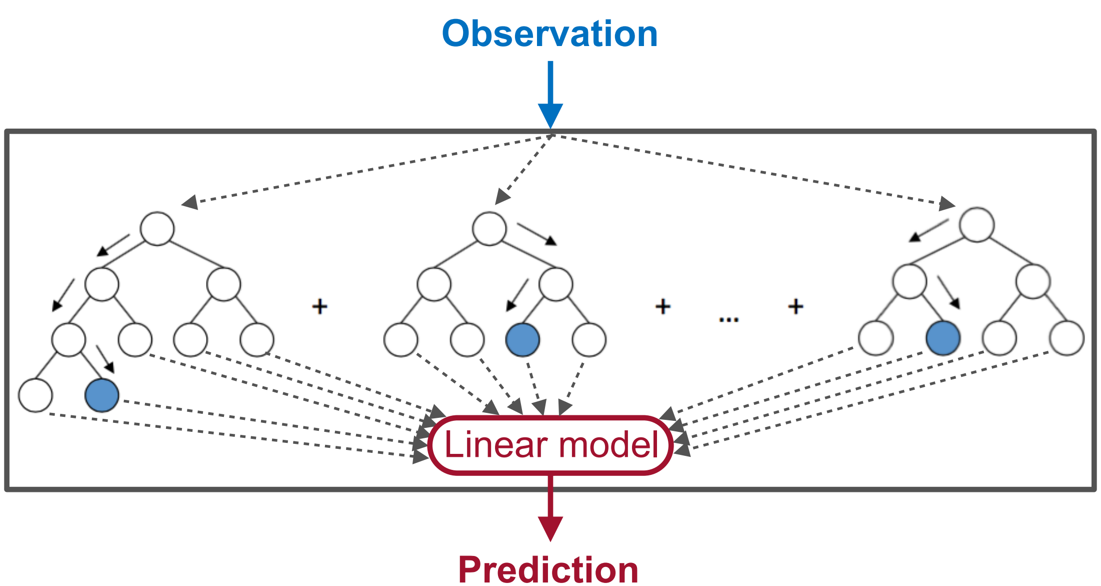
Boosting (3/3)

- The individual trees are **grown small**
  - *Each tree has **low variance** and **high bias***
  - *Their combination yields a **bias reduction***



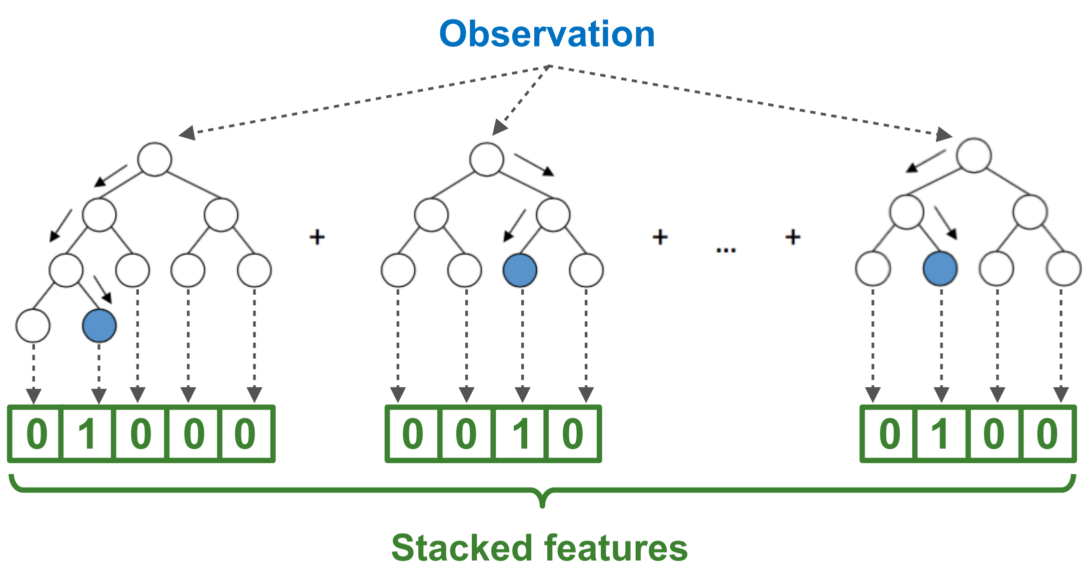
One-hot stacking (1/2)

- **One-hot stacking** is a way to build a hybrid ensemble
  - ***Training*** ➜ *Boosting + One-hot encoding*
  - ***Structure*** ➜ *Parallel trees + Linear model*
  - ***Combination*** ➜ *Stacking*



One-hot stacking (2/2)

- The stacked linear model is fed with…
  - *… the* ***one-hot encoded*** *leaves of each tree in the ensemble*
  - *… and not directly the ensemble tree predictions*
  - ***Remark*** ➜ *This works with binary classification trees*



Stacked generalization

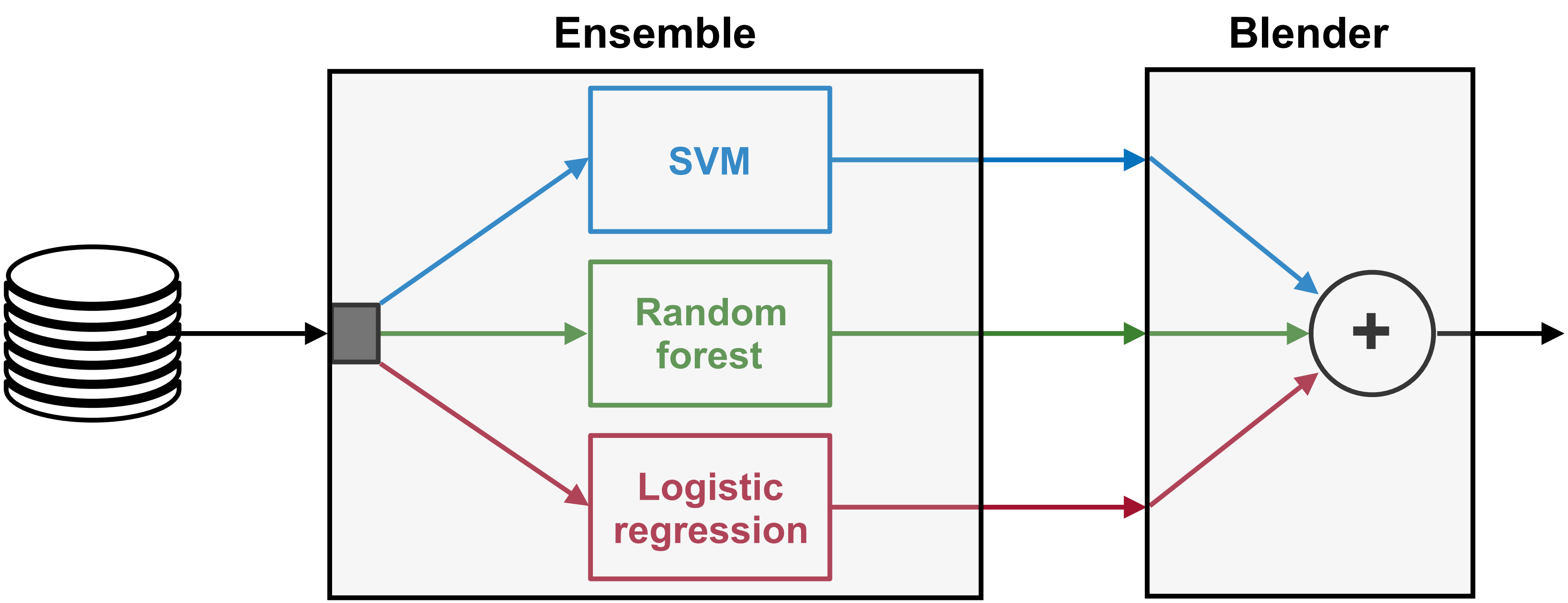
Variant 1. No splitting

Variant 2. Hold-out splitting

Variant 3. K-fold splitting

General idea

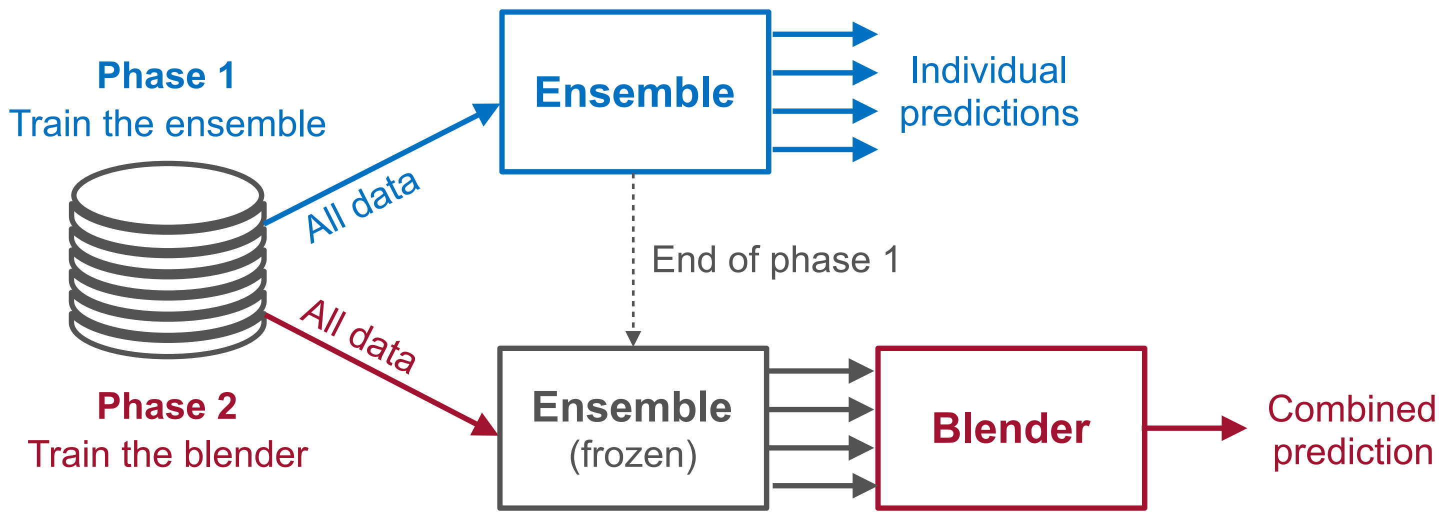
- **Stacking ➜** Multiple models are combined by another model
  - ***Regression* ➜** *Stacking works directly on predictions*
  - ***Classification* ➜** *Stacking works on class probabilities*



Simple stacking

1) No splitting

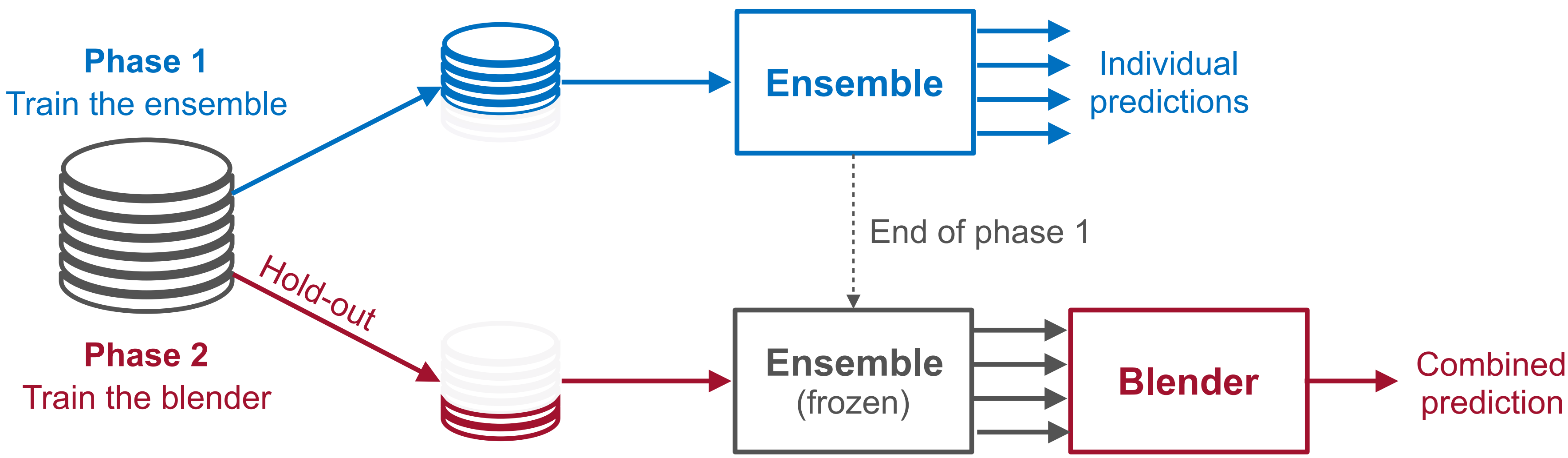
- *All data is used for training the ensemble*
- *All the ensemble predictions are used for training the blender*
- ***Drawback* ➜ *Prone to overfitting due to information leakage***



Hold-out stacking

2) Hold-out splitting

- *A fraction of data is used for training the ensemble*
- *The remaining data are used for training the blender*
- ***Drawback*** ➜ *Blender is trained on a small portion of data*



K-fold stacking

3) K-fold splitting

- *Data in K-1 folds are used for training the ensemble*
- *Unused data are fed to the trained models to gather predictions*
- *Repeat K times. All the predictions are used to train the blender*
- *The ensemble is eventually retrained on the whole data*

Dataset

Input	Output
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300

K-fold

Split 1

2104	400
1600	330
2400	369
1416	232
3000	540
1985	300

Train* & *Predict

Split 2

2104	400
1600	330
2400	369
1416	232
3000	540
1985	300

Train* & *Predict

Split 3

2104	400
1600	330
2400	369
1416	232
3000	540
1985	300

Train* & *Predict

Logistic stacking

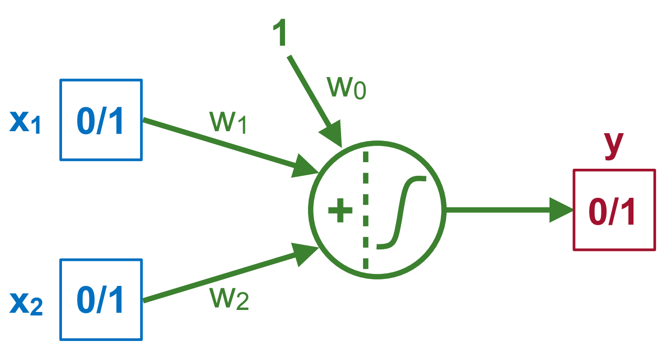
Boolean algebra

Logic gates

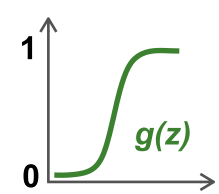
Stack of logic gates

Boolean algebra (1/2)

- Let’s consider the following problem
  - *Assume the inputs x_1 and x_2 are* ***binary values***
  - *Let f_θ be a* ***logistic model*** *with parameters $\theta = [w_0, w_1, w_2]$*
  - *By varying θ, which boolean functions can be reproduced by f_θ?*



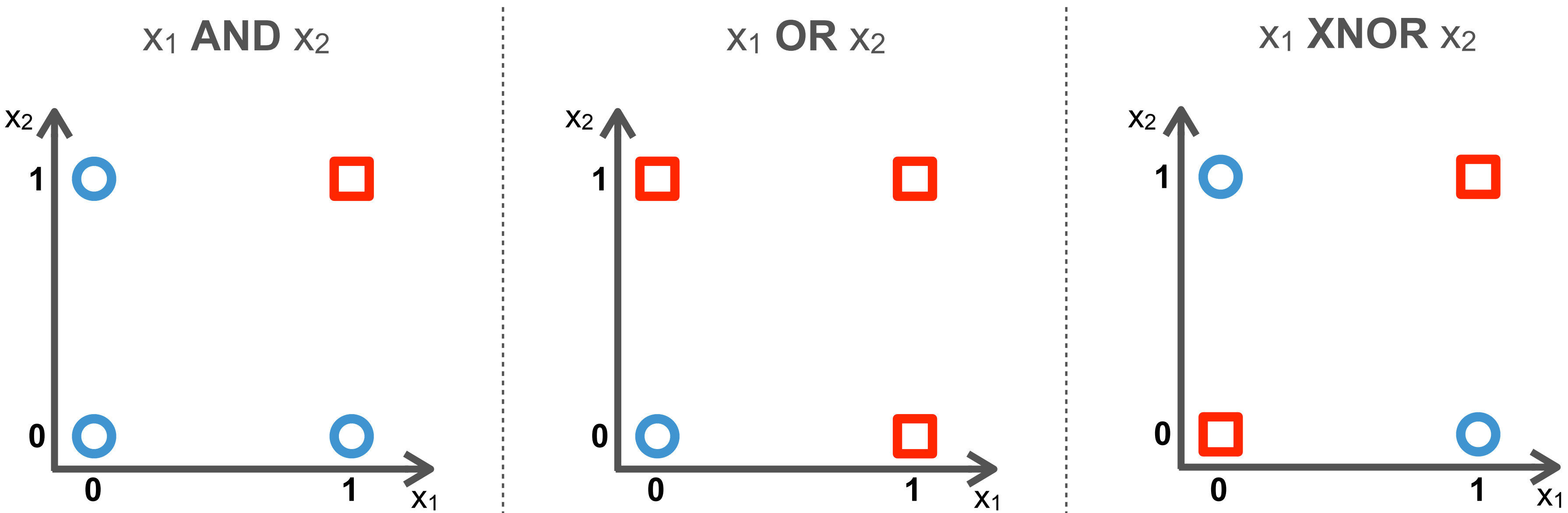
$y = g(w_0 + w_1x_1 + w_2x_2)$



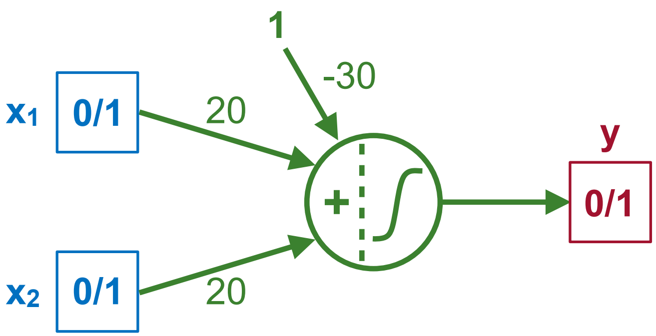
x_1	x_2	y
0	0	?
0	1	?
1	0	?
1	1	?

Boolean algebra (2/2)

- Why do we bother with Boolean algebra?
  - *Boolean functions represent some data grouped into 2 classes*
  - *They help us analyze the classification capacity of different models*



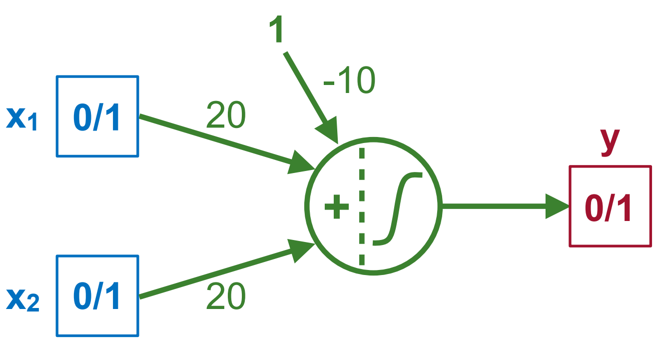
Logic gates (1/2)



x_1 AND x_2

$y = g(-30+20x_1+20x_2)$

x_1	x_2	y
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

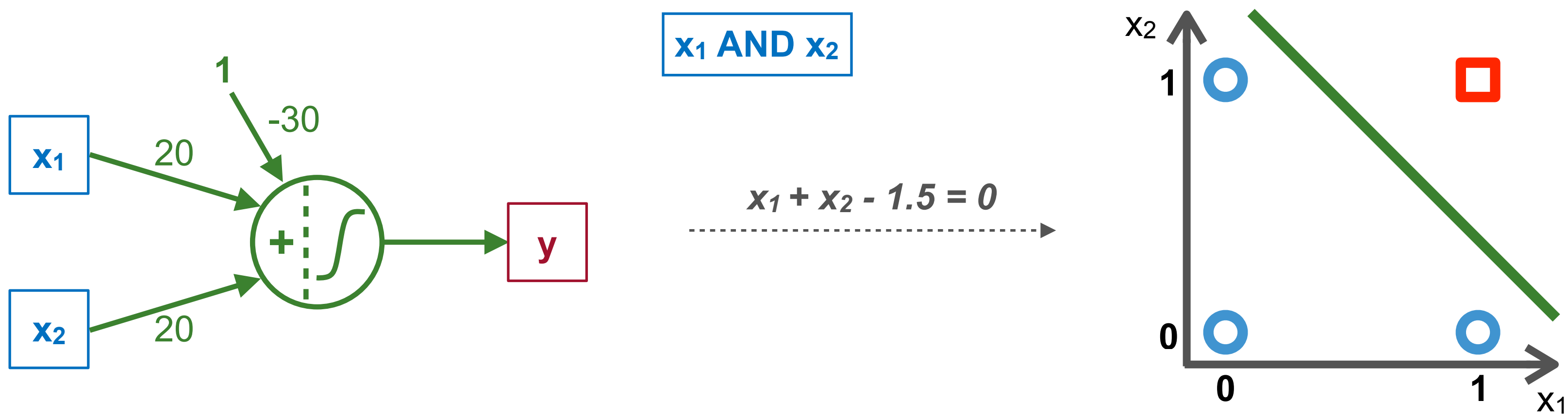


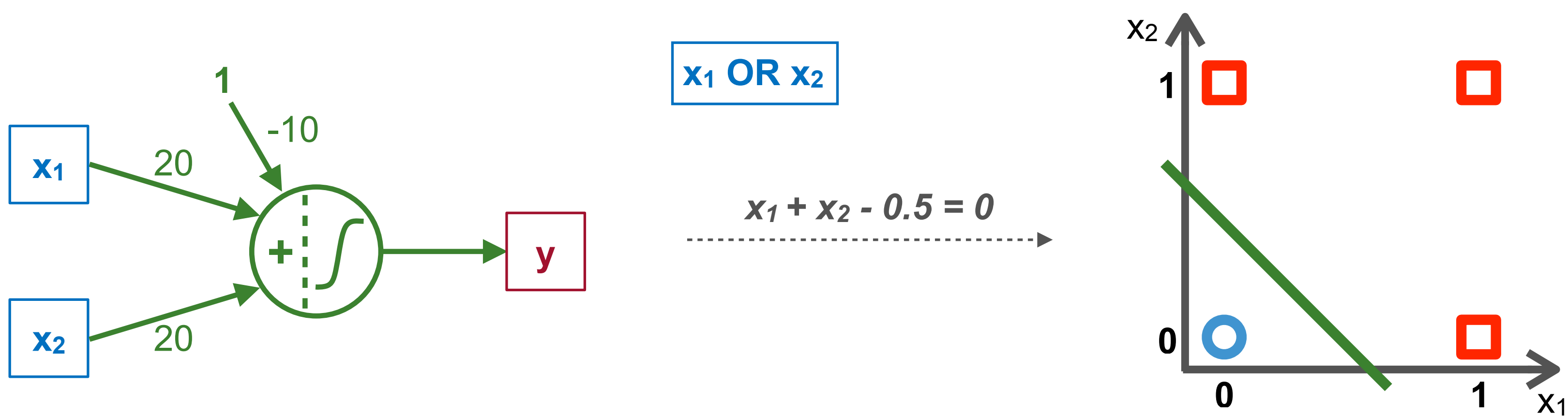
x_1 OR x_2

$y = g(-10+20x_1+20x_2)$

x_1	x_2	y
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

Logic gates (2/2)

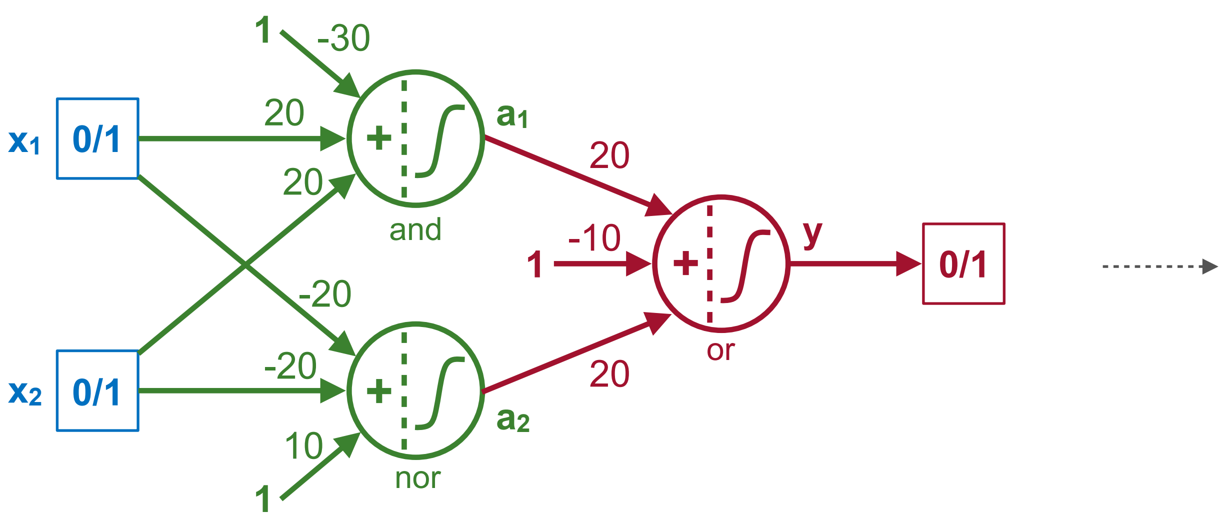




Stack of logic gates (1/2)

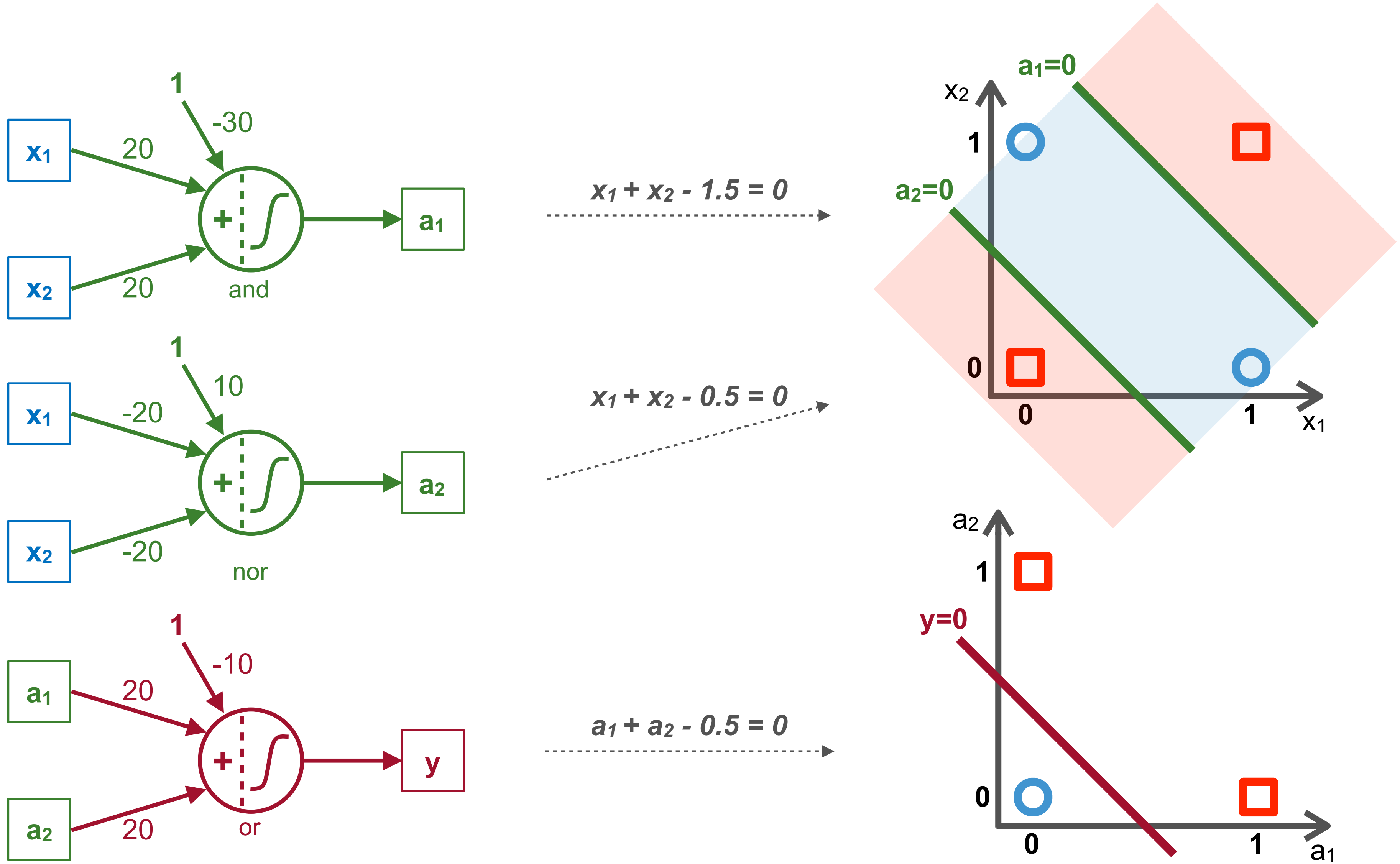
- Can we reproduce more complex Boolean functions?
  - *Logistic model can represent elementary gates (AND, OR, NOT).*
  - *Hence, any function can be represented by a network of logistic models.*

x_1 XNOR x_2 = (x_1 AND x_2) OR (x_1 NOR x_2)



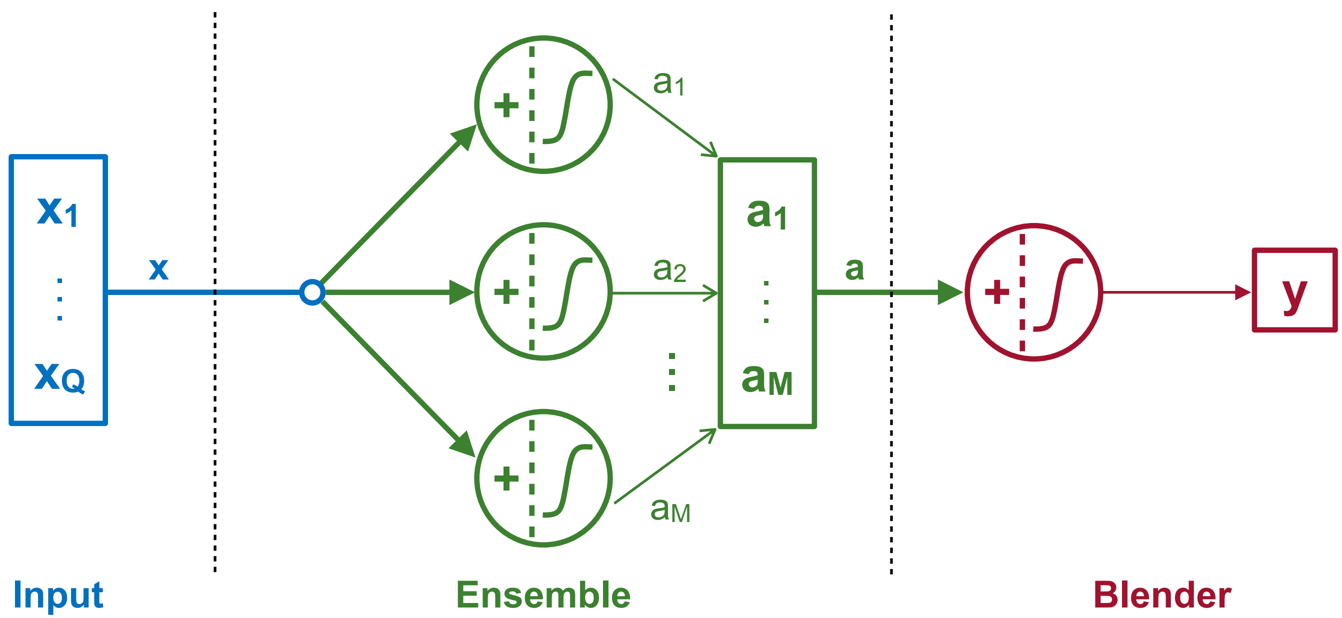
x_1	x_2	a_1	a_2	y
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Stack of logic gates (2/2)



Stack of logistic classifiers

- A stack of logistic classifiers yields a nonlinear model
  - ***Ensemble*** ➜ *The input is transformed into (learned) features*
  - ***Blender*** ➜ *The features are used for linear classification (or regression)*

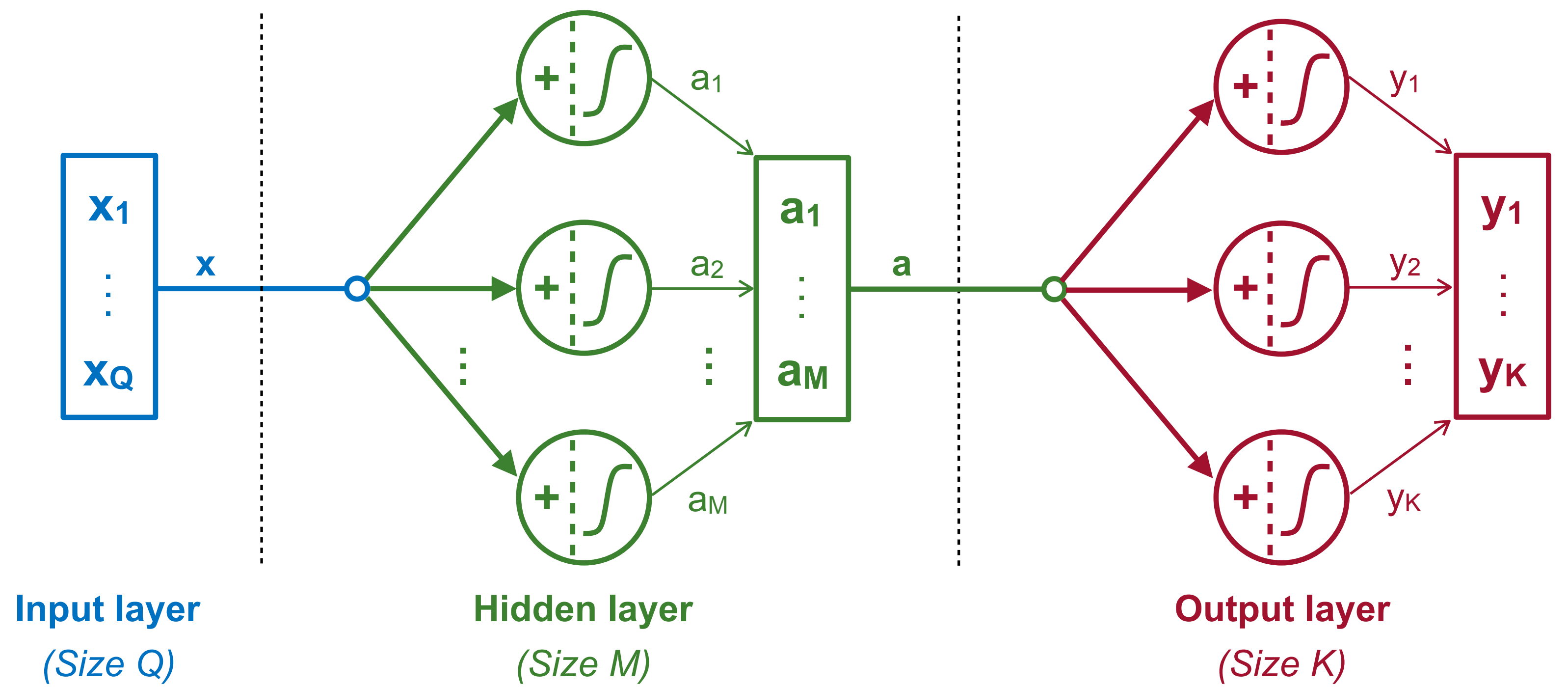


Two-layer neural networks

Hidden layer
Output layer
Forward propagation

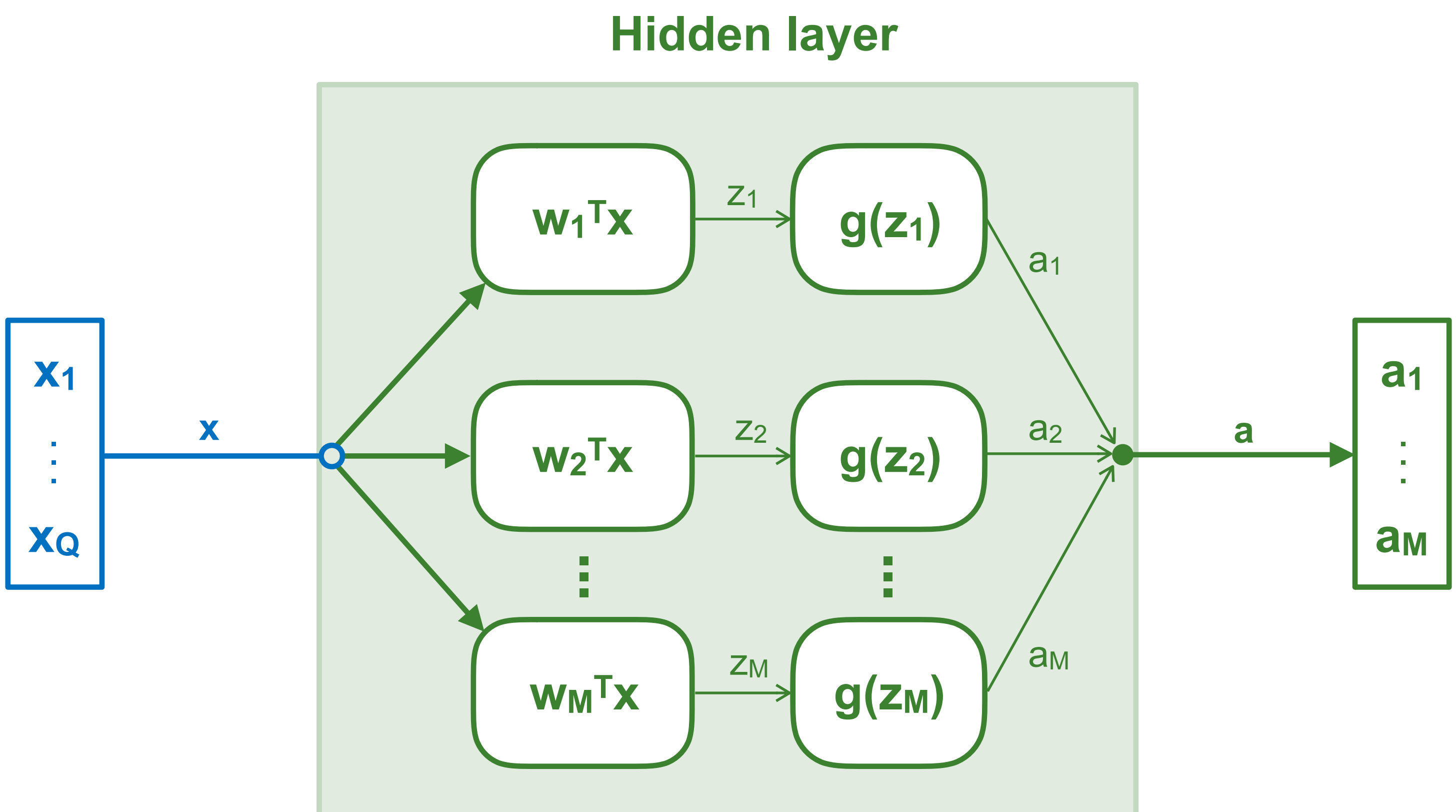
Two-layer neural networks

- A **neural network** consists of units organized in layers
  - ***Feed-forward*** ➔ *Special case when connections don't form cycles*
  - ***Two-layer network*** ➔ *The simplest instance of a feed-forward network*



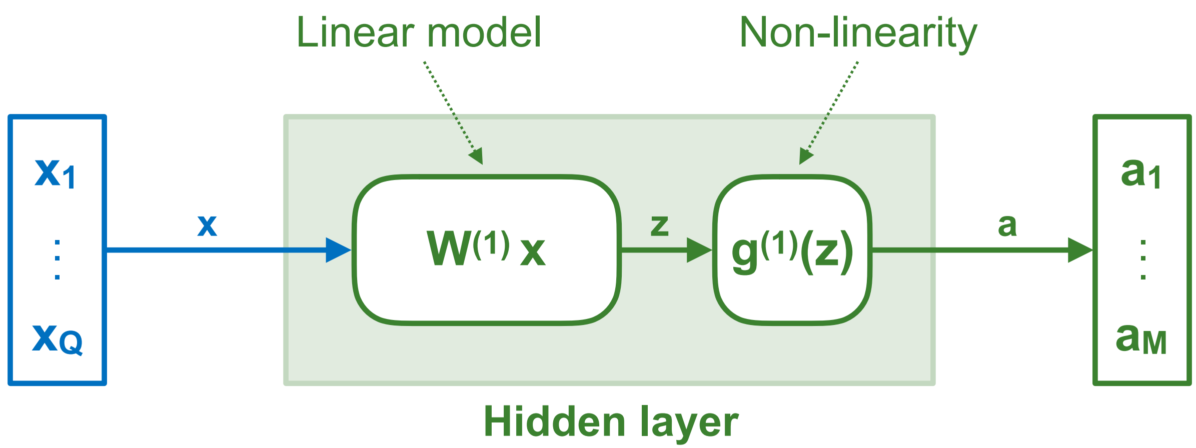
Hidden layer (1/3)

- The **hidden layer** is formed by many "parallel" units



Hidden layer (2/3)

- **Hidden layer** ➔ Linear model $\mathbf{W}^{(1)}$ + Non-linearity $\mathbf{g}^{(1)}$



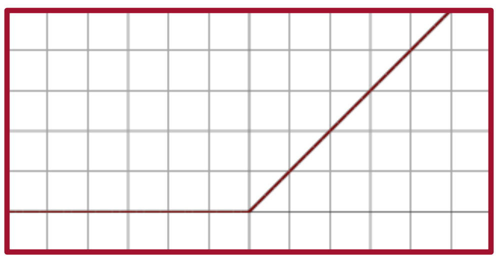
$$\mathrm{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_Q \end{bmatrix} \qquad \mathrm{W}^{(1)} = \begin{bmatrix} - \mathrm{w}_1^\top - \\ \vdots \\ - \mathrm{w}_M^\top - \end{bmatrix} \qquad \mathrm{g}^{(1)}(\mathrm{z}) = \begin{bmatrix} 1 \\ g(z_1) \\ \vdots \\ g(z_M) \end{bmatrix}$$

$$\boxed{\mathrm{a} = \mathrm{g}^{(1)}\big(\mathrm{W}^{(1)}\mathrm{x}\big)}$$

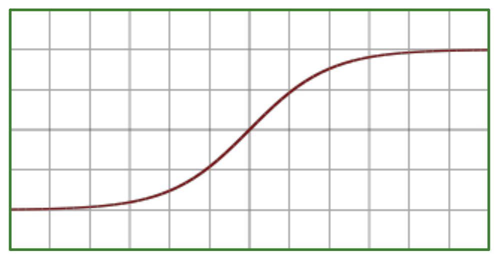
Hidden layer (3/3)

- Different choices for the function $\mathbf{g}^{(1)}$ are possible

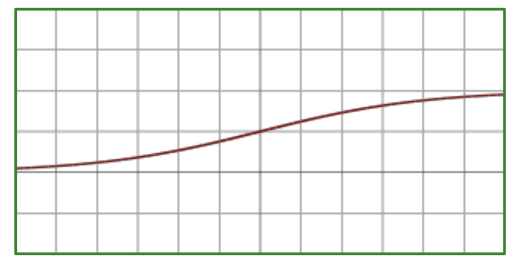
ReLU



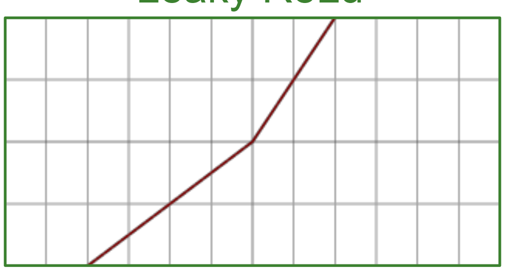
Tanh



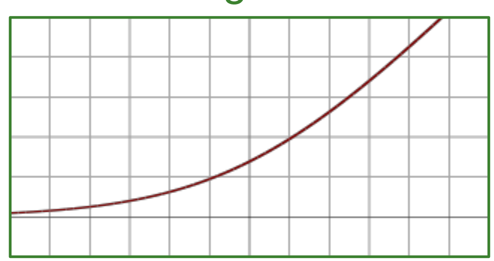
Sigmoid



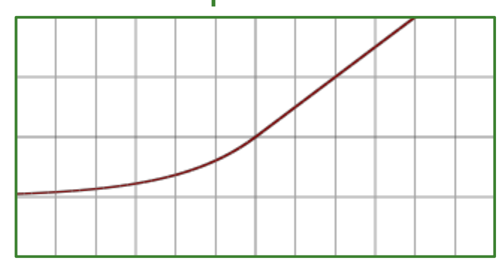
Leaky ReLu



Logistic

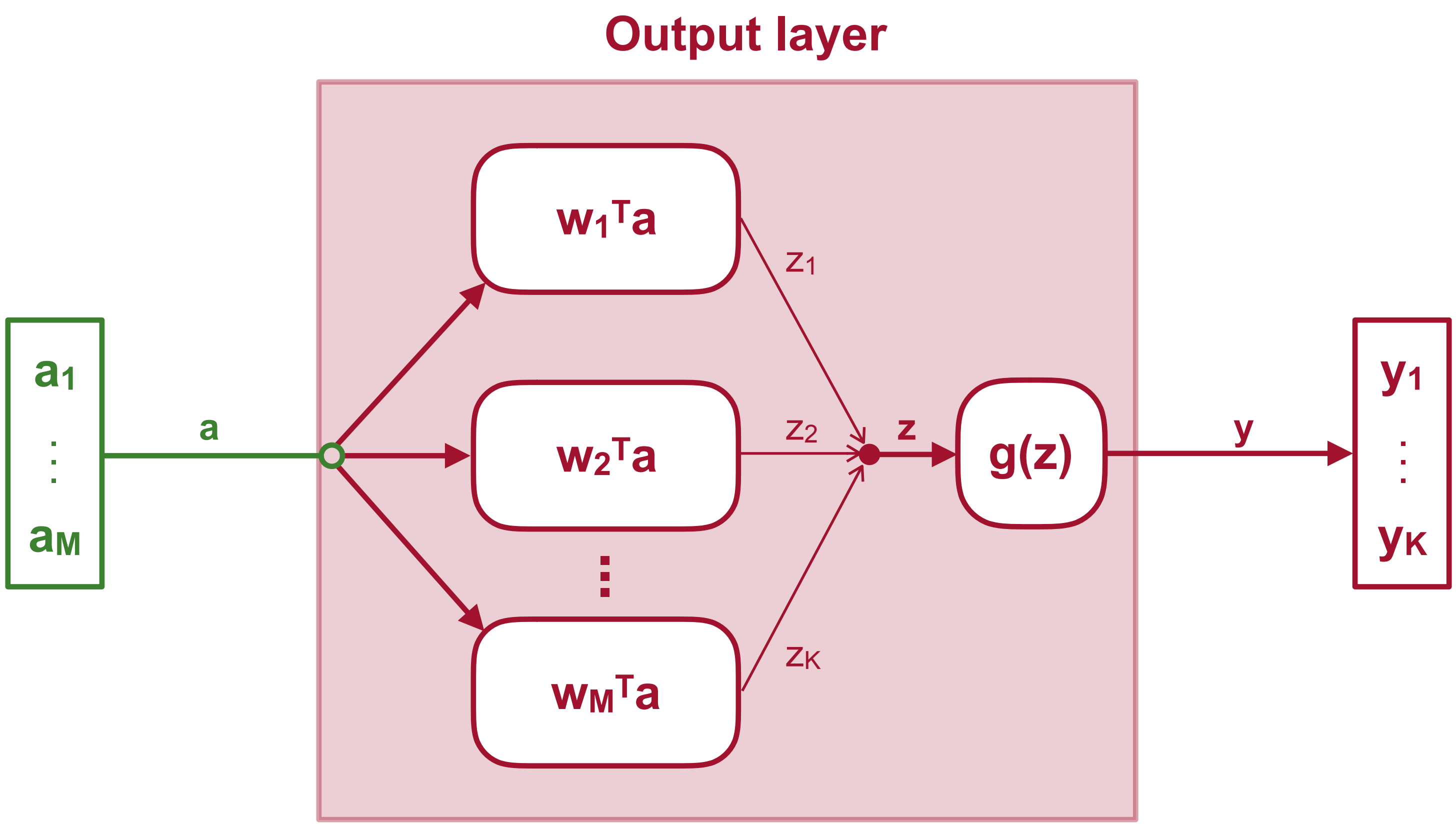


Exp-linear



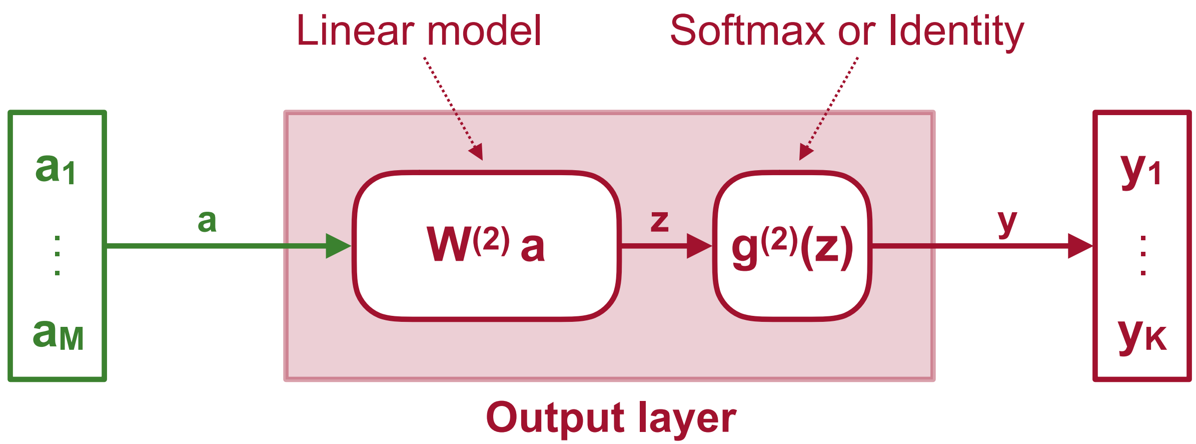
Output layer (1/2)

- The **output layer** is either a **regressor** or a **classifier**



Output layer (2/2)

- **Output layer** ➔ Linear model $\mathbf{W}^{(2)}$ + Softmax/Identity $\mathbf{g}^{(2)}$



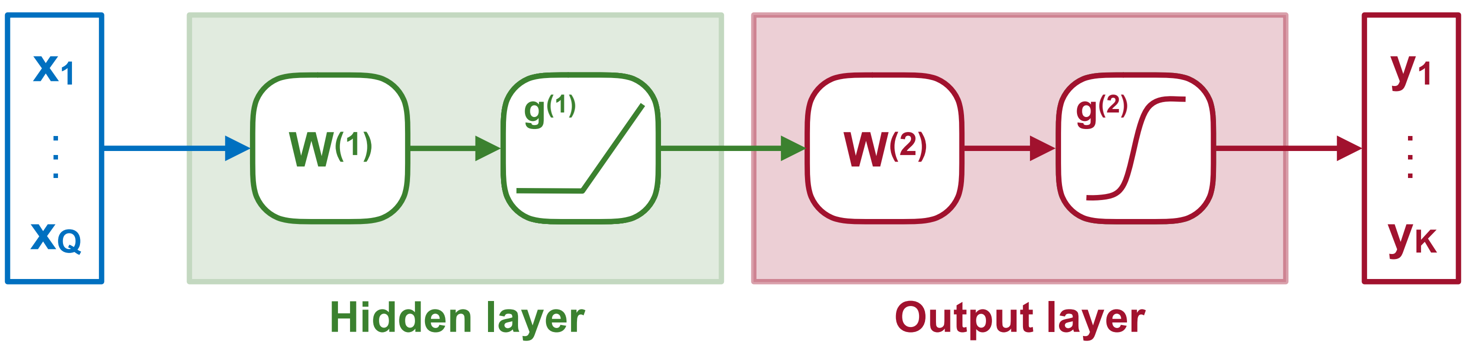
$$\mathrm{W}^{(2)} = \begin{bmatrix} -\mathrm{w}_1^\top - \\ \vdots \\ -\mathrm{w}_K^\top - \end{bmatrix} \qquad \mathrm{g}^{(2)}(\mathrm{z}) = \begin{bmatrix} \sigma_1(\mathrm{z}) \\ \vdots \\ \sigma_K(\mathrm{z}) \end{bmatrix}$$

$$\boxed{\mathrm{y} = \mathrm{g}^{(2)}\big(\mathrm{W}^{(2)}\mathrm{a}\big)}$$

Forward propagation (1/2)

- Neural network with **2 layers**
  - ***Hidden layer*** ➔ *The input is transformed into (learned) features*
  - ***Output layer*** ➔ *The features are used for regression or classification*

$$f_\theta(\mathrm{x}) = \mathrm{g}^{(2)}\big(\mathrm{W}^{(2)}\mathrm{g}^{(1)}\big(\mathrm{W}^{(1)}\mathrm{x}\big)\big)$$



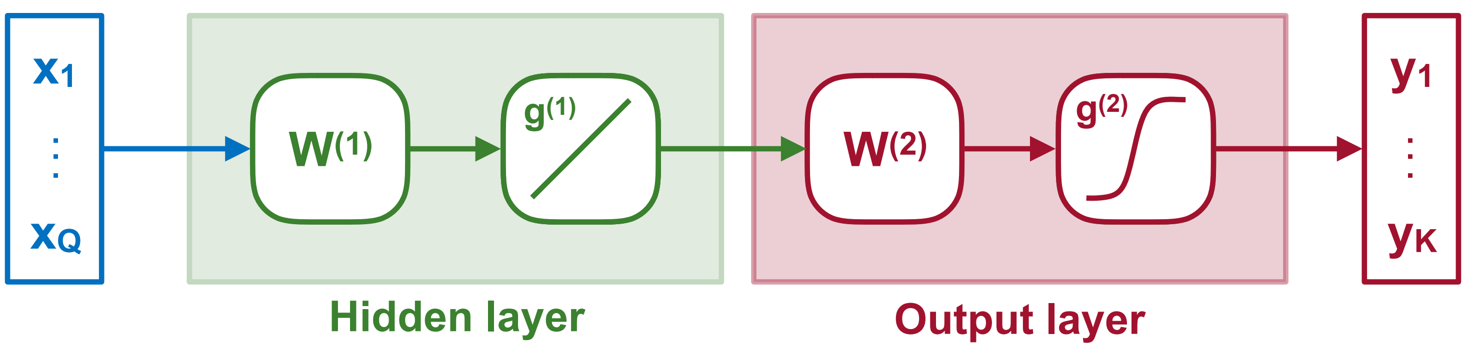
Forward propagation (2/2)

- The **non-linearity** of **$g^{(1)}$** is essential for neural networks
  - *Otherwise, the network behaves like a **linear regressor/classifier***

$$g^{(1)}(z) = z \quad \Rightarrow \quad f_\theta(x) = g^{(2)}\left(W^{(2)}W^{(1)}x\right) = g^{(2)}(Wx) \quad \longrightarrow \quad \text{linear}$$

This behaves like a linear model !!!

(with more parameters than strictly necessary)



Neural network training

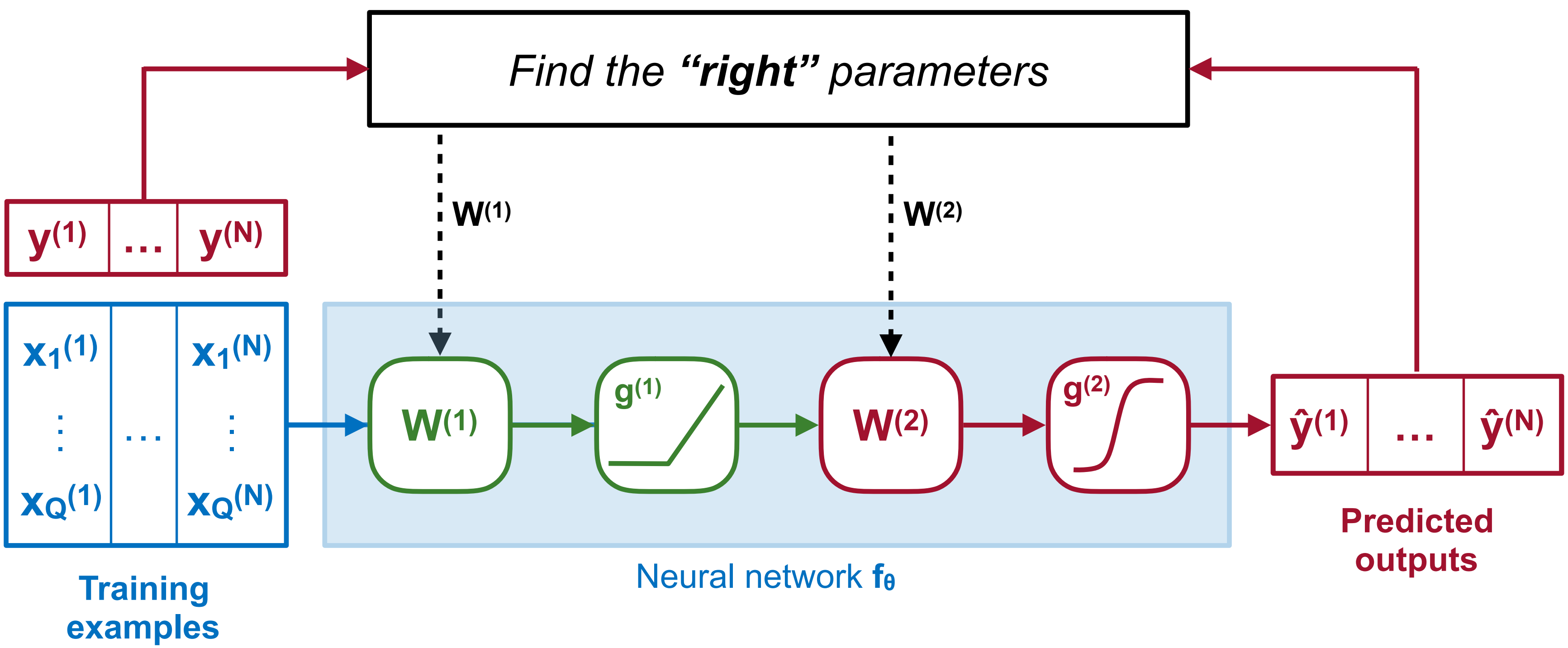
Cost function

Practical advice

Hyper-parameters

Cost function for neural networks (1/3)

- Our goal is to **learn** the prediction $\mathbf{f}_\theta$ from training data
  - *This amounts to finding the "right values" for parameters* $\boldsymbol{\theta} = (W^{(1)}, W^{(2)})$



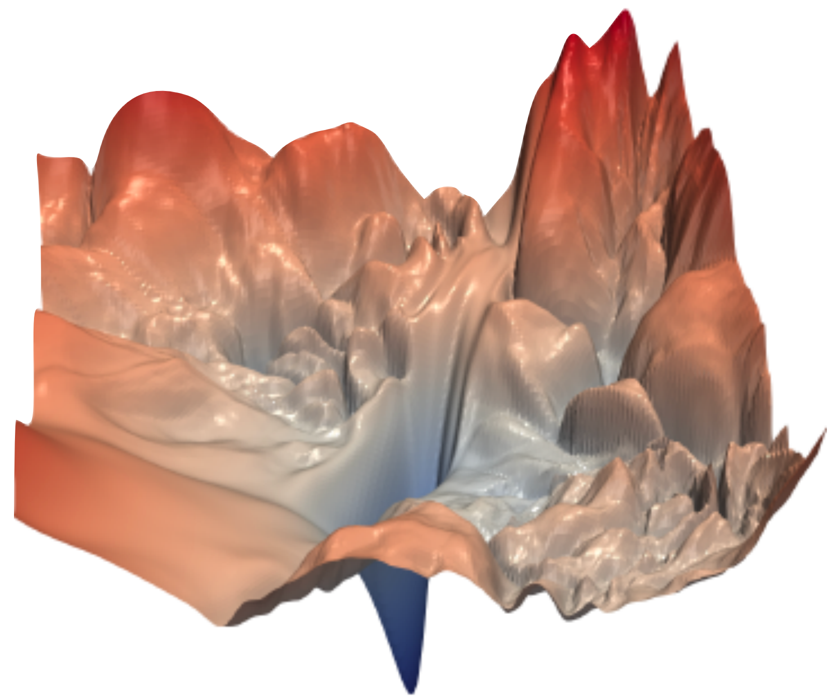
Cost function for neural networks (2/3)

- How to choose the **"right values"** for parameters **θ** ?
  - *We select* ***θ*** *such that the* ***model*** f_θ ***is fitted*** *to the training data*

$$\widehat{\theta} = \arg\min_{\theta} \sum_{n=1}^{N} C\Big(f_\theta(\mathrm{x}^{(n)}), \mathrm{y}^{(n)}\Big)$$

Prediction Output

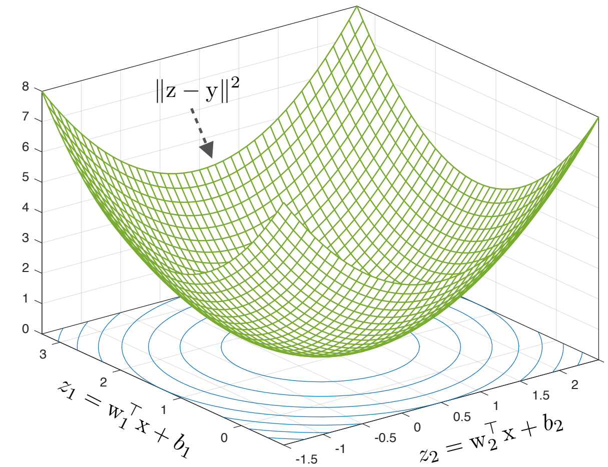
Cost function



Cost function for neural networks (3/3)

- Euclidean distance for **regression**

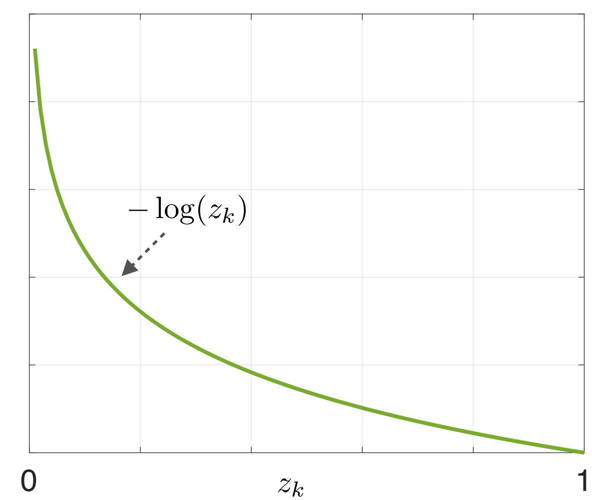
$$C\big(f_\theta(\mathrm{x}), \mathrm{y}\big) = \big\| f_\theta(\mathrm{x}) - \mathrm{y} \big\|^2$$



- Cross-entropy for **classification**

$$C\big(f_\theta(\mathrm{x}), \mathrm{y}\big) = -\mathrm{y}^\top \log\big(f_\theta(\mathrm{x})\big)$$

One-hot encoding



Gradient descent (1/2)

- How to **minimize the cost J(θ)** on the training set ?
  - *We find the optimal **θ** through **gradient descent***

Current solution

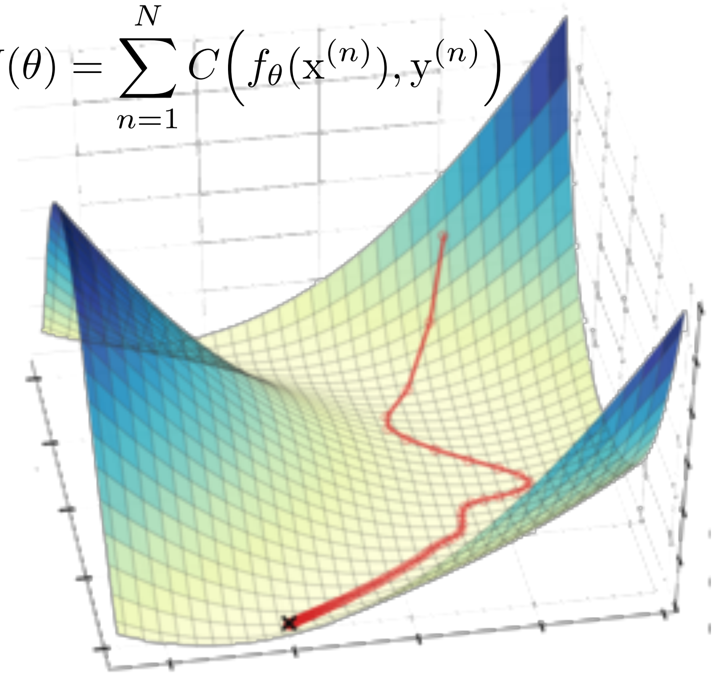
$$\theta^{[i+1]} = \theta^{[i]} - \alpha_i \nabla J(\theta^{[i]})$$

Step-size

Gradient in current solution

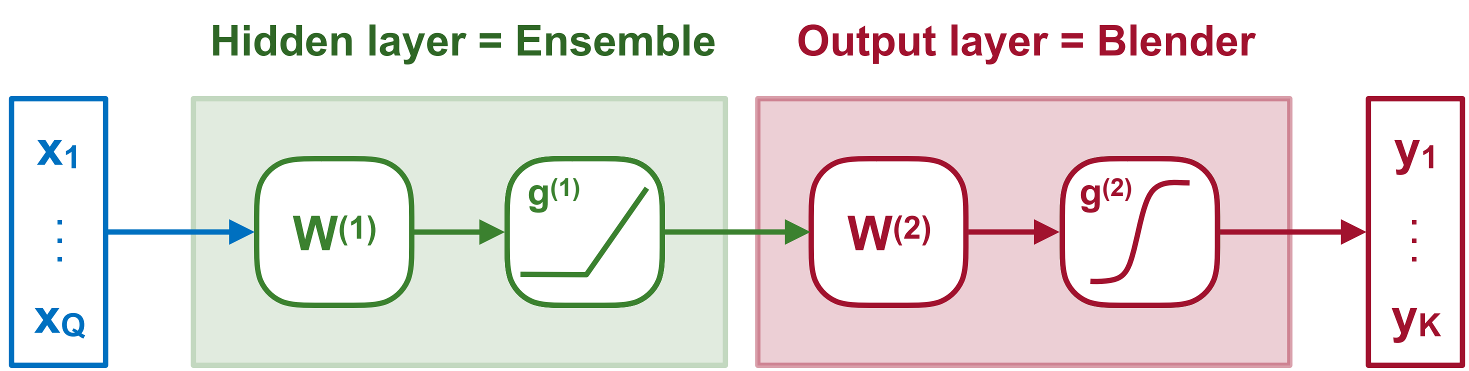
Updated solution

$$J(\theta) = \sum_{n=1}^{N} C\Big(f_\theta(\mathrm{x}^{(n)}), \mathrm{y}^{(n)}\Big)$$



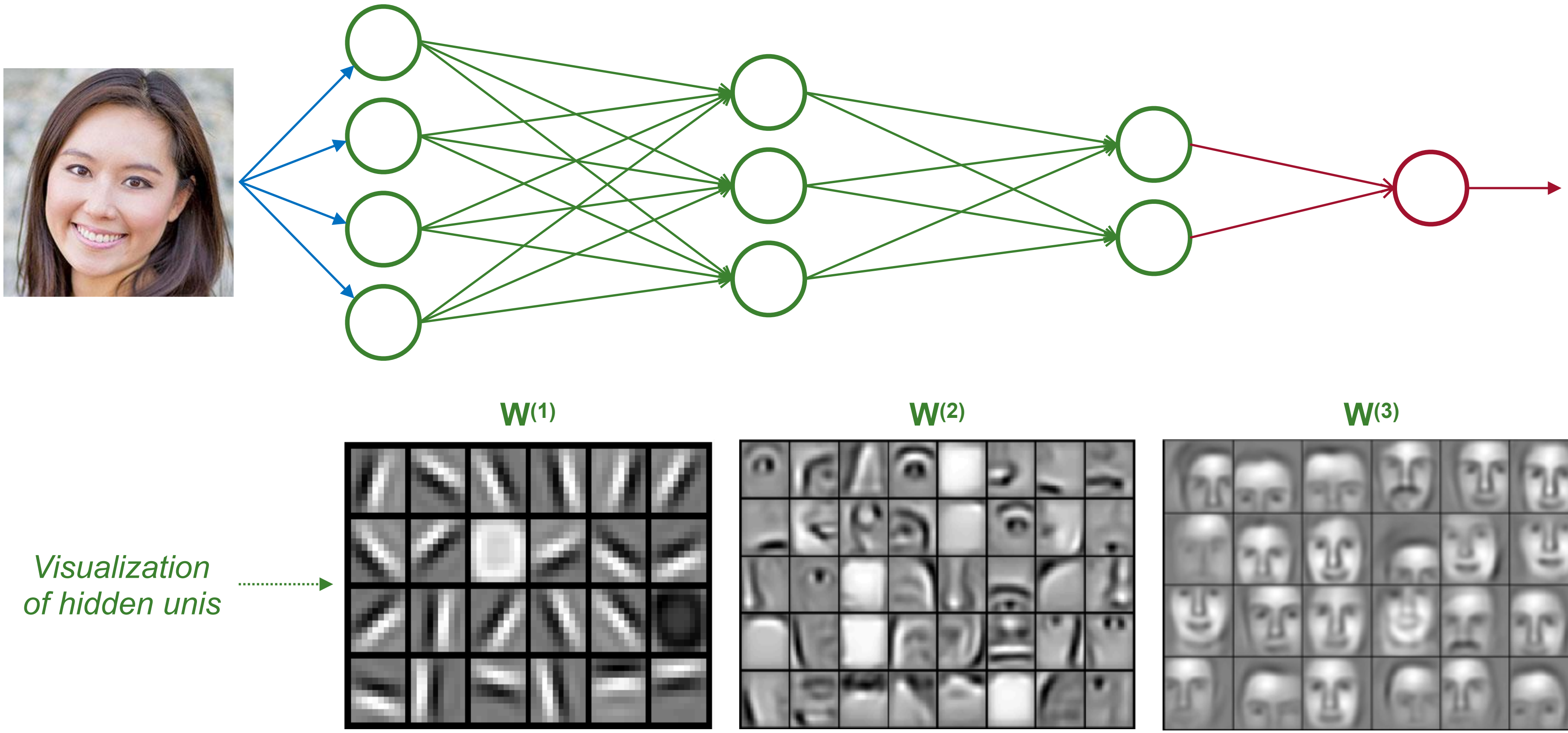
Gradient descent (2/2)

- The neural network parameters are **randomly initialized**
  - *If the parameters were initialized to zero, each neuron in the hidden layer would perform the same computation…*
  - *… so even after multiple iterations of gradient descent, each neuron in the layer would be computing the same thing as other neurons.*
  - ***Recall* ➜ Random initialization introduces *diversity* in the ensemble.**



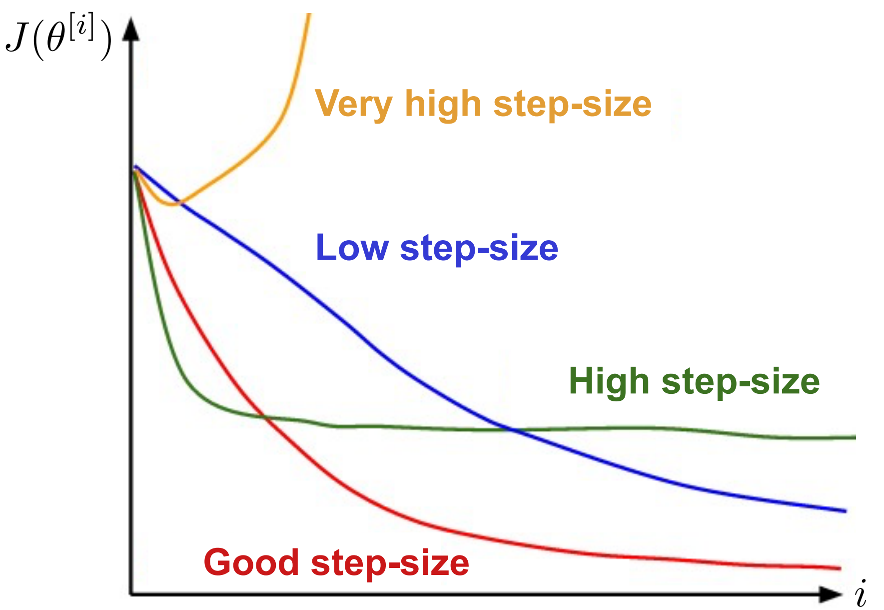
Why neural networks?

- Neural networks can learn a **hierarchical representation**



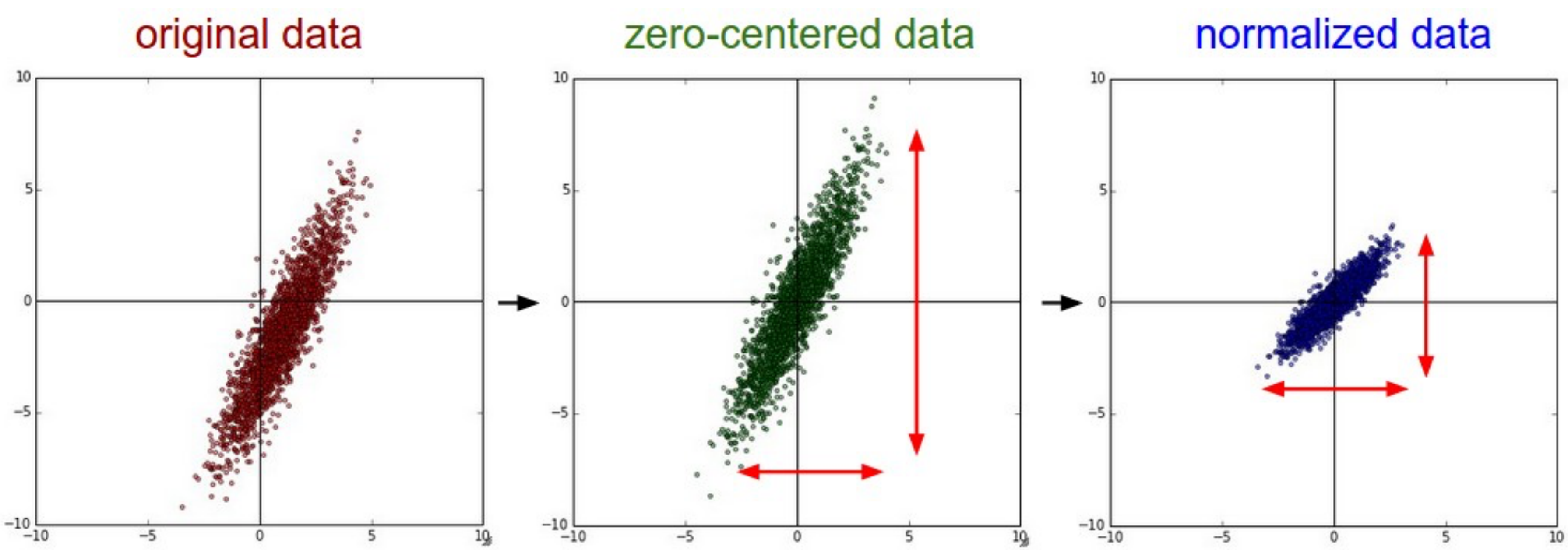
Practical advice (1/3)

- **Advice ➜** Track the cost function during training
  - *Compute* $J(\theta^{[i]})$ *at each iteration* i *and save/plot its value*
  - *The shape of* $J(\theta^{[0]}),\dots,J(\theta^{[i]})$ *will tell you about the* ***step-size***



Practical advice (2/3)

- **Advice ➜** Normalize data at the network's input
  1) *Subtract the mean across every individual feature in the data*
  2) *Divide each feature by its standard deviation (after mean subtraction)*



Practical advice (3/3)

- **Advice ➜** Train an ensemble of networks

1) ***Same model, different initialization.***
   - Use cross-validation to determine the best hyper-parameters, then train several models with the same hyper-parameters, but with different random initialization.

2) ***Top models discovered during cross-validation.***
   - Use cross-validation to determine the best hyper-parameters, then pick the models having the best-performing sets of hyper-parameters.

3) ***Different checkpoints of a single model.***
   - If training is very expensive, take different checkpoints of a single network over time. For example, pick a network after a fixed number of epochs. Alternatively, start with a large step-size and a decaying schedule, train the network for a fixed time, and restart with a large step-size after saving the network. Another way is to maintain a running average of network parameters during training.

Conclusion

Algorithms for supervised learning

Nonlinear models

Practical advice

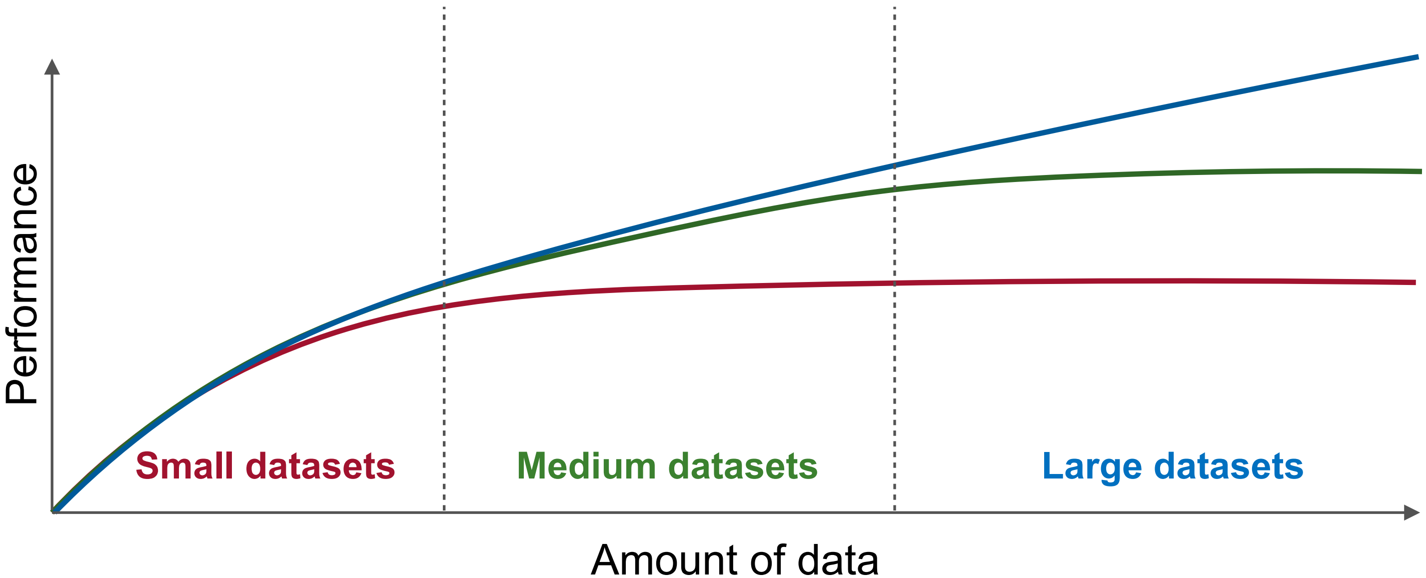
Algorithms for supervised learning

- **Linear models**
  - *Linear regression*
  - *Logistic regression*
  - *Support vector machine*

- **Nonlinear models**
  - *Linear/logistic regression with polynomial features*
  - *SVM with nonlinear kernels (polynomial, Gaussian, …)*
  - *Decision trees*
  - *Ensemble methods*
  - *Neural networks*

Nonlinear models

- How to get **better performance** out of machine learning?
  - *More data for training*
  - *More complexity in nonlinear models*



Deep learning
- Big neural nets
- Custom layers
- GPUs

Medium neural nets
- Dropout & Batch norm.
- Convolutional nets
- Data augmentation
- Recurrent nets

Traditional methods
- Linear models
- SVM & kernels
- Tree ensembles
- Small neural nets
- …

Practical advice

- **Q** (features) **>> N** (examples) ➜ Q = 10’000 & N < 10’000
  - *Risk of over-fitting*
  - *Use linear models with regularization*

- **Q << N** & **N** intermediate ➜ Q < 1’000 & N < 100’000
  - *Use “light” nonlinear models: SVM, ensembles, small-medium nets*

- **Q << N** & **N** large ➜ Q < 10’000 & N > 100’000
  - *If you have a GPU, use deep neural networks*
  - *Otherwise, learning will be too slow. If this is the case, manually create or add more features, and use light nonlinear models*